A Formal Foundation for ODRL

Riccardo Pucella and Vicky Weissman

Department of Computer Science
Cornell University
Ithaca, NY 14853
{riccardo,vickyw}@cs.cornell.edu

Abstract. ODRL is a popular XML-based language for stating the conditions under which resources can be accessed. The language is underspecified, and can be reasonably interpreted in a number of ways. To remove the ambiguity, we propose a formal semantics for a representative fragment of the language. We also define when a permission is implied by a set of ODRL statements.

1 Introduction

ODRL, the Open Digital Rights Language [6], is an XML-based language for stating the conditions under which resources can be accessed legitimately. The language has been endorsed by nearly twenty organizations including:

- Nokia, a multi-industry conglomerate focused on mobile communications;
- the DAFNE project (District Architecture for Networked Editions), a research project funded by the Italian Ministry of Education, University and Research to develop a prototype of the national infrastructure for electronic publishing in Italy;
- the RoMEO Project (Rights MEtadata for Open archiving), created to investigate rights management of "self-archived" research in the United Kingdom academic community.

The complete list of supporters can be found at `www.odrl.net`; however, this small sample illustrates the widespread impact that ODRL has on rights management. The success of these projects depends on ODRL.

Unfortunately, ODRL does not have formal semantics. The meaning of the language's syntax is described in English and, as a result, agreements written in ODRL are open to interpretation. For example, ODRL supports the statement 'the group comprised of Alice and Bob is permitted to withdraw money from the bank account BA'. However, it is not clear from the English definition of groups who may access the account. One reasonable interpretation is that either Alice or Bob may withdraw money. Another option is that Alice and Bob together may withdraw the money, although neither has permission to do so alone. In fact, it even seems somewhat plausible that the group comprised of Alice and Bob refers to some third individual, perhaps someone who Alice and Bob both trust. This example shows that the ODRL specification alone is not sufficient to interpret ODRL agreements correctly (i.e., as intended by the language designers).

To address this problem, we give a formal semantics to ODRL. Our particular interest is in understanding whether or not an agreement implies a permission (or a prohibition). We propose a translation from the key components in ODRL to formulas in a fragment of many-sorted first-order logic with equality. Our semantics is consistent with the ODRL specification and discussions that we had with one of the language designers. There is not much more that can be done to verify our translation. In particular, we cannot prove that our semantics is correct, because the language is ambiguous. Also, we cannot compare our approach to others, because to the best of our knowledge, we are the first to give formal semantics to ODRL. Therefore, we do not view our approach as the final say in ODRL semantics; instead it is a first step towards an understanding of the language's subtleties and towards a universally recognized, formal foundation.

The rest of this abstract is organized as follows. In the next section, we present a representative fragment of ODRL. In Section 3, we give a semantics to this fragment by translating expressions in the language to formulas in first-order logic. For reasons of space, a detailed discussion of the implications of our semantics is left to the full paper.

2 The ODRL Language

In this section we present a syntax for a representative fragment of ODRL. Because ODRL is an XML-based language, the syntax is convenient for automatic processing, but is somewhat verbose for human readers. To make our discussion more concise and intuitive, we present a modified version of the ODRL syntax. As we point out in Appendix A, there is an obvious correspondence between our syntax and ODRL's (when restricted to the appropriate fragment). We consider only a fragment of the language for brevity, as well as ease of exposition; we expect that our syntax and semantics can be readily extended to include the entire language. A summary of the main differences between our syntax and ODRL's is given at the end of this section.

The central construct of ODRL is an agreement. An agreement gives the policies (i.e., rules) under which a principal $prin_o$ allows another principal $prin_u$ to use an asset a, which is presumably owned by $prin_o$. Typically, $prin_o$ is called the agreement's owner and $prin_u$ is called the agreement's user. For example, suppose that an agreement says *the Disney Corp. allows Alice to play 'Finding Nemo', if she first pays five dollars*. Then, the owner is the Disney Corp., the user is Alice, the asset is 'Finding Nemo', and the policy is *the user may play the asset, if she pays five dollars*. We assume that the application provides a set $Assets$ of assets and a set $Subjects$ of subjects. A primitive principal in ODRL is a subject; a principal can also be a group (i.e., set) of principals. The permissions given to a group are given to each member of the group. More generally, a group in ODRL does not have an identity beyond its members. Thus, in the example presented in the introduction, both Alice and Bob may (individually) withdraw money from the bank account BA, if the group comprised of Alice and Bob may withdraw money from BA. We formally define the syntax of agreements and principals as follows, where ps is a policy set (defined later in the section).

Agreements:

agr ::=	agreement

$$
\begin{array}{ll}
\quad \textbf{agreement} & \\
\quad\quad \textbf{between } prin_o \textbf{ and } prin_u & \\
\quad\quad \textbf{about } a & \\
\quad\quad \textbf{with } ps & \\
prin ::= & \text{principal} \\
\quad s & \quad \text{individual} \\
\quad \{prin_1, \ldots, prin_m\} & \quad \text{group} \\
a \in Assets & \text{asset} \\
s \in Subjects & \text{subject}
\end{array}
$$

An agreement refers to a policy set. A policy set is any number of prerequisites and a policy. Roughly speaking, if all of the prerequisites are met, then the policy holds and is taken into consideration when answering questions about what is and what is not permitted. In addition, a policy set can be tagged as being *exclusive*. An exclusive policy set indicates that only the agreement's user (either the subject or the members of the set) may perform the actions regulated by the policy set. Every other principal is forbidden from doing the regulated actions. Policy sets are closed under conjunction, disjunction, and exclusive-or. Conjunctions allow for more than one policy set to appear in an agreement. Disjunctions are useful, because they allow a degree of flexibility when honoring agreements. For example, suppose that Alice has the right to print a file on either printer-1, printer-2, or printer-3. Under the corresponding agreement, Alice may always print the file, but the choice of printers is not hers; it might change with each printing based on the load of the printers or some other criteria. As for the exclusive-or, this construct is well-suited to agreements such as *Alice may view the low resolution version of the movie five times or the high resolution twice*. We define the syntax of a policy set as follows, where p is a policy (defined later in the section).

Policy Sets:

$$
\begin{array}{ll}
ps ::= & \text{policy set} \\
\quad prq_1 \ \ldots \ prq_m \longrightarrow p & \quad \text{primitive policy set } (m \geq 0) \\
\quad prq_1 \ \ldots \ prq_m \longmapsto p & \quad \text{primitive exclusive policy set } (m \geq 0) \\
\quad \mathsf{and}[ps_1, \ldots, ps_m] & \quad \text{conjunction } (m \geq 1) \\
\quad \mathsf{or}[ps_1, \ldots, ps_m] & \quad \text{disjunction } (m \geq 1) \\
\quad \mathsf{xor}[ps_1, \ldots, ps_m] & \quad \text{exclusive disjunction } (m \geq 1)
\end{array}
$$

We abbreviate a policy set of the form $\longrightarrow p$ (i.e., one with no prerequisites) as p.

A policy is a set of prerequisites and an action. If all of the prerequisites are met, then the policy says that the agreement's user may perform the action to the agreement's asset. As with policy sets, a policy can be a conjunction, disjunction, or exclusive-or of policies. We tag policies with an identifier (taken from a set $PolIds$). It will be necessary, when dealing with counts and payments, to identify the policy to which the count or payment apply. For example, if Bob gives the bank \$500, we need to know if the money should go towards the policy concerning the car payment or the one concerning the mortgage. We will sometimes omit the identifier if it is not relevant to our examples. Also, we abbreviate $\Longrightarrow act$, which is a policy that does not have prerequisites, as act. Finally, for the purposes of this discussion, we restrict the set of actions to play, print, and display. The syntax for policies and actions are given below.

Policies:

p ::=	policy
$prq_1 \ \ldots \ prq_m \Longrightarrow_{id} act$	primitive policy ($m \geq 0$)
$\mathsf{and}[p_1, \ldots, p_m]$	conjunction ($m \geq 1$)
$\mathsf{or}[p_1, \ldots, p_m]$	disjunction ($m \geq 1$)
$\mathsf{xor}[p_1, \ldots, p_m]$	exclusive disjunction ($m \geq 1$)
act ::=	action
play	play asset
print	print asset
$\mathsf{display}$	display asset
$id \in PolIds$	policy identifier

A prerequisite is either a constraint, a requirement, or a condition. Constraints are facts that are outside the user's influence. For example, there is nothing that Alice can do to meet the constraint *the user is Bob*. Requirements are facts that are typically within the user's power to meet. For example, Alice can meet the requirement *the user has paid five dollars* by making the payment. Although the distinction between constraints and requirements is not relevant when answering questions about what is and is not permitted, we remark that it is useful for other types of queries. In particular, it provides key information when determining what a principal can do to obtain a permission. Finally, conditions are constraints that must *not* hold. The statement *the user is not Bob* is an example of a condition. We now consider each of these prerequisites in turn.

Our fragment of ODRL includes two primitive forms of constraints, users and counts. A user constraint is a principal $prin$; the constraint is met by every subject in $prin_u$ (the agreement's user) that is also a subject in $prin$. A count constraint is parameterized by an integer that indicates the number of times a policy or policy set is used to justify an action performed by some principal $prin_c$. If a user constraint $prin$ appears alongside the count constraint, then $prin_c$ is $prin$, otherwise, it is $prin_u$ (the agreement's user).

Example 2.1. Consider an agreement in which the user is $\{\text{Alice}, \text{Bob}, \text{Charlie}\}$. The policy

$$\mathsf{count}[5] \Longrightarrow \mathsf{print}$$

says that if Alice, Bob, and Charlie have used the policy to justify the print action a, b, and c times respectively, then any of them may do so again if $a + b + c \leq 5$. On the other hand, the policy

$$\text{Alice} \ \ \mathsf{count}[5] \Longrightarrow \mathsf{print}$$

says that if Alice has used the policy to justify the print action a times, she may do so again if $a \leq 5$. A count constraint that appears in a policy set is interpreted in a similar way. The policy set

$$\mathsf{count}[5] \longrightarrow \mathsf{and}[\mathsf{print}, \mathsf{display}]$$

says that if Alice, Bob, and Charlie have used the policy to justify the print action a_p, b_p, and c_p times respectively, and have used the policy to justify the display action a_d, b_d, and c_d times respectively, then any of them may display or print again if $a_p + b_p + c_p + a_d + b_d + c_d \leq 5$. On the other hand, the policy set

$$\{\mathrm{Alice}, \mathrm{Bob}\}\ \ \mathsf{count}[5] \longrightarrow \mathsf{and}[\mathsf{print}, \mathsf{display}]$$

says that if Alice and Bob have used the policy to justify the print action a_p and b_p times respectively, and have used the policy to justify the display action a_d and b_d times respectively, then either of them may do so again if $a_p + b_p + a_d + b_d \leq 5$. ∎

A constraint forEachMember takes a principal $prin$ (usually a group) and a list of constraints and indicates that these constraints hold for each principal in $prin$, taken individually.

There are two primitive requirements, prePay and attribution. The prePay requirement takes an amount of money; it is met if the user pays the money before trying to perform the action. The attribution requirement takes a subject s; it is met if during the execution of the action, s is properly acknowledged (e.g., as the writer, producer, etc.). (The ODRL specification does not explain how a user can make an acknowledgement.) The set of requirements is closed under the inSequence construct, which says the requirements must be met in a particular order (e.g., acknowledge, then pay), and under the anySequence construct, which says the requirements can be met in any order.

Finally, conditions are constraints and policies that must not hold. For example, a condition could specify that the user is not Alice or that the user is not permitted to play 'Finding Nemo'. Notice that, because of the exclusion tag, the notion of "not permitted" is ambiguous. The statement *Alice is not permitted to play 'Finding Nemo'* could mean that Alice does not have explicit permission or it could mean that Alice is explicitly forbidden. Which interpretation is made can effect which permissions are granted and, thus, the distinction is important. Since the ODRL specification does not address this issue, we define a policy condition to hold if the permission is not granted. In the next section, we show that this choice leads to a more intuitive semantics, and also simplifies our translation. The syntax for prerequisites is given below.

Prerequisites:

prq ::=	prerequisite
$\quad cons$	constraint
$\quad req$	requirement
$\quad cond$	condition
$cons$ ::=	constraint
$\quad prin$	principal
$\quad \mathsf{forEachMember}[prin; cons_1, \ldots, cons_m]$	constraint distribution ($m \geq 1$)
$\quad \mathsf{count}[n]$	number of executions ($n \in \mathbb{N}$)
req ::=	requirement
$\quad \mathsf{prePay}[r]$	prepayment ($r \in \mathbb{R}_+$)
$\quad \mathsf{attribution}[s]$	attribution to subject s
$\quad \mathsf{inSequence}[req_1, \ldots, req_m]$	ordered constraints ($m \geq 1$)

$\quad\mathsf{anySequence}[req_1, \ldots, req_m]$	$\quad$ unordered constraints ($m \geq 1$)
$cond ::=$	condition
$\quad\mathsf{not}[ps]$	$\quad$ suspending policy set
$\quad\mathsf{not}[cons]$	$\quad$ suspending constraint

Example 2.2. Consider the following agreement:

$$\begin{array}{l}\textbf{agreement}\\ \quad\textbf{between } \mathit{Richard} \textbf{ and } \{\mathit{Alice}, \mathit{Bob}\}\\ \quad\textbf{about } \mathit{ebook}\\ \quad\textbf{with } \mathsf{count}[10] \longrightarrow \mathsf{and}[\mathsf{forEachMember}[\{\mathit{Alice}, \mathit{Bob}\}; \mathsf{count}[5]] \Longrightarrow_{id_1} \mathsf{display},\\ \qquad\qquad\qquad\qquad\qquad \mathsf{forEachMember}[\{\mathit{Alice}, \mathit{Bob}\}; \mathsf{count}[1]] \Longrightarrow_{id_2} \mathsf{print}]\end{array}$$

Intuitively, the agreement says that Alice and Bob are given the following rights: they may each display the asset *ebook* up to five times, and they may each print it once. However, the total number of actions, either displays or prints, done by Alice and Bob may be at most ten. ∎

Example 2.3. Consider the following agreement:

$$\begin{array}{l}\textbf{agreement}\\ \quad\textbf{between } \mathit{Richard} \textbf{ and } \{\mathit{Alice}, \mathit{Bob}\}\\ \quad\textbf{about } \mathit{latestJingle}\\ \quad\textbf{with } \mathsf{inSequence}[\mathsf{prePay}[5.00], \mathsf{attribution}[\mathit{Charlie}]] \longmapsto\\ \qquad\qquad (\mathit{Alice}\ \ \mathsf{count}[10] \Longrightarrow_{id} \mathsf{play})\end{array}$$

Intuitively, the agreement says that after paying \$5 and then acknowledging Charlie, Alice is permitted to play the asset *latestJingle* up to ten times. Moreover, any subject that is neither Alice nor Bob is forbidden from playing *latestJingle*. (Bob's right is unregulated.) ∎

As mentioned at the beginning of this section, the syntax presented here is not identical to the one given in the ODRL document. The key differences are discussed below.

Offers. In addition to agreements, ODRL supports offers, which are essentially agreements without users. Intuitively, an offer is a contract (governing the use of an asset) that has yet to be accepted by a user; once accepted, it becomes an agreement. We can interpret offers in a manner that is similar to our interpretation of agreements.

Permissions versus Policies. The ODRL document uses the term *permission* to refer to actions, policies, and policy sets, as defined here. We introduce the distinction to clarify the exposition and to emphasize the two-tier structure of ODRL. Notice that it is the two layers in the framework that allow a prerequisite to apply to multiple policies. These two layers already appear in the XML-based syntax of ODRL, although the term permission is used for both layers.

Contexts. ODRL uses contexts to assign additional information to agreements, prerequisites, and other entities. A context might include a unique identifier, a human-readable name, an expiration date, and so on. We represent the context elements that are included in our fragment directly in the syntax. Adding full contexts to our syntax is straightforward, but it does not add any insight. In fact, it obfuscates the main issues.

Prerequisites. Payments and other requirements in ODRL take a number of arguments. For instance, payments can take an amount and a percentage to be collected for taxes. We restrict every prerequisite to at most one argument for simplicity; it is easy to extend our approach to include multiple arguments. ODRL also supports nested constraints, that is, constraints that apply to other constraints. These can be handled in a manner similar to that used for forEachMember.

Sequences and containers. In ODRL, sequences (inSequence, anySequence) and containers (and, or, xor) apply to a number of entities. For simplicity, we associate containers with policies and policy sets, and associate sequences with requirements. The general case is a straightforward extension.

Right holders. In ODRL, right holders have a royalty annotation, indicating the amount of royalty that they receive. This does not reflect an obligation on the part of the agreement's user, since payment obligations are captured by requirements. Instead, the annotations record how the payments are distributed. Since we are primarily interested in capturing permissions, we do not consider royalty annotations, and as a result, do not distinguish right holders from other principals.

Revocation. Finally, the ODRL document mentions revocation, however it is not clearly defined. A revocation invalidates a previously established agreement. Unfortunately, answers to key questions, such as who can revoke an agreement, under what conditions, and subject to what penalties, are not discussed in the ODRL document. As it stands, a revocation simply indicates that an agreement has been nullified, and thus may be ignored.

3 A Semantics in First-Order Logic

In this section, we formalize the intuitive description of ODRL given in Section 2. Specifically, we present a translation from agreements to formulas in many-sorted first-order logic with equality. For the rest of this discussion, we assume knowledge of many-sorted first-order logic at the level of Enderton [3]. 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 notion of satisfiability and validity of first-order formulas.

We assume sorts $Actions$, $Subjects$, $Reals$, $Assets$, $PolIds$, and $SetPolIds$ (for sets of policy identifiers). Since we need to reason about time, we assume that time is represented by real numbers, and occasionally refer to the sort $Reals$ as the sort $Times$.

The vocabulary includes:

- A predicate **Permitted** on $Subjects \times Actions \times Assets$. The literal **Permitted**(s, act, a) means s is permitted to perform action act on asset a.
- A predicate **Paid** on $Reals \times SetPolIds \times Times$. The literal **Paid**$(r, P, t)$ means an amount r was paid at time t towards the policy set P.
- A predicate **Attributed** on $Subjects \times Times$. The literal **Attributed**(s, t) means s was acknowledged at time t.
- Constants of sort $PolIds$, $SetPolIds$, $Subjects$, and $Assets$; we also assume constants *play*, *display*, and *print* of sort $Actions$.

- A function $count : Subjects \times PolIds \rightarrow Reals$. Intuitively, $count(s, id)$ is the number of times the policy with identifier id is used by subject s to justify an action.
- Standard functions for addition and comparison of real numbers.

We write $t < \infty$ as an abbreviation for **true** (for all t of sort $Reals$).

Before presenting the translation, we define some useful auxiliary functions. The function $principals$ takes a principal and returns the set of principals that are members of a group, or a singleton set if the principal is an individual subject. The function $subjects$ takes a prerequisite; if the prerequisite is a user constraint $prin$, then $subjects$ returns the set of subjects in $prin$; otherwise, it returns the full set $Subjects$. (Intuitively, $subjects(prq)$ describes the constraint on subjects that prq imposes.) Finally, the function ids takes a policy p, and returns the set of policy identifiers that are mentioned in p.

Extracting Principals, Subjects, and Policy Identifiers:

$$principals(s) \triangleq \{s\}$$
$$principals(\{prin_1, \ldots, prin_k\}) \triangleq \{prin_1, \ldots, prin_k\}$$

$$subjects(prq) \triangleq \begin{cases} \{s\} & \text{if } prq = s \\ \bigcup_{i=1}^{k} subjects(prin_i) & \text{if } prq = \{prin_1, \ldots, prin_k\} \\ Subjects & \text{otherwise} \end{cases}$$

$$ids(pr_1 \ \ldots \ pr_m \Longrightarrow_{id} act) \triangleq \{id\}$$
$$ids(\mathsf{and}[p_1, \ldots, p_m]) \triangleq \bigcup_{i=1}^{m} ids(p_i)$$
$$ids(\mathsf{or}[p_1, \ldots, p_m]) \triangleq \bigcup_{i=1}^{m} ids(p_i)$$
$$ids(\mathsf{xor}[p_1, \ldots, p_m]) \triangleq \bigcup_{i=1}^{m} ids(p_i)$$

We present the translation inductively on the structure of the agreement. Intuitively, an agreement is translated into a formula of the form $\forall x(P(x))$, where $P(x)$ is a Boolean combination of formulas of the form prereqs$(x) \Rightarrow$ **Permitted**(x, act, a) and x is a variable of sort $Subjects$ that is free in $P(x)$.

Translation of Agreement agr to Formula $[\![agr]\!]$:

$$[\![\textbf{agreement between } prin_o \textbf{ and } prin_u \textbf{ about } a \textbf{ with } ps]\!] \triangleq \forall x([\![ps]\!]_x^{prin_u, a})$$

Note that $prin_o$ is not involved in the translation. This is in keeping with the ODRL specification, which assumes each agreement was issued legitimately, and thus the particular identity of the agreement's owner is irrelevant.

The translation of a policy set ps is a formula $[\![ps]\!]_x^{prin_u, a}$, where $prin_u$ is the agreement's user, a is the asset, and x is a variable of sort $Subjects$. A (nonexclusive) primitive policy set $prq_1 \ \ldots \ prq_k \longrightarrow p$ translates to an implication: if the prerequisites hold, then the policy holds. A primitive policy set that is exclusive is translated as a primitive policy set in conjunction with a clause that captures the prohibition (i.e., every subject that is not mentioned in the agreement's user is forbidden from performing the actions). Boolean combinations of policy sets translate to Boolean combinations of the corresponding formulas. (In the translation, we follow the convention that $\bigwedge_{i=1}^{m} \varphi_i$ is **true** when $m = 0$.)

Translation of Policy Set ps to Formula $[\![ps]\!]_x^{prin_u,a}$:

$$[\![prq_1 \ \ldots \ prq_m \longrightarrow p]\!]_x^{prin_u,a} \triangleq [\![prin_u]\!]_x \Rightarrow \left(\bigwedge_{i=1}^{m} [\![prq]\!]_x^{ids(p),S}\right) \Rightarrow [\![p]\!]_x^{+,prin_u,a}$$
where $S = subjects(prin_u) \cap (\bigcap_{i=1}^{m} subjects(prq_i))$
$$[\![prq_1 \ \ldots \ prq_m \longmapsto p]\!]_x^{prin_u,a} \triangleq \left([\![prin_u]\!]_x \Rightarrow \left(\bigwedge_{i=1}^{m} [\![prq]\!]_x^{ids(p),S}\right) \Rightarrow [\![p]\!]_x^{+,prin_u,a}\right) \wedge (\neg [\![prin_u]\!]_x \Rightarrow [\![p]\!]_x^{-,a})$$
where $S = subjects(prin_u) \cap (\bigcap_{i=1}^{m} subjects(prq_i))$
$$[\![\mathsf{and}[ps_1, \ldots, ps_m]]\!]_x^{prin_u,a} \triangleq \bigwedge_{i=1}^{m} [\![ps_i]\!]_x^{prin_u,a}$$
$$[\![\mathsf{or}[ps_1, \ldots, ps_m]]\!]_x^{prin_u,a} \triangleq \bigvee_{i=1}^{m} [\![ps_i]\!]_x^{prin_u,a}$$
$$[\![\mathsf{xor}[ps_1, \ldots, ps_m]]\!]_x^{prin_u,a} \triangleq \bigvee_{i=1}^{m} \left([\![ps_i]\!]_x^{prin_u,a} \wedge \left(\bigwedge_{j=1, j\neq i}^{m} \neg [\![ps_j]\!]_x^{prin_u,a}\right)\right)$$

The translation of policy sets refers to the following translation of principals. The formula $[\![prin]\!]_x$ is true if and only if the subject denoted by the variable x is one of the subjects in principal $prin$.

Translation of Principal $prin$ to Formula $[\![prin]\!]_x$:

$$[\![s]\!]_x \triangleq x = s$$
$$[\![\{prin_1, \ldots, prin_k\}]\!] \triangleq ([\![prin_1]\!]_x \vee \ldots \vee [\![prin_k]\!]_x)$$

The translation of policy sets also refers to the translation of policies. In fact, there are two translations for policies: a positive translation, where the permissions described by a policy are granted, and a negative translation, where they are forbidden. The positive translation of a policy p is a formula $[\![p]\!]_x^{+,prin_u,a}$, where $prin_u$ is the user of the agreement, a is the asset, and x is the variable that ranges over the subjects. A policy of the form $prq_1 \ \ldots \ prq_m \Longrightarrow act$ translates to an implication: if the prerequisites hold, then the subject represented by x is permitted to perform action act on asset a. As with policy sets, Boolean combinations of policies translate to Boolean combinations of the corresponding formulas.

Translation of Positive Policy p to Formula $[\![p]\!]_x^{+,prin_u,a}$:

$$[\![prq_1 \ \ldots \ prq_m \Longrightarrow_{id} act]\!]_x^{+,prin_u,a} \triangleq \left(\bigwedge_{i=1}^{m} [\![prq_i]\!]_x^{\{id\},S}\right) \Rightarrow \textbf{Permitted}(x, [\![act]\!], a)$$
where $S = subjects(prin_u) \cap (\bigcap_{i=1}^{m} subjects(prq_i))$
$$[\![\mathsf{and}[p_1, \ldots, p_m]]\!]_x^{+,prin_u,a} \triangleq \bigwedge_{i=1}^{m} [\![p_i]\!]_x^{+,prin_u,a}$$
$$[\![\mathsf{or}[p_1, \ldots, p_m]]\!]_x^{+,prin_u,a} \triangleq \bigvee_{i=1}^{m} [\![p_i]\!]_x^{+,prin_u,a}$$
$$[\![\mathsf{xor}[p_1, \ldots, p_m]]\!]_x^{+,prin_u,a} \triangleq \bigvee_{i=1}^{m} \left([\![p_i]\!]_x^{+,prin_u,a} \wedge \left(\bigwedge_{j=1, j\neq i}^{m} \neg [\![p_j]\!]_x^{+,prin_u,a}\right)\right)$$

The negative translation of a policy p is a formula $[\![p]\!]_x^{-,a}$, where a is the asset, and x is the variable that ranges over the subjects. If p is $prq_1 \ \ldots \ prq_m \Longrightarrow act$, then the translation simply forbids the action act. Roughly speaking, if p is a Boolean combination of policies, then the translation forbids all of the mentioned actions.

Translation of Negative Policy p to Formula $[\![p]\!]_x^{-,a}$:

$[\![prq_1 \ \ldots \ prq_m \Longrightarrow_{id} act]\!]_x^{-,a} \triangleq \neg\textbf{Permitted}(x, [\![act]\!], a)$
$[\![\mathsf{and}[p_1, \ldots, p_m]]\!]_x^{-,a} \triangleq \bigwedge_{i=1}^{m} [\![p_i]\!]_x^{-,a}$
$[\![\mathsf{or}[p_1, \ldots, p_m]]\!]_x^{-,a} \triangleq \bigwedge_{i=1}^{m} [\![p_i]\!]_x^{-,a}$
$[\![\mathsf{xor}[p_1, \ldots, p_m]]\!]_x^{-,a} \triangleq \bigwedge_{i=1}^{m} [\![p_i]\!]_x^{-,a}$

The positive and negative translations of policies use the following translation of actions, which simply returns the constant corresponding to the action.

Translation of Action act to Term $[\![act]\!]$:

$[\![\mathsf{play}]\!] \triangleq$ *play*
$[\![\mathsf{display}]\!] \triangleq$ *display*
$[\![\mathsf{print}]\!] \triangleq$ *print*

The translations of policy sets and policies refer to a translation of prerequisites. The translation of a prerequisite *prq* is a formula $[\![prq]\!]_x^{I,S}$, where I is a set of policy identifiers, S is a set of subjects, and x is a variable of sort $Subjects$. Intuitively, I includes (the identifier of) the policies that are implied by the prerequisites and S includes the subjects to which the prerequisites apply (the agreement's user, unless overridden by a user constraint). A user constraint $prin$ translates to a formula that is true only if the current subject x is a member of $prin$. The translation of the other constraints is more complicated. A forEachMember constraint translates to a formula that is true if, intuitively, each constraint in forEachMember is met by each principal of the group. A count constraint translates to a formula that is true if the sum of the count of each subject in S and each policy identifier in I is no more than the specified integer.

Requirements have a significantly different translation than other prerequisites, because of their dependence on time. (As an example of this dependence, recall that $\mathsf{inSequence}[\mathsf{prePay}[r], \mathsf{attribution}[s]]$ holds if r was paid *before* attribution to s was given.) To handle time correctly, we translate $[\![req]\!]_x^{I,S}$ to $\exists t([\![req]\!]_{t,\infty}^{I})$, where t is a variable of sort $Times$, and $[\![req]\!]_{t,t'}^{I}$ is an auxiliary translation that in some sense restricts the occurrences of events to the interval of time between t and t'. If *req* is a primitive requirement (i.e., a payment or attribution), then we translate $[\![req]\!]_{t,t'}^{I}$ to a formula that is true if the relevant payment or attribution occurred at time t. (For primitive requirements, the value of t' is irrelevant.) An inSequence requirement is satisfied if there exists appropriate successive times between t and t' at which each subrequirement is satisfied. Similarly, an anySequence requirement is satisfied if the subrequirements are satisfied in an arbitrary order between times t and t'.

Conditions are translated by negating the translation of either the policy set or the constraint specified as an argument. A negated policy set can refer to a different subject than the enclosing policy (or policy set). This ensures that we can interpret statements such as *If Alice is not permitted to play 'Finding Nemo', then Bob is permitted to borrow 'Finding Nemo'*. Notice that Bob may borrow the movie if Alice is not explicitly permitted to play it; this is a consequence of our translation. The translation for requirements is given below, where we abbreviate $\varphi(a_1) + \ldots + \varphi(a_k)$ as $\sum_{a \in \{a_1,\ldots,a_k\}} \varphi(a)$.

Translation of Prerequisite prq **to Formula** $[\![prq]\!]_x^{I,S}$**:**

$[\![prin]\!]_x^{I,S} \triangleq [\![prin]\!]_x$

$[\![\mathsf{forEachMember}[prin; cons_1, \ldots, cons_m]]\!]_x^{I,S} \triangleq \bigwedge_{(prin',i) \in P_m} [\![cons_i]\!]_x^{I,subjects(prin')}$

where $P_m = principals(prin) \times \{1, \ldots, m\}$

$[\![\mathsf{count}[n]]\!]_x^{I,S} \triangleq \left(\sum_{(id,s) \in I \times S} count(s, id)\right) \leq n$

$[\![req]\!]_x^{I,S} \triangleq \exists t([\![req]\!]_{t,\infty}^I)$

where $[\![\mathsf{prePay}[r]]\!]_{t,t'}^I \triangleq \mathbf{Paid}(r, I, t)$

$[\![\mathsf{attribution}[s]]\!]_{t,t'}^I \triangleq \mathbf{Attributed}(s, t)$

$[\![\mathsf{inSequence}[req_1, \ldots, req_k]]\!]_{t_1,t_{k+1}}^I \triangleq$

$\exists t_2 \ldots \exists t_k (t_1 < \ldots < t_{k+1} \wedge \bigwedge_{i=1}^k [\![req_i]\!]_{t_i,t_{i+1}}^I)$

$[\![\mathsf{anySequence}[req_1, \ldots, req_k]]\!]_{t,t'}^I \triangleq$

$\exists t_1 \ldots \exists t_k (\bigwedge_{i=1}^k (t_i \geq t \wedge t_i < t' \wedge [\![req_i]\!]_{t_i,t'}^I))$

$[\![\mathsf{not}[ps]]\!]_x^{I,S} \triangleq \neg \forall y([\![ps]\!]_y)$

$[\![\mathsf{not}[cons]]\!]_x^{I,S} \triangleq \neg [\![cons]\!]_x^{I,S}$

Notice that, according to our translation, the condition $\mathsf{not}[ps]$ could be met, even if ps does not appear as such in an agreement. This is because we might be able to infer ps. For example, consider the agreement that includes the the user $\{Alice, Bob\}$, the asset 'Finding Nemo', and the policy set

$$\mathsf{and}[\mathsf{not}[Alice] \longrightarrow \mathsf{play}, \mathsf{not}[Bob] \longrightarrow \mathsf{play}].$$

Roughly speaking, the first policy says that Bob may play the movie and the second says that Alice may play it. From this agreement we can infer that Alice or Bob may play the movie. Therefore, this agreement implies a second one that includes the same user and asset, but its policy set is simply play; it follows that by creating the first agreement, the condition $\mathsf{not}[\mathsf{play}]$ is violated.

Another subtlety arises in the interpretation of sequence requirements, particularly *nested* sequence requirements. To illustrate the issue, consider the nested requirement $\mathsf{anySequence}[\mathsf{inSequence}[req_1, req_2], req_3]$. What are the allowed sequences of requirements req_1, req_2, and req_3? One possibility, the one we adopt, is that $\mathsf{inSequence}[req_1, req_2]$ is met if req_1 happens before req_2. Thus, the following sequences are allowed: $req_1\ req_2\ req_3$, $req_1\ req_3\ req_2$, and $req_3\ req_1\ req_2$. Alternatively, one could say that $\mathsf{inSequence}[req_1, req_2]$ is met if req_1 and req_2 happen consecutively. Under this interpretation, only the following sequences are allowed: $req_1\ req_2\ req_3$ and $req_3\ req_1\ req_2$. We can capture this last interpretation by taking:

$$[\![\mathsf{anySequence}[req_1, \ldots, req_k]]\!]^I_{t_1,t_{k+1}} \triangleq$$
$$\exists t_2 \ldots \exists t_k (t_1 < \ldots < t_{k+1} \wedge \bigvee_{\pi \in S_k} (\bigwedge_{i=1}^{k} [\![req_{\pi(i)}]\!]^I_{t_i,t_{i+1}})),$$

where S_k is the set of all permutations of sets of k elements.

The translations presented above are admittedly complex. However, the translated agreements correspond rather closely to the original syntax. To illustrate this, we translate Examples 2.2 and 2.3 from Section 2.

Example 3.1. The agreement in Example 2.2 translates to the formula:

$$\begin{aligned}&\forall x((x = Alice \vee x = Bob) \Rightarrow\\ &\quad count(Alice, id_1) + count(Alice, id_2) + count(Bob, id_1) + count(Bob, id_2) \leq 10 \Rightarrow\\ &\qquad (count(Alice, id_1) \leq 5 \wedge count(Bob, id_1) \leq 5 \Rightarrow \textbf{Permitted}(x, display, ebook)) \wedge\\ &\qquad (count(Alice, id_2) \leq 1 \wedge count(Bob, id_2) \leq 1 \Rightarrow \textbf{Permitted}(x, print, ebook))). \blacksquare\end{aligned}$$

Example 3.2. The agreement in Example 2.3 translates to the formula:

$$\begin{aligned}&\forall x((x = Alice \vee x = Bob) \Rightarrow\\ &\quad \exists t_1 \exists t_2 (t_1 < t_2 \wedge \textbf{Paid}(5.00, t_1) \wedge \textbf{Attributed}(Charlie, t_2)) \Rightarrow\\ &\qquad (x = Alice \wedge count(Alice, id) \leq 10 \Rightarrow \textbf{Permitted}(x, play, latestJingle)) \wedge\\ &\qquad (\neg(x = Alice \vee x = Bob) \Rightarrow \neg\textbf{Permitted}(x, play, latestJingle))). \blacksquare\end{aligned}$$

We are now in a position to formally define when an agreement implies a permission. In order to do this, we assume that there is an environment that includes all of the relevant facts about the world, such as when payments are made and the number of times a particular policy was used to justify a subject's action. More precisely, we assume a formula E that is conjunction of ground literals. Determining whether a subject s is permitted to perform action act to asset a under agreement agr in environment E amounts to asking if the formula $E \wedge [\![agr]\!] \Rightarrow$ **Permitted**(s, act, a) is valid. Similarly, a subject s is forbidden to perform action act to asset a under agreement agr in environment E if the formula $E \wedge [\![agr]\!] \Rightarrow \neg$**Permitted**$(s, act, a)$ is valid. If neither formula is valid, then subject s is neither permitted nor forbidden to perform action act to asset a under agreement agr in environment E. We can easily extend the definition to take several agreements into account. A subject s is permitted to perform action act to asset a under the agreements $agr_1, \ldots, agr_n$ in environment E if the formula $E \wedge [\![agr_1]\!] \wedge \ldots \wedge [\![agr_n]\!] \Rightarrow$ **Permitted**(s, act, a) is valid. Prohibitions are defined similarly.

Thus, the problem of deciding whether or not a particular subject is permitted or not to perform a certain action to a given asset under a set of agreements in an environment is reduced to the problem of deciding whether or not a formula of first-order logic is valid. In fact, we can be a little more precise, by noting that for all agreements agr, the formula $E \wedge [\![agr]\!] \Rightarrow$ **Permitted**(s, act, a) is an existential first-order formula, where an *existential* first-order formula is a closed first-order formula that can be written in the form $\exists x_1 \ldots \exists x_k(\varphi)$ and φ is quantifier-free. Determining the validity of an existential formula is, in general, an undecidable problem. It may be possible to exploit the particular structure of agreements to prove that our queries can be answered.

4 Conclusion

Languages for writing agreements typically fall into one of three categories: native languages, such as English, that cannot be interpreted by machines, XML-based languages [1, 6] that enjoy popular support by application writers, and formal logics [12, 9, 5] that are endorsed by computer scientists, because they have formal semantics (no ambiguity) and are tractable (queries can be answered typically in a low-order polynomial time). ODRL belongs to the second category. By providing formal semantics to a fragment of ODRL, we get the benefit of using a formal approach, namely no ambiguity, and we can begin to search for a tractable fragment. In this way, we get the best of both worlds.

The translation also allows us to compare ODRL with the formal approaches. For example, many of the languages in the Computer Science literature are based on a fragment of first-order logic called Datalog with Negation [10, 8, 2, 7]; others are based on Datalog with Constraints [11]. It is easy to show that ODRL's use of negation and functions cannot be duplicated in either of these fragments. Of course, the cost of the additional expressiveness on the language's tractability has yet to be determined; we hope to explore this issue in future work.

Acknowledgments

Thanks to Renato Iannella (Chief Scientist at IPR Systems when ODRL version 1.0 was released), who took the time to answer our questions about the interpretation of various ODRL constructions.

A XML-Based ODRL Syntax

We stated in Section 2 that it is straightforward to translate from the XML-based syntax of ODRL to our syntax, at least for the fragment of ODRL that we support. To illustrate this translation, and to give an example of an agreement in XML, consider the following agreement, taken from [4]. (For readability, we have omitted namespace information.)

```
<agreement> <context> <uid> license-12345 </uid>
                      <pLocation> Sydney, Australia </pLocation>
                      <remark> Transacted by Example.Com </remark> </context>
  <asset> <context> <uid> rossi-12345 </uid> </context> </asset>
  <permission>
    <display>
      <constraint>
        <cpu> <context> <uid> Intel-12345 </uid> </context> </cpu>
      </constraint>
    </display>
    <print>
      <constraint> <count> 2 </count> </constraint>
    </print>
    <requirement>
      <prepay>
        <payment> <amount currency="AUD"> 20.00</amount>
                  <taxpercent code="GST"> 10.00</taxpercent> </payment>
      </prepay>
     </requirement>
  </permission>
  <party> <context> <uid> msmith </uid>
                    <name> Mary Smith </name> </context>
  </party>
</agreement>
```

We see the use of contexts to hold information, as we noted in Section 2. We also see that the `prepay` requirement applies to both the `display` and the `print` permissions, since it occurs on the same level as the permissions. The agreement also does not specify a right holder. This information would presumably be specified by some external means. For the purposes of translation, we assume a right holder r. The above agreement would be expressed as follows in our syntax, where $msmith$ is the subject name of Mary Smith.

$$\begin{array}{l} \textbf{agreement} \\ \quad \textbf{between } r \textbf{ and } msmith \\ \quad \textbf{about } rossi12345 \\ \quad \textbf{with } \mathsf{prePay}[20.00] \longrightarrow \mathsf{and}[\mathsf{cpu}[intel12345] \Longrightarrow \mathsf{display}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{count}[2] \Longrightarrow \mathsf{print}] \end{array}$$

References

1. ContentGuard. XrML: Extensible rights Markup Language. Available from `http://www.xrml.org`, 2001.
2. J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Research in Security and Privacy*, pages 95–103. IEEE Computer Society Press, 2002.
3. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.
4. S. Guth, G. Neumann, and M. Strembeck. Experiences with the enforcement of access rights extracted from ODRL-based digital contracts. In *Proceedings of the Workshop on Digital Rights Management (DRM'03)*, pages 90–101. ACM Press, 2003.
5. J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 187–201. IEEE Computer Society Press, 2003.
6. R. Iannella. Open Digital Rights Language (ODRL) version 1.1. Available from `http://www.w3.org/TR/odrl`, 2002.
7. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.
8. T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Research in Security and Privacy*, pages 106–115. IEEE Computer Society Press, 2001.
9. R. M. Lee. International contracting—a formal language approach. In *Hawaii International Conference on Systems Sciences*, pages 69–78, 1988.
10. 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)*, 6(1):128–171, 2003.
11. 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*, volume 2562 of *Lecture Notes in Computer Science*, pages 58–73, 2003.
12. R. van der Meyden. The dynamic logic of permission. *Journal of Logic and Computation*, 6(3):465–479, 1996.