# A Formal Foundation for XrML

Joseph Y. Halpern and Vicky Weissman
Cornell University

XrML is becoming a popular language in industry for writing software licenses. The semantics for XrML is implicitly given by an algorithm that determines if a permission follows from a set of licenses. We focus on a fragment of the language and use it to highlight some problematic aspects of the algorithm. We then correct the problems, introduce formal semantics, and show that our semantics captures the (corrected) algorithm. Next, we consider the complexity of determining if a permission is implied by a set of XrML licenses. We prove that the general problem is undecidable, but it is polynomial-time computable for an expressive fragment of the language. We extend XrML to capture a wider range of licenses by adding negation to the language. Finally, we discuss the key differences between XrML and MPEG-21, an international standard based on XrML.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security*; *integrity*; *protection*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*

General Terms: Security, Languages

Additional Key Words and Phrases: Digital Rights Management

## 1. INTRODUCTION

The eXtensible rights Markup Language (XrML) is becoming an increasingly popular language in which to write software licenses. When first released in 2000, XrML received the support of many technology providers, content owners, distributors, and retailers, including Adobe Systems, Hewlett-Packard Laboratories, Microsoft, Xerox Corp., Barnesandnoble.com, and Time Warner Trade Publishing. In fact, Microsoft, OverDrive, and DMDsecure have publicly announced their agreement to build products and/or services that are XrML compliant. Currently, XrML is being used by international standard committees as the basis for application-specific languages that are designed for use across entire industries. For example, the Moving Picture Experts Group (MPEG) has selected XrML as the foundation for their MPEG-21 Rights Expression Language, henceforth referred to as MPEG-21 (see http://www.xrml.org). It is clear that a number of industries are moving towards a

Authors' address: J. Halpern and V. Weissman, Cornell University, Ithaca, NY 14853.
Authors supported in part by NSF under grants CTC-0208535, ITR-0325453, and IIS-0534064, 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 grants F49620-02-1-0101 and FA9550-05-1-0055. A preliminary version of this paper appeared at the 17th IEEE Computer Security Foundations Workshop in Pacific Grove, California, 2004.

standard language for writing licenses and that many of these standard languages are likely to be based on XrML. To understand the new standards, we need to understand XrML.

XrML does not have formal semantics. Instead, the XrML specification [ContentGuard 2001] presents the semantics in two ways. First is an English description of the language. Second is an English description of an algorithm that determines if a permission follows from a set of licenses. Unfortunately, the two versions of the semantics do not agree. To make matters worse, the algorithm has unintuitive consequences that do not seem to reflect the language developers' intent.

To address these issues, we provide formal semantics for a fragment of XrML. We focus on a fragment because the entire language is somewhat unwieldy. An XrML license says that an agent grants a permission if certain conditions hold. Our fragment includes only two types of permissions and only two types of conditions. We give our fragment formal semantics by defining a translation from licenses in the fragment to formulas in first-order logic extended with a validity operator. We argue that the translation preserves the meaning of the XrML statements by proving that the algorithm included in the XrML document, slightly modified to correct the unintuitive behavior, matches our semantics. More precisely, the algorithm says that a permission follows from a set of licenses if and only if the translated permission is a logical consequence of the translated licenses. We then consider the complexity of determining if a permission is implied by a set of licenses. We show that the general problem is undecidable, even for our fragment. The problem is decidable in polynomial time if we restrict the fragment slightly.

A shortcoming of XrML is that it does not support negation. For example, in XrML, we cannot write "customers may *not* edit the software". The XrML developers deal with this limitation, to some extent, by assuming that an action is forbidden unless it is explicitly permitted. As a result, a license writer does not need to say that an action is forbidden, because the prohibition is already implied. This approach might be acceptable in various instances, but it is difficult to believe that most license writers really want to forbid *every* action that they do not explicitly permit. So, the approach does not capture the license writer's actual intent. Moreover, it limits the class of licenses that can be expressed, because it removes the distinction between forbidden and unregulated actions. For example, in XrML, we cannot say "a hospital may petition for an exemption if it permits an action that the government forbids". Similarly, a course instructor cannot say "if the university does not object, then Alice is permitted to audit the class". In this paper, we extend XrML to include such statements and consider the effect of the addition on the language's tractability.

MPEG-21 is an international standard based on XrML. When we first decided to give XrML formal semantics, the MPEG committee had released a beta version of its language, which was XrML with minor revisions, and was preparing the final release. We chose to give semantics to the beta language first (before analyzing the official XrML specification, as is done here), because we hoped that any problems we found would be corrected in the final version of MPEG-21. This is, in fact, what occured. After discussing our results with Thomas DeMartini and Xin Wang of the MPEG Standards Committee, the committee released their ISO standard

[MPEG 2004]; the shortcomings that we identified are addressed in the standard. We conjecture that all of our results for XrML hold with minor changes for MPEG-21, although we have not verified the details.

The rest of the paper is organized as follows. In the next section we present our fragment of XrML. In Section 3 we review XrML's algorithm for answering queries. After considering some examples in which the algorithm's behavior is unintuitive and almost certainly unintended, we propose corrections that we believe captures the designers' intent. Formal semantics for our fragment are given in Section 4, and the revised algorithm is shown to be sound and complete with respect to the semantics. In Section 5 we show that the problem of determining if a permission follows from a set of licenses is undecidable. We also discuss a fragment of XrML that is both tractable and relatively expressive. In Section 6 we outline how our results can be extended to a substantial fragment of XrML. Negation is added to XrML in Section 7. The analysis of this paper had an impact on practice. MPEG-21 REL, an international standard based on XrML, incorporates the developers' response to our concerns about XrML. We describe MPEG-21 REL, and how it deals with our concerns, in Section 8. We conclude in Section 9. All of the proofs are in the appendix.

## 2. SYNTAX

XrML is an XML-based language; it follows XML-conventions. Rather than present that syntax, we use an alternative syntax that is more concise and, we believe, more intuitive. In this section, we introduce our syntax for a fragment of XrML (the rest of the language is discussed in Section 6) and describe the key differences between the syntax used in the XrML specification and that used here.

At the heart of XrML is the notion of a *license*. A license is a (principal, grant) pair, where the license $(p, g)$ means $p$ issues (i.e., says) $g$. For example, the license (Alice, Bob is smart) means "Alice says 'Bob is smart' ".

A grant has the form $\forall x_1 \ldots \forall x_n(\text{condition} \rightarrow \text{conclusion})$, which intuitively means that the condition implies the conclusion under all appropriate substitutions. Conditions and conclusions are defined as follows.

—A condition has the form $d_1 \wedge \ldots \wedge d_n$, where each $d_i$ is either **true** or $\mathbf{Said}(p, e)$ for some principal $p$ and conclusion $e$. Roughly speaking, the condition **true** always holds and the condition $\mathbf{Said}(p, e)$ holds if $p$ issues a grant that says $e$ holds if a condition $d$ holds, and $d$ does, in fact, hold.

—A conclusion has either the form $\mathbf{Perm}(p, r, s)$ or the form $\mathbf{Pr}(p)$, where $\mathbf{Pr}$ is a property, $p$ is a principal, $r$ is a right (i.e., an action), and $s$ is a resource. The conclusion $\mathbf{Perm}(p, r, s)$ means $p$ may exercise $r$ over $s$. For example, $\mathbf{Perm}(Bob, edit, budget\ report)$ means Bob may edit the budget report. The conclusion $\mathbf{Pr}(p)$ means $p$ has the property $\mathbf{Pr}$. For example, the conclusion $\mathbf{Attractive}(Bob)$ means Bob is attractive.

We abbreviate the grant $\forall x_1 \ldots \forall x_n(\mathbf{true} \rightarrow e)$ as $\forall x_1 \ldots \forall x_n e$. Also, we try to consistently use $d$, possibly subscripted, to denote a generic condition and $e$, possibly subscripted, to denote a generic conclusion.

Consider the following example. Suppose that Alice issues the grant "Bob is smart" and Amy issues the grant "if Alice says that Bob is smart, then he is attractive". We can write the first license in our syntax as $(\mathit{Alice}, g_1)$, where $g_1 = \mathbf{Smart}(\mathit{Bob})$ (recall that this is an abbreviation for $\mathbf{true} \rightarrow \mathbf{Smart}(\mathit{Bob})$), and we can write the second as $(\mathit{Amy}, g_2)$, where $g_2 = \mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})) \rightarrow \mathbf{Attractive}(\mathit{Bob})$. Because $(\mathit{Alice}, g_1)$ is in the set of issued licenses, $\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob}))$ holds. It follows from this fact and the license $(\mathit{Amy}, g_2)$ that $\mathbf{Said}(\mathit{Amy}, \mathbf{Attractive}(\mathit{Bob}))$ holds as well.

The sets of principals, properties, rights, and resources depend on the particular application. For example, a multimedia application might have a principal for each employee and each customer; properties such as "hearing impaired" and "manager"; rights such as "edit" and "download"; and a resource for each object such as a movie. We assume the application gives us a finite set *primitivePrin* of principals and a finite set *primitiveProp* of properties. We then define the components in our language as follows.

—The set $P$ of principals is the result of closing *primitivePrin* under union. (Here and elsewhere we identify a principal $p \in \mathit{primitivePrin}$ with the singleton $\{p\}$ and write $\{p_1, \ldots, p_n\}$ rather than $\{p_1\} \cup \ldots \cup \{p_n\}$.) The interpretation of a principal $\{p_1, \ldots, p_n\}$ depends on context; that is, the interpretation depends on whether the principal appears as the first argument in a **Said** condition, in a conclusion, or in a license. We discuss this later in the paper (primarily in Section 5).

—The set of properties is *primitiveProp*. We assume that every property in *primitiveProp* takes a single argument and that argument is of sort *Principal*. For example, *primitiveProp* can include the property **Employee**, where $\mathbf{Employee}(x)$ means principal $x$ is an employee, but it cannot include the property **MotherOf**, where $\mathbf{MotherOf}(x, y)$ means principal $x$ is the mother of principal $y$, nor can it include the property **Vehicle**, where $\mathbf{Vehicle}(x)$ means resource $x$ is a vehicle (e.g., a motorcycle, car, or truck). The results in this paper continue to hold if we extend the language to include properties that take multiple arguments of various sorts (i.e., principals, rights, and resources). It is also easy to show that closing *primitiveProp* under conjunction adds no expressive power to the language. Closing under negation does add expressive power; we return to this issue in Section 7.

—The only right in our language is `issue` and the only resources are grants. Intuitively, if a principal $p$ has the right to issue a grant $g$, and $p$ does issue $g$, then $g$ is a true statement. Including additional rights and resources in our language does not significantly affect the discussion.

We formally define the syntax according to the following grammar.

$$\begin{array}{lll}
\mathit{license} & ::= & (\mathit{prin}, \mathit{grant}) \\
\mathit{grant} & ::= & \forall \mathit{var} \ldots \forall \mathit{var} (\mathit{cond} \rightarrow \mathit{conc}) \\
\mathit{var} & ::= & x_p \mid x_r \\
\mathit{cond} & ::= & \mathbf{true} \mid \mathbf{Said}(\mathit{prin}, \mathit{conc}) \mid \mathit{cond} \wedge \mathit{cond} \\
\mathit{conc} & ::= & \mathbf{Pr}(\mathit{prin}) \mid \mathbf{Perm}(\mathit{prin}, \mathit{right}, \mathit{rsrc}) \\
\mathit{prin} & ::= & \{p\} \mid \{x_p\} \mid \mathit{prin} \cup \mathit{prin} \\
\mathit{right} & ::= & \texttt{issue} \\
\mathit{rsrc} & ::= & \mathit{grant} \mid x_r,
\end{array}$$

where **Pr** is an element of *primitiveProp*, $p$ is an element of *primitivePrin*, $x_p$ is an element of *prinVar*, which is the set of variables ranging over primitive principles, and $x_r$ is an element of *rsrcVar*, which is the set of variables ranging over resources. For the remainder of this paper we assume that the first argument in a license is a singleton. Because the XrML document treats the license $(\{p_1, \ldots, p_n\}, g)$ as an abbreviation for the set of licenses $\{(p, g) \mid p \in \{p_1, \ldots, p_n\}\}$, it is easy to modify our discussion to support all of the licenses included in the grammar.

As mentioned at the beginning of this section, the grammar presented here is not identical to that described in the XrML document. Certain components of XrML are omitted from our language. These are discussed in Section 6. The XrML components that are included are represented using a syntax that we believe is more intuitive. The main differences between the syntax of our language and the syntax of XrML are described below.

—Instead of assuming that the application provides a set of primitive principals, XrML assumes that the application provides a set $K$ of cryptographic keys; the set of primitive principals is $\{\texttt{KeyHolder}(k) \mid k \in K\}$. We could take *primitivePrin* to be this set; however, our more general approach leads to a simpler discussion. Moreover, our results do not change if we restrict primitive principals to those of the form $\texttt{KeyHolder}(k)$.

—XrML does not have conclusions of the form $\mathbf{Pr}(p)$. To capture properties, XrML uses a right called `PossessProperty` and considers the properties given by the application to be resources. The conclusion $\mathbf{Pr}(p)$ in our grammar corresponds to the conclusion $\mathbf{Perm}(p, \texttt{PossessProperty}, \mathbf{Pr})$ in XrML. We have two types of conclusions because we believe the grammar should help distinguish the conceptually different notions of permissions and properties, rather than confounding them.

—Rather than writing $\texttt{AllPrincipals}(p_1, \ldots, p_n)$, $\texttt{AllConditions}(c_1, \ldots, c_n)$, and $\texttt{AllConditions}()$, we use the more standard notations $\{p_1, \ldots, p_n\}$, $c_1 \wedge \ldots \wedge c_n$, and **true**, respectively. Rather than writing $\mathbf{PrerequisiteRight}(p, e)$, we use the shorter and, we believe, more appropriate notation $\mathbf{Said}(p, e)$.

—As discussed previously, XrML abbreviates a set of licenses $\{(p_i, g_j) \mid i \leq n, j \leq m\}$ as the single license $(\{p_1, \ldots, p_n\}, \{g_1, \ldots, g_m\})$. For ease of exposition, we do not do this.

## 3. XRML'S AUTHORIZATION ALGORITHM

The XrML document includes a procedure that we call **Query** to determine if a conclusion follows from a set of licenses (and some additional input that is discussed below). In this section we present and analyze the parts of the algorithm that pertain to our fragment.

Before describing the algorithm, we note that some aspects of **Query** are inefficient. This is acknowledged in the XrML document, which explains that **Query** was designed with clarity as the primary goal; it is the responsibility of the language implementors to create efficient algorithms with the same input/output behavior as **Query**. (In Section 5, we show that it is highly unlikely that such an efficient algorithm exists.)

### 3.1 A Description of **Query**

The input to **Query** is a closed conclusion $e$ (i.e., a conclusion with no free variables), a set $L$ of licenses $(p, g)$ such that $p$ is variable-free, and a set $R$ of grants; **Query** returns **true** if $e$ is implied by $L$ and $R$, and returns **false** otherwise. To explain the intuition behind $L$ and $R$, we first note that the procedure treats a predefined set of principals as trusted. If a trusted principal issues the grant $g$, then $g$ is in $R$ and it is assumed to be true. If the license $(p, g)$ is in $L$, then $p$ issued $g$ (i.e., $p$ says $g$) and $p$ is not an implicitly trusted principal. To clarify the inferences that are drawn from $R$ and $L$, suppose that the grant $g$ is $\mathbf{QueenOfSiam}(\mathit{Alice})$, which means Alice is Queen of Siam, and the grant $g'$ is $\mathbf{Perm}(\mathit{Alice}, \mathtt{issue}, g)$, which means Alice may issue $g$. If $g \in R$, then we assume that Alice really is queen. If $(\mathit{Alice}, g)$ is in $L$, then Alice says that she is the queen, but we cannot conclude that she is royalty from this statement alone. If $(\mathit{Alice}, g)$ is in $L$ and $g'$ is in $R$, then we assume that Alice has the authority to declare herself queen, because $g' \in R$; we assume that she exercises that authority, because $(\mathit{Alice}, g) \in L$; and we conclude that Alice is queen, because this follows from the two assumptions.

**Query** begins by calling the **Auth** algorithm. **Auth** takes $e$, $L$, and $R$ as input; it returns a set $D$ of closed conditions (i.e., conditions with no free variables). Roughly speaking, a closed condition $d$ is in $D$ if $d$, $L$, and $R$ together imply $e$. To determine if a condition in $D$ holds, **Query** relies on the **Holds** algorithm. The input to **Holds** is a closed condition $d$ and a set $L$ of licenses; $\mathbf{Holds}(d, L)$ returns true if the licenses in $L$ imply $d$, and returns **false** otherwise. If $\mathbf{Holds}(d, L)$ returns **true** for some $d$ in $D$, then **Query** returns **true**, indicating that $L$ implies $e$. **Query** is summarized in Figure 1.

<u>$\mathbf{Query}(e, L, R)$:</u>

$D := \mathbf{Auth}(e, L, R)$
**if** $\mathbf{Holds}(d, L) = \mathbf{true}$ for a condition $d \in D$
**then** return **true**
**else** return **false**

Fig. 1. The **Query** Algorithm

We now discuss **Auth** and **Holds** in some detail. To define **Auth**, we first consider the case where $L = \emptyset$. Define a *closed substitution* to be a mapping from variables to closed expressions of the appropriate sort. Given a closed substitution $\sigma$ and an expression $t$, let $t\sigma$ be the expression that arises after all free variables $x$ in $t$ are replaced by $\sigma(x)$. Roughly speaking, $\mathbf{Auth}(e, \emptyset, R)$ returns the set $D$ of closed conditions such that each condition in $D$, in conjunction with the grants in $R$, implies $e$. That is, $d \in D$ iff there is a grant $g = \forall x_1 \ldots \forall x_n (d_g \rightarrow e_g)$ in $R$ and a closed substitution $\sigma$ such that $d = d_g\sigma$ and $e_g$ implies $e$. **Auth** determines whether $e_g$ implies $e$ in a somewhat nonstandard way. In particular, it makes the *subset assumption*, which says that any property or permission attributed to a principal $p$ is attributed to every principal that includes $p$. In other words, if $p \subseteq p'$, then

$\mathbf{Pr}(p)$ implies $\mathbf{Pr}(p')$ and $\mathbf{Perm}(p, r, s)$ implies $\mathbf{Perm}(p', r, s)$. Thus,

$$\begin{aligned}\mathbf{Auth}(\mathbf{Pr}(p), \emptyset, R) \;=\; & \{d \mid \text{for some grant } g = \forall x_1 \ldots \forall x_n(d_g \rightarrow \mathbf{Pr}(p_g)) \in R \text{ and closed} \\ & \text{substitution } \sigma, d_g\sigma = d \text{ and } p_g\sigma \subseteq p\} \text{ and}\end{aligned}$$

$$\begin{aligned}\mathbf{Auth}(\mathbf{Perm}(p, r, s), \emptyset, R) \;=\; & \{d \mid \text{for some grant } \forall x_1 \ldots \forall x_n(d_g \rightarrow \mathbf{Perm}(p_g, r_g, s_g)) \in R \\ & \text{and closed substitution } \sigma, d_g\sigma = d, p_g\sigma \subseteq p, r_g\sigma = r, \\ & \text{and } s_g\sigma = s\}.\end{aligned}$$

Suppose that $L \neq \emptyset$. Then we reduce to the previous case by taking $\mathbf{Auth}(e, L, R) = \mathbf{Auth}(e, \emptyset, R')$, where, intuitively, $R'$ is the set of legitimate grants; that is, $R'$ consists of the grants in $R$ and the grants issued by someone who has the authority to do so. It seems reasonable to call $\mathbf{Query}(\mathbf{Perm}(p, \texttt{issue}, g), L, R)$ to determine if a principal $p$ has the authority to issue a grant $g$. However, if **Auth** calls $\mathbf{Query}(\mathbf{Perm}(p, \texttt{issue}, g), L, R)$ to construct $R'$, then the algorithm will not terminate, because **Query** calls **Auth**, leading to an infinite call tree. So, instead of calling $\mathbf{Query}(\mathbf{Perm}(p, \texttt{issue}, g), L, R)$, the XrML algorithm determines if $p$ is permitted to issue $g$ by checking if $\mathbf{Holds}(d, L) = \mathbf{true}$ for some $d$ in the set $\mathbf{Auth}(\mathbf{Perm}(p, \texttt{issue}, g), L - \{(p, g)\}, R)$. We discuss the consequences of this solution in Section 3.2. In summary,

$$\begin{aligned}R' &= R \cup R'', \text{ where} \\ R'' &= \{g \mid \text{ for some licence}(p, g) \in L \text{ and condition } d, \\ &\qquad d \in \mathbf{Auth}(\mathbf{Perm}(p, \texttt{issue}, g), L - \{(p, g)\}, R) \text{ and} \mathbf{Holds}(d, L) = \mathbf{true}\}\end{aligned}$$

Pseudocode for **Auth** is given in Figure 2.

We define $\mathbf{Holds}(d, L)$ by induction on the structure of $d$. If $d$ is **true**, then $\mathbf{Holds}(d, L) = \mathbf{true}$. If $d = \mathbf{Said}(p, e)$, then $\mathbf{Holds}(d, L) = \mathbf{true}$ iff $p$ issues a grant $\forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$ such that, for some substitution $\sigma$, $e_g\sigma = e$ and $\mathbf{Holds}(d_g\sigma, L) = \mathbf{true}$. In this context, a principal $\{p_1, \ldots, p_n\}$ issues a grant $g$ if $p_i$ issues $g$ for some $i = 1, \ldots, n$. If $d = d_1 \wedge \ldots \wedge d_n$, where each $d_i$ is **true** or a **Said** condition, then $\mathbf{Holds}(d, L) = \bigwedge_{i=1,\ldots,n} \mathbf{Holds}(d_i, L)$. Pseudocode for **Holds** is given in Figure 3.

Example 3.1. In Section 2, we argued informally that Amy says Bob is attractive if the set of licenses is $L = \{(\mathit{Alice}, g_1), (\mathit{Amy}, g_2)\}$, where $g_1 = \mathbf{Smart}(\mathit{Bob})$ and $g_2 = \mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})) \rightarrow \mathbf{Attractive}(\mathit{Bob})$. The formal algorithm gives the same conclusion. Specifically, $\mathbf{Holds}(\mathbf{Said}(\mathit{Amy}, \mathbf{Attractive}(\mathit{Bob})), L)$ sets $R_{\mathit{Amy}} = \{g_2\}$ and calls $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})), L)$. During this call $R_{\mathit{Alice}}$ is set to $\{g_1\}$ and $\mathbf{Holds}(\mathbf{true}, L)$ is called. Because $\mathbf{Holds}(\mathbf{true}, L) = \mathbf{true}$, $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})), L) = \mathbf{true}$ and, thus, $\mathbf{Holds}(\mathbf{Said}(\mathit{Amy}, \mathbf{Attractive}(\mathit{Bob})), L) = \mathbf{true}$.

Suppose that a trusted principal says that Amy has the authority to issue $g_2$ (i.e., if Amy says $g_2$, then $g_2$ holds). Then we can conclude that Bob really is attractive, because $\mathbf{Query}(\mathbf{Attractive}(\mathit{Bob}), L, R) = \mathbf{true}$, where $R = \{\mathbf{Perm}(\mathit{Amy}, \texttt{issue}, g_2)\}$. Specifically, **Query** begins by calling $\mathbf{Auth}(\mathbf{Attractive}(\mathit{Bob}), L, R)$. $\mathbf{Auth}(\mathbf{Attractive}(\mathit{Bob}), L, R)$, in turn, calls $\mathbf{Auth}(\mathbf{Attractive}(\mathit{Bob}), \emptyset, R')$, where $R' = \{g_2, \mathbf{Perm}(\mathit{Amy}, \texttt{issue}, g_2)\}$. $\mathbf{Auth}(\mathbf{Attractive}(\mathit{Bob}), \emptyset, R') = \{\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob}))\}$. So, Bob is attractive if the condition $\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob}))$ holds. To determine if the condition

**Auth**$(e, L, R)$:

$D := \emptyset$
**if** $L = \emptyset$
**then**
  % Find $D$, the conditions under which $R$ implies $e$
  **if** $e = \mathbf{Pr}(p)$
    **for** each grant $\forall x_1 \ldots \forall x_n(d_g \rightarrow \mathbf{Pr}(p_g)) \in R$
      $D := D \cup \{d \mid d_g\sigma = d \text{ and } p_g\sigma \subseteq p, \text{ for some closed substitution } \sigma\}$
  **if** $e = \mathbf{Perm}(p, r, s)$
    **for** each grant $\forall x_1 \ldots \forall x_n(d_g \rightarrow \mathbf{Perm}(p_g, r_g, s_g)) \in R$
      $D := D \cup \{d \mid d_g\sigma = d, p_g\sigma \subseteq p, r_g\sigma = r, \text{ and } s_g\sigma = s, \text{ for some closed substitution } \sigma\}$
**else**
  % Find $R'$
  $R' := R$
  **for** each license $(p, g) \in L$
    $L' := L - \{(p, g)\}$
    $D' := \mathbf{Auth}(\mathbf{Perm}(p, \mathtt{issue}, g), L', R)$
    **if** $\mathbf{Holds}(d, L) = \mathbf{true}$ for a condition $d \in D'$
    **then** $R' := R' \cup \{g\}$
  % Find $D$, the conditions under which $R'$ implies $e$
  $D := \mathbf{Auth}(e, \emptyset, R')$
return $D$

Fig. 2. The **Auth** Algorithm

**Holds**$(d, L)$:

**if** $d = \mathbf{true}$
**then** return **true**

**if** $d = \mathbf{Said}(p, e)$
**then**
  $R_p = \{g \mid \text{for some principal } p', (p', g) \in L \text{ and } p' \in p\}$
  $D := \{d' \mid$ for some grant $\forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in R_p$ and
    closed substitution $\sigma$, $d_g\sigma = d'$ and $e_g\sigma = e\}$
  **if** $\mathbf{Holds}(d', L) = \mathbf{true}$ for a condition $d' \in D$
  **then** return **true**
  **else** return **false**

**if** $d = d_1 \wedge \ldots \wedge d_n$, where each $d_i$ is **true** or a **Said** condition
**then** return $\bigwedge_{i=1,\ldots,n} \mathbf{Holds}(d_i, L)$

Fig. 3. The **Holds** Algorithm

holds, **Query** calls $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})), L)$. We have already shown that $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Bob})), L) = \mathbf{true}$; we evaluated this call during our analysis of $\mathbf{Holds}(\mathbf{Said}(\mathit{Amy}, \mathbf{Attractive}(\mathit{Bob})), L)$. So Bob is indeed attractive. □

**Query** as described here and in the XrML specification is somewhat ambiguous. For example, the specification does not say in which order the conditions in $D$ should be tested to see if at least one condition in $D$ holds. As a result, there are a number of possible executions of a call $\mathbf{Query}(e, L, R)$, depending on the implementation of **Query**. It is easy to see that, for a particular input, every execution that terminates returns the same output. However, as we show in Example 3.4, whether **Query** terminates can depend on how it is implemented. A similar issue arises with **Auth** and **Holds**. We talk about an execution of **Query**, **Auth**, or **Holds** only if the choice of execution affects whether the algorithm terminates. For example, we write $\mathbf{Query}(e, L, R) = \mathbf{true}$ if every execution of $\mathbf{Query}(e, L, R)$ returns **true**.

### 3.2 An Analysis of **Query**

In this section we present five examples in which **Query** gives unexpected results. Example 3.2 reveals a mismatch between **Query** and the informal language description; the discrepancy exists because **Auth** makes the subset assumption and the informal language description does not. Example 3.3 demonstrates that a license $(p, g)$ should not be removed from the set of licenses when determining if $p$ is permitted to issue $g$. Examples 3.4, 3.5, and 3.6, show that a reasonable implementation of **Query** does not terminate on all inputs, for three quite different reasons: Example 3.4 shows that on some inputs **Holds** makes infinitely many identical calls, Example 3.5 shows that on some inputs the call tree for **Query** includes an infinite path of distinct nodes; and Example 3.6 shows that on some inputs the call tree for **Query** includes a node with infinitely many distinct children.

EXAMPLE 3.2. Suppose that Alice is quietly walking beside her two giggling daughters, Betty and Bonnie. Are the three of them a quiet group? Intuitively, they are not, because Betty and Bonnie are giggling. According to **Query**, however, the answer is yes. Since Alice is quiet and **Auth** makes the subset assumption, **Query** concludes that the principal $\{\mathit{Alice}, \mathit{Betty}, \mathit{Bonnie}\}$ is quiet; that is, $\mathbf{Query}(\mathbf{Quiet}(\{\mathit{Alice}, \mathit{Betty}, \mathit{Bonnie}\}), \emptyset, \{\mathbf{Quiet}(\mathit{Alice})\}) = \mathbf{true}$. □

EXAMPLE 3.3. Suppose that Alice says that she is smart, and if Alice says that she is smart, then she is permitted to say that she is smart. Is Alice smart? Intuitively, she is, because Alice is permitted to say that she is smart and she does so. But consider $\mathbf{Query}(\mathbf{Smart}(\mathit{Alice}), L, R)$, where $L = \{(\mathit{Alice}, g)\}$, $R = \{\mathbf{Said}(\mathit{Alice}, \mathbf{Smart}(\mathit{Alice})) \rightarrow \mathbf{Perm}(\mathit{Alice}, \texttt{issue}, g)\}$, and $g = \mathbf{Smart}(\mathit{Alice})$. $\mathbf{Query}(\mathbf{Smart}(\mathit{Alice}), L, R)$ begins by calling $\mathbf{Auth}(\mathbf{Smart}(\mathit{Alice}), L, R)$. **Auth** checks whether or not Alice is permitted to issue $g$. It determines that Alice may not issue $g$, because the permission does not follow from $R$ and $L - \{(\mathit{Alice}, g)\}$. Since Alice is not permitted to issue $g$, **Auth** sets $R' = R$ and returns $\emptyset$. Because **Auth** returns $\emptyset$, **Query** returns **false**. □

EXAMPLE 3.4. Suppose that Alice issues the grant "if I say Bob is smart, then he is" and Alice is permitted to issue this grant. Can we conclude that Bob is smart?

To answer the question using **Query**, let $e = \mathbf{Smart}(\mathit{Bob})$, $g = \mathbf{Said}(\mathit{Alice}, e) \Rightarrow e$, $L = \{(\mathit{Alice}, g)\}$, and $R = \{\mathbf{Perm}(\mathit{Alice}, \mathtt{issue}, g)\}$. We are interested in the output of $\mathbf{Query}(e, L, R)$. $\mathbf{Query}(e, L, R)$ begins by calling $\mathbf{Auth}(e, L, R)$, which returns the set $D = \{\mathbf{Said}(\mathit{Alice}, e)\}$. **Query** then calls $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, e), L)$, which sets $R_{\mathit{Alice}} = \{g\}$ and calls $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, e), L)$ again. It is easy to see that an infinite number of calls to $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, e), L)$ are made during the execution of $\mathbf{Query}(e, L, R)$ and thus the execution does not terminate.

It is tempting to conclude that a set $L$ of licenses and a set $R$ of grants imply a conclusion $e$ only if $\mathbf{Query}(e, L, R)$ terminates and returns **true**. Unfortunately, whether $\mathbf{Query}(e, L, R)$ terminates can depend on the order in which the calls to **Holds** are made. To see why, consider a slight modification of the previous example where we add the grant $\{\mathbf{Smart}(\mathit{Bob})\}$ to $R$. Intuitively, this means that an implicitly trusted principal says that Bob is smart. It now seems reasonable to expect that every execution of $\mathbf{Query}(e, L, R')$ returns **true**, where $R' = R \cup \{e\}$, and $e$, $L$, and $R$ are as defined in the original example. Surely the issued grants imply that Bob is smart, since a grant issued by a trusted principal says just that! However, only some of the executions terminate. Every execution of **Query** begins by calling $\mathbf{Auth}(e, L, R')$, and every execution of $\mathbf{Auth}(e, L, R')$ returns $\{\mathbf{Said}(\mathit{Alice}, e), \mathbf{true}\}$. If an execution of **Query** next calls $\mathbf{Holds}(\mathbf{true}, L)$, then that execution of **Query** returns **true**. On the other hand, if the execution calls $\mathbf{Holds}(\mathbf{Said}(\mathit{Alice}, e), L)$ and then waits for the call to return before calling $\mathbf{Holds}(\mathbf{true}, L)$, then the execution does not terminate for the same reason that every execution of $\mathbf{Query}(e, L, R)$ does not terminate. □

EXAMPLE 3.5. Suppose that Alice says "for all grants $g$, if I say I am allowed to issue the grant $\mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, g)$, then I am allowed to issue $g$", and Alice is allowed to issue that statement. Is Alice allowed to issue the grant $\mathbf{Nap}(\mathbf{Alice})$? To answer this question using **Query**, some abbreviations are useful. For all grants $g$, we abbreviate the condition $\mathbf{Said}(\mathbf{Alice}, \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, g)))$ as $d(g)$ and we abbreviate the grant $\mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, g)$ as $h(g)$. We execute $\mathbf{Query}(e, L, R)$, where $e = \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, \mathbf{Nap}(\mathbf{Alice}))$, $R = \{\mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, \forall x(d(x) \Rightarrow \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, x)))\}$, and $L = \{(\mathbf{Alice}, \forall x(d(x) \Rightarrow \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, x)))\}$. $\mathbf{Query}(e, L, R)$ begins by calling $\mathbf{Auth}(e, L, R)$, which returns $\{d(\mathbf{Nap}(\mathbf{Alice}))\}$. Next **Query** calls $\mathbf{Holds}(d(\mathbf{Nap}(\mathbf{Alice})), L)$, which calls $\mathbf{Holds}(d(h(\mathbf{Nap}(\mathbf{Alice}))), L)$, which calls $\mathbf{Holds}(d(h(h(\mathbf{Nap}(\mathbf{Alice})))), L)$, and so on. It is not hard to see that, for all integers $n > 0$, $\mathbf{Holds}(d(h^n(\mathbf{Nap}(\mathbf{Alice}))), L)$ is called, where $h^1(g) = h(g)$ and $h^n(g) = h(h^{n-1}(g))$, for all grants $g$. It follows that **Holds** does not terminate and, thus, **Query** does not terminate. □

EXAMPLE 3.6. Suppose that Alice may say that she is trusted if Bob says that Alice may issue some grant (any grant at all). May Alice say that she is trusted? To answer this question using **Query**, we run $\mathbf{Query}(e, \emptyset, R)$, where $e = \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, \mathbf{Trusted}(\mathbf{Alice}))$, $R = \{\forall x(d(x) \rightarrow e)\}$, and $d(x) = \mathbf{Said}(\mathbf{Bob}, \mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, x))$. **Query** begins by calling $\mathbf{Auth}(e, \emptyset, R)$, which returns $D = \{d(g) \mid g \text{ is a grant}\}$. We show below that $D$ is an infinite set, so every execution of **Auth** that tries to compute $D$ does not terminate. Even if $D$ is defined without explicitly listing all of its elements, **Query** must determine if some element in $D$ holds. In fact, none do. Thus, any approach to testing if some condition in $D$ holds by explicitly testing each condition will not terminate.

It remains to show that $D = \{d(g) \mid g \text{ is a grant}\}$ is an infinite set. The key observation is that infinitely many distinct grants can be expressed in the language, even if the vocabulary consists of only one property **Pr** and one principal $p$. To see why, define grants $g_n$, $n \geq 1$, inductively by taking $g_1 = \mathbf{true} \rightarrow \mathbf{Pr}(p)$ and $g_{n+1} = \mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, g_n)) \rightarrow \mathbf{Pr}(p)$ for all $n > 0$. Since each of these grants is clearly distinct, $D$ is infinite. □

### 3.3 A Corrected Version of **Query**

In this section we revise **Query** to correct the problems observed in Section 3.2. One of the corrections is fairly straightforward. We resolve the mismatch illustrated in Example 3.2 by removing the subset assumption from **Auth**. We note that the language is sufficiently expressive to force the subset assumption, if desired, by including the following grants in $R$:

$$g = \forall x_1 \forall x_2 \forall x_3 (\mathbf{Perm}(x_1, \mathtt{issue}, x_2) \rightarrow \mathbf{Perm}(x_1 \cup x_3, \mathtt{issue}, x_2))$$
$$g_i = \forall x_1 \forall x_2 (\mathbf{Pr}_i(x_1) \rightarrow \mathbf{Pr}_i(x_1 \cup x_2)), \text{for } i = 1, \ldots, n,$$

where $x_1$, $x_2$, and $x_3$ are variables of the appropriate sorts and $\mathbf{Pr}_1, \ldots, \mathbf{Pr}_n$ are the properties in the language. We now consider Examples 3.3, 3.4, 3.5, and 3.6, in turn.

The problem illustrated in Example 3.3 lies in the definition of $R'$. Recall that we define $\mathbf{Auth}(e, L, R) = \mathbf{Auth}(e, \emptyset, R')$. Roughly speaking, $R'$ should consist of the set of grants in $R$ together with those issued by someone who has the authority to do so. In other words, $R'$ should be $R \cup \{g \mid \text{for some principal } p, (p, g) \in L \text{ and } \mathbf{Query}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R) = \mathbf{true}\}$. However, when computing $\mathbf{Query}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R)$, **Auth** is given the argument $L - \{(p, g)\}$ rather than $L$. Our solution is to do the "right" thing here, and compute $\mathbf{Query}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R)$. But now we have to deal with the problem of termination, since a consequence of our change is that $\mathbf{Query}(e, L, R)$ terminates only if the set $L = \emptyset$. To ensure termination, we modify **Auth** so that no call is evaluated twice. Specifically, the revised **Auth** takes a fourth argument $E$ that is the set of closed conditions that have been the first argument to a previous call; $\mathbf{Auth}(e, L, R, E)$ returns $\emptyset$ if $e \in E$. Because the revised **Auth** calls **Query**, which calls **Auth**, we modify **Query** to take $E$ as its fourth argument. A closed condition $e$ is implied by a set $L$ of licenses and a set $R$ of grants if the modified **Query** algorithm returns **true** on input $(e, L, R, \emptyset)$. Pseudocode for the revised version of **Query**, which we call **Query2**, and for the revised version of **Auth**, which we call **Auth2**, are given in Figures 4 and 5, respectively. **Query2** refers to the algorithm **Holds2**, which is **Holds** modified to correct the behavior seen in Example 3.4 (discussed below).

The type of nontermination seen in Example 3.4 occurs because **Query** tries to verify that a condition of the form $\mathbf{Said}(p, e)$ holds by checking if $\mathbf{Said}(p, e)$ holds. To correct the problem, we modify **Holds** to take a third argument $S$ that is the set of **Said** conditions that have been the first argument to a previous call; that is, $S$ is the set of **Said** conditions that are currently being evaluated. If the revised

**Query2**$(e, L, R, E)$:

$D :=$ **Auth2**$(e, L, R, E)$
**if Holds2**$(d, L, \emptyset) =$ **true** for a condition $d \in D$
**then** return **true**
**else** return **false**

Fig. 4. The **Query2** Algorithm

**Auth2**$(e, L, R, E)$:

**if** $e \in E$
**then** return $\emptyset$
**else**
  $E' := E \cup \{e\}$
  $R' := R$
  **for** each license $(p, g) \in L$
    **if Query2**(**Perm**$(p, \texttt{issue}, g), L, R, E') =$ **true**
    **then** $R' := R' \cup \{g\}$
  $D := \emptyset$
  **for** each grant $\forall x_1 \ldots \forall x_n (d_g \rightarrow e_g) \in R'$
    $D := D \cup \{d \mid d_g\sigma = d \text{ and } e_g\sigma = e, \text{ for some closed substitution } \sigma\}$
  return $D$

Fig. 5. The **Auth2** Algorithm

**Holds** is called with a first argument $d$ that is in $S$ (which means that the call was made when trying to determine whether $d$ holds), then the algorithm returns **false**, thereby halting the cycle. Pseudocode for the revised version of **Holds**, which we call **Holds2**, is given in Figure 6.

It is easy to see that the problem illustrated by Example 3.4 does not occur during the execution of **Holds2**. Moreover, the following theorem shows that **Holds2** is correct in the sense that every execution of **Holds** and **Holds2** have the same input/output behavior on the inputs for which both executions terminate and, if an execution of **Holds** terminates for a particular input $(d, L)$, then some execution of **Holds2**$(d, L, \emptyset)$ terminates as well.

PROPOSITION 3.7. *For all closed conditions $d$ and sets $L$ of licenses,*

*(a) every execution of **Holds**$(d, L)$ that terminates returns the same output,*
*(b) every execution of **Holds2**$(d, L, \emptyset)$ that terminates returns the same output,*
*(c) if an execution of **Holds**$(d, L)$ terminates by returning the truth value $t$, then an execution of **Holds2**$(d, L, \emptyset)$ terminates by returning $t$.*

Now consider Examples 3.5 and 3.6. To address the type of nontermination seen in these examples, we might hope to find an algorithm **Query3** that returns the same output as **Query2** on inputs for which an execution of **Query2** terminates

**Holds2**$(d, L, S)$:

**if** $d =$ **true**
**then** return **true**

**if** $d = d_1 \wedge \ldots \wedge d_n$
**then** return $\bigwedge_{i=1,\ldots,n}$ **Holds2**$(d_i, L, S)$

**if** $d =$ **Said**$(p, e)$ and $d \in S$
**then** return **false**

**if** $d =$ **Said**$(p, e)$ and $d \notin S$
**then**
  $S' = S \cup \{d\}$
  $R_p = \{g \mid$ for some principal $p', (p', g) \in L$ and $p' \in p\}$
  $D := \{d' \mid$ for some grant $\forall x_1 \ldots \forall x_n (d_g \rightarrow e_g) \in R_p$ and closed substitution $\sigma$, $d_g\sigma = d'$ and $e_g\sigma = e\}$
  **if Holds2**$(d', L, S') =$ **true** for a condition $d' \in D$
  **then** return **true**
  **else** return **false**

Fig. 6. The **Holds2** Algorithm

and returns **false** on all other inputs. Returning **false** when no execution of **Query2** terminates gives an intuitively reasonable answer; moreover, this approach is essentially what is done in MPEG-21 REL (see Section 8 for details). Unfortunately, as we show shortly (see Theorem 5.1) this approach will not work in general; there is no algorithm **Query3** with these properties, since whether **Query2** terminates on a given input is undecidable.

Since we cannot "fix" **Query2**, the best we can do is define some restrictions such that, if the restrictions hold for a particular query, then the problems seen in Examples 3.5 and 3.6 do not occur for that query. We now describe some conditions that are sufficient and that we suspect often hold in practice.

To describe our approach for avoiding the problem seen in Example 3.5, let $g$ and $g'$ be the grants $\forall x_1 \ldots \forall x_n (d_g \rightarrow e_g)$ and $\forall x_1 \ldots \forall x_m (d_{g'} \rightarrow e_{g'})$ respectively. The license $(p, g)$ *affects* the license $(p', g')$ if and only if there are closed substitutions $\sigma$ and $\sigma'$ such that a condition of the form **Said**$(p'', e_g\sigma)$ is mentioned in $d_{g'}\sigma'$ and $p \subseteq p''$. For example, consider the license set $L = \{(\mathbf{Alice}, g_1), (\mathbf{Amy}, g_2)\}$, where $g_1 = \mathbf{Smart}(\mathbf{Bob})$ and $g_2 = \forall x(\mathbf{Said}(\mathbf{Alice}, \mathbf{Smart}(x)) \Rightarrow \mathbf{Attractive}(x))$. The license $(\mathbf{Alice}, g_1)$ affects the license $(\mathbf{Amy}, g_2)$ because the conditions are satisfied if $\sigma$ is a closed substitution and $\sigma'$ is a closed substitution such that $\sigma'(x) = \mathbf{Bob}$. A set $L$ of licenses is *hierarchical* if there exists a strict partial order $\prec$ on the licenses in $L$ such that, for all license $\ell, \ell' \in L$, if $\ell$ affects $\ell'$ then $\ell \prec \ell'$. Continuing our example, $L$ is hierarchical because the ordering $(\mathbf{Alice}, g_1) \prec (\mathbf{Amy}, g_2)$ satisfies the requirements. Observe that no hierarchical license set includes the license $(\mathbf{Alice}, \mathbf{Said}(\mathbf{Alice}, e) \Rightarrow e)$ because this license affects itself. The license set in

Example 3.5 is not hierarchical for essentially the same reason. It is not hard to see that by restricting the set of queries $(e, L, R, E)$ to those in which $L$ is hierarchical, we avoid the type of circularity that causes the problem seen in Example 3.5. In the next result and elsewhere, we use $\#(X)$ to denote the cardinality of a set $X$.

PROPOSITION 3.8. *If $d$ is a closed condition, $L$ is a hierarchical set of licenses, $S$ is a set of closed* **Said** *conditions, and $T$ is the call tree of an execution of* ***Holds2***$(d, L, S)$*, then the height of $T$ is at most $2\#(L) + 1$.*

We further restrict the language to avoid the problem seen in Example 3.6. To understand our restriction, recall that $\mathbf{Auth}(e, L, R)$ first extends $R$ to $R'$ by adding all the grants that are issued by someone who has the authority to do so. Since all the grants in $R' - R$ are in $L$, the set $R'$ must be finite. Then **Auth** creates the possibly infinite set $R_\Sigma$ consisting of all substitution instances of grants in $R'$, and returns $\{d \mid d \rightarrow e \in R_\Sigma\}$. (For simplicity here, we are assuming that **Auth** does not use the subset assumption; the subset assumption does not affect our discussion.) Since **Auth** considers only the grants in $R_\Sigma$ whose conclusion matches the first input to **Auth**, we could certainly replace $R_\Sigma$ by $R'_\Sigma$, where

$$R'_\Sigma = \{d_g\sigma \rightarrow e \mid \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in R', \sigma \text{ is a closed substitution, and } e_g\sigma = e\}.$$

Because $e$ is closed, $R'_\Sigma$ is finite if, for every grant $g$ in $R'$, if the condition of $g$ mentions a free variable $x$, then either $x$ ranges over a finite set or $x$ appears in the conclusion of $g$. Our solution is simply to restrict the language so that every grant has this property. Since, in our fragment, there are infinitely many resources (grants) and only finitely many principles, this amounts to restricting the language so that if $\forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$ is a grant, then every free variable of sort *Resource* that appears in $d_g$ also appears in $e_g$. We call a grant *restrained* if it has this property; we call a license $(p, g)$ restrained if $g$ is restrained. Thus, for example, $\forall x \forall y(\mathbf{Said}(\emptyset, \mathbf{Perm}(x, \mathtt{issue}, y)) \rightarrow \mathbf{Perm}(Alice, \mathtt{issue}, y))$ is restrained, but neither

$$\forall y \forall z(\mathbf{Said}(\emptyset, \mathbf{Perm}(Alice, \mathtt{issue}, y)) \rightarrow \mathbf{Perm}(Alice, \mathtt{issue}, z))$$

nor the grant $\forall x(d(x) \Rightarrow e)$ in Example 3.6 is restrained. It is easy to see that, for all restrained grants $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$ and closed conclusions $e$, if $n$ is the number of primitive principals in the language and $|g|$ is the length of $g$, then there are at most $n^{|g|}$ grants of the form $d_g\sigma \rightarrow e_g\sigma$ such that $\sigma$ is a closed substitution and $e_g\sigma = e$. Thus, by considering only restrained grants and licenses, we solve the problem raised in Example 3.6.

## 4. FORMAL SEMANTICS

In this section we provide formal semantics for the XrML fragment described in Section 2. We show that the semantics is correct in the sense that it captures the output of the (corrected) query algorithm, **Query2**. We then consider two, arguably more intuitive, semantics and show that neither captures **Query2**..

### 4.1 A Correct Translation

To give formal semantics to our fragment, we translate licenses in the grammar to formulas in a modal many-sorted first-order logic. The logic has three sorts:

*Principal*, *Right*, and *Resource*. The vocabulary includes the following symbols, where *primitivePrin* is the application-provided set of primitive principals and *primitiveProp* is the application-provided set of properties:

—a constant $p$ of sort *Principal* for every principal $p \in \mathit{primitivePrin}$;

—a constant `issue` of sort *Right*;

—a ternary predicate **Perm** that takes arguments of sort *Principal*, *Right*, and *Resource*;

—a unary predicate **Pr** that takes an argument of sort *Principal* for each property $\mathbf{Pr} \in \mathit{primitiveProp}$;

—a function $\cup : \mathit{Principal} \times \mathit{Principal} \longrightarrow \mathit{Principal}$;

—a function $f_g : s_1 \times \ldots \times s_n \longrightarrow \mathit{Resource}$ for each grant $g$ in the language; if $x_1, \ldots, x_n$ are the free variables in $g$, then $x_i$ is of sort $s_i$, for $i = 1, \ldots, n$. If $g$ is closed, then the corresponding function is a constant that we denote as $c_g$; and

—a modal operator $\mathsf{Val}$ that takes a formula as its only argument.

Intuitively, $\mathbf{Pr}(p)$ means principal $p$ has property **Pr**, and $\mathsf{Val}(\varphi)$ means formula $\varphi$ is valid. Notice that every principal in the grammar corresponds to a term in the language, because $\cup$ is a function symbol.

The semantics of our language is just the standard semantics for first-order logic, extended to deal with $\mathsf{Val}$. We restrict attention to models for which $\cup$ satisfies the following standard properties:

U1. $\forall x((x \cup x) = x)$

U2. $\forall x_1 \forall x_2((x_1 \cup x_2) = (x_2 \cup x_1))$

U3. $\forall x_1 \forall x_2 \forall x_3((x_1 \cup (x_2 \cup x_3)) = ((x_1 \cup x_2) \cup x_3))$

U4. $\forall x((x \cup \emptyset) = x)$

We call such models *acceptable*. $\mathsf{Val}(\varphi)$ is true in a model $m$ if $\varphi$ is true in all acceptable models. If a formula $\varphi$ is true in all acceptable models, then we say that $\varphi$ is *acceptably valid*. Thus, $\mathsf{Val}(\varphi)$ is true in an acceptable model iff $\varphi$ is acceptably valid.

The translation takes four finite sets as parameters. They are a set $L$ of licenses, a set $A$ of closed resources, a set $S$ of closed **Said** conditions, and a set $E$ of closed conclusions. Roughly speaking, $L$ is the set of licenses that have been issued and $A$ is the set of resources that are relevant to a particular application For all XrML queries, $S = \emptyset$ and $E = \emptyset$. (The reader is encouraged to take $S = E = \emptyset$ when first trying to understand the details of the semantics.) The input parameter $S$ allows users to specify a set of **Said** conditions that do not hold, regardless of $L$. We also use the parameter to insure that the translation of a **Said** condition does not enter an infinite loop. The input parameter $E$ corresponds to the fourth argument of **Query2**. (Recall that an XrML query asks if a conclusion $e$ follows from a set $L$ of licenses and set $R$ of grants; the answer is "yes" if $\mathbf{Query2}(e, L, R, \emptyset)$ returns **true**.) By including $E$, we can give a translation that agrees with the **Query2** algorithm. The translation is defined below, where $s^{L,A,S,E}$ is the translation of the string $s$ given input $L$, $A$, $S$, and $E$.

—If $\mathbf{Perm}(p, \mathtt{issue}, g) \in E$ or $(p, g) \notin L$, then $(p, g)^{L,A,S,E} = \mathbf{true}$.

—If $\mathbf{Perm}(p, \mathtt{issue}, g) \notin E$ and $(p, g) \in L$, then $(p, g)^{L,A,S,E} = \mathbf{Perm}(p, \mathtt{issue}, c_g) \Rightarrow g^{L,A,S,E}$. Note that we assume $g$ is closed, because this assumption is built into **Query**.

—$(d_g \rightarrow e_g)^{L,A,S,E} = ((\bigwedge_{e \in E} \neg \mathsf{Val}(e^{L,A,S,E} \Leftrightarrow e_g^{L,A,S,E})) \wedge d_g^{L,A,S,E}) \Rightarrow e_g^{L,A,S,E}$.

—$(\forall x \varphi)^{L,A,S,E} = \bigwedge_{t \in T} (\varphi[x/t])^{L,A,S,E}$, where $T = A$ if $x$ is of sort *Resource*, and $T = P$ if $x$ is of sort *Principal*. (Recall that $P$ is the set of principals.)

—$\mathbf{true}^{L,A,S,E} = \mathbf{true}$.

—If $\mathbf{Said}(p, e)^{L,A,S,E} \in S$, then $\mathbf{Said}(p, e)^{L,A,S,E} = \mathbf{false}$.

—If $\mathbf{Said}(p, e)^{L,A,S,E} \notin S$, then $\mathbf{Said}(p, e)^{L,A,S,E} = \mathsf{Val}((\bigwedge_{g \in R_p} g^{L,A,S',\emptyset}) \Rightarrow e^{L,A,S',\emptyset})$, where $R_p = \{g \mid (p', g) \in L \text{ for a } p' \in p\}$ and $S' = S \cup \{\mathbf{Said}(p, e)\}$.

—$(d_1 \wedge d_2)^{L,A,S,E} = d_1^{L,A,S,E} \wedge d_2^{L,A,S,E}$.

—$\mathbf{Perm}(p, r, s)^{L,A,S,E} = \mathbf{Perm}(p, r, s^*)$, where $s^* = s$ if $s$ is a variable of sort *Resource*, $s^* = c_s$ if $s$ is a closed grant, and $s^* = f_s(x_1, \ldots, x_n)$ if $s$ is an open grant with free variables $x_1, \ldots, x_n$.

—$\mathbf{Pr}(p)^{L,A,S,E} = \mathbf{Pr}(p)$.

—for every principal $p$, $\{p\}^{L,A,S,E} = p$.

This translation has two features that seem somewhat inelegant. The first is that, in dealing with a universal quantifier, variables are replaced by the constants over which they range; the second is the use of the $\mathsf{Val}$ operator. In the next section, we explain in more detail why we translated in this way. For now, we show that, in a precise sense, our translation captures the intended interpretation of the language.

Note that $\mathbf{Said}(p, e)^{L,A,S,E}$ does not depend on $E$. This matches our intuition that the meaning of a **Said** condition depends only on what principals have said, rather than on what is actually true. By adding $\mathbf{Said}(p, e)$ to $S$, we ensure that the meaning of the condition does not depend on itself. Finally, observe that $\mathbf{Said}(p, e)^{L,A,S,E}$ is defined in terms of the translation of potentially more complex expressions. Nevertheless, the following result shows that the translation is well defined.

THEOREM 4.1. *For all strings $s$ in the language and all finite sets $L$ of licenses, $A$ of closed resources, $S$ of closed* **Said** *conditions, and $E$ of closed conclusions, $s^{L,A,S,E}$ is well defined.*

We believe that our semantics captures the intended meaning of XrML expressions, as implied by the specification. To make this precise, we show that **Query2** agrees with the semantics on all queries. Specifically, we show that for all terminating executions $X$ of $\mathbf{Query2}(e, L, R, E)$, $X$ returns **true** iff $\bigwedge_{\ell \in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g \in R} g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid, where $A = A(e, L, R, E, X)$ is the set of closed resources that appear in the first argument of a call to **Query2**, **Auth2**, or **Holds2** during execution $X$. Intuitively, $A$ is the set of resources relevant to answering the query $(e, L, R, E)$. For example, suppose that, during a particular execution $X$ of $\mathbf{Query2}(e, L, R, E)$, $\mathbf{Holds2}(\mathbf{Said}(p, \mathbf{Perm}(p', \mathtt{issue}, \mathbf{Perm}(p'', \mathtt{issue}, g))), L, S)$ is called. Then $A(e, L, R, E, X)$ includes $\mathbf{Perm}(p'', \mathtt{issue}, g)$ and $g$. Notice that if $X$ is a terminating execution, then $A(e, L, R, E, X)$ is finite.

THEOREM 4.2. *Suppose that $(e, L, R, E)$ is a query and $X$ is a terminating execution of **Query2**$(e, L, R, E)$. Then $X$ returns* **true** *iff*

$$\bigwedge_{\ell \in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g \in R} g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$$

*is acceptably valid, where $A = A(e, L, R, E, X)$.*

### 4.2 Two Alternative Translations

We now discuss why we captured universal quantification by replacing variables by constants and the need for the Val operator. We do so by giving two arguably more natural alternative translations that do not have these "features", and showing where they go wrong. While this does not show that there is no correct translation that translates universal quantification as universal quantification, and does not use Val, it does show why finding such a translation is nontrivial.

For all strings $s$ in our fragment, let $s_1^{L,A,S,E}$ be a translation of $s$, where $L$, $A$, $S$, and $E$ are as defined in Section 4.1. The formula $s_1^{L,A,S,E}$ is identical to $s^{L,A,S,E}$ except that $(\forall x\varphi)_1^{L,A,S,E} = \forall x(\varphi_1^{L,A,S,E})$. Notice that the new translation often leads to more concise formulas and does not depend on the input parameter $A$. Unfortunately, this translation does not interact well with our use of Val when it comes to universally quantified formulas involving **Said**. The following example shows why we rejected this translation.

EXAMPLE 4.3. Suppose that Alice may issue any grant. Alice issues the grants "if I say some principal $p$ is great, then $p$ is also good", "if I say Bob is good, then Charlie is great", and "Bob is great." Can we conclude that Charlie is good?

To answer our question using **Query2**, let $L = \{(Alice, g_A), (Alice, g_B), (Alice, g_C)\}$ and consider $\textbf{Query2}(\textbf{Good}(Charlie), \{(Alice, g_A), L, R, \emptyset)$, where

$$\begin{aligned} g_A &= \forall x(\textbf{Said}(Alice, \textbf{Great}(x)) \rightarrow \textbf{Good}(x)), \\ g_B &= \textbf{Said}(Alice, \textbf{Good}(Bob)) \rightarrow \textbf{Great}(Charlie), \\ g_C &= \textbf{Great}(Bob) \\ R &= \{\forall x(\textbf{Perm}(Alice, \texttt{issue}, x))\} \end{aligned}$$

It is not hard to see that the algorithm returns **true** (i.e., Charlie is good), which is the intuitively correct answer. Roughly speaking, the algorithm deduces that Charlie is good if Alice says he is great; Alice says Charlie is great if Alice says Bob is good; Alice says Bob is good if Alice says Bob is great; and Alice does indeed say Bob is great.

To answer our question using the revised translation, we need to determine the validity of the formula

$$(\bigwedge_{\ell \in L} \ell_1^{L,\emptyset,\emptyset,\emptyset} \wedge \forall x(\textbf{Perm}(Alice, \texttt{issue}, x))) \Rightarrow \textbf{Good}(Charlie).$$

It is easy to see that this formula equivalent to

$$(g_A)_1^{L,\emptyset,\emptyset,\emptyset} \wedge (g_B)_1^{L,\emptyset,\emptyset,\emptyset} \wedge (g_C)_1^{L,\emptyset,\emptyset,\emptyset} \Rightarrow \textbf{Good}(Charlie).$$

Clearly, $(g_C)_1^{L,\emptyset,\emptyset,\emptyset} = (g_C)^{L,\emptyset,\emptyset,\emptyset} = \textbf{Great}(Bob)$. In the original translation, we combine $\textbf{Great}(Bob)$ with $(g_A)^{L,\emptyset,\emptyset,\emptyset}$ to conclude $\textbf{Great}(Bob)$, then combine

**Great**(*Bob*) with $(g_A)^{L,\emptyset,\emptyset,\emptyset}$ to derive **Good**(*Charlie*); that is, the formula corresponding to the query is valid under the original translation. Unfortunately, the latter two steps fail with the revised translation. As suggested above, the problem lies in the interaction of quantified formulas and **Said** in $g_A$. Consider the first step. Note that $(g_A)^{L,\emptyset,\emptyset,\emptyset}$ is equivalent to a conjunction of which one conjunct is $(\mathbf{Said}(\mathit{Alice}, \mathbf{Great}(\mathit{Bob})) \rightarrow \mathbf{Good}(\mathit{Bob}))$. When combined with **Great**(*Bob*), we can indeed conclude **Good**(*Bob*). On the other hand, with the revised translation, $(g_A)_1^{L,\emptyset,\emptyset,\emptyset}$ is

$$\forall x(\mathsf{Val}((g_B)_1^{L,\emptyset,\emptyset,\emptyset} \wedge (g_C)_1^{L,\emptyset,\emptyset,\emptyset} \Rightarrow \mathbf{Great}(x)) \Rightarrow \mathbf{Good}(x)).$$

The Val formula is vacuously false, so $(g_A)_1^{L,\emptyset,\emptyset,\emptyset}$ is vacuosuly true, and does not help in concluding **Good**(*Bob*). Thus, the formula corresponding to the query is not valid under the revised translation; we do not get the intuitively correct answer. □

Next, suppose that we modify our original translation so that the Val operator is not used. In particular, we fix the input parameter $E$ to be the empty set and remove the validity operator from the translation of **Said** conditions. For all strings $s$, let $s_2^{L,A,S,\emptyset}$ be the translation of $s$ that is identical to $s^{L,A,S,\emptyset}$ except that, if $s$ is of the form "$d_g \rightarrow e_g$", then $s_2^{L,A,S,\emptyset} = d_{g2}^{L,A,S,\emptyset} \Rightarrow e_{g2}^{L,A,S,\emptyset}$ and, if $s$ is of the form $\mathbf{Said}(p,e)$ and $s \notin S$, then $\mathbf{Said}(p,e)_2^{L,A,S,\emptyset} = (\bigwedge_{g \in R_p} g_2^{L,A,S',\emptyset}) \Rightarrow e_2^{L,A,S',\emptyset}$, where $R_p = \{g \mid (p', g) \in L \text{ for a } p' \in p\}$ and $S' = S \cup \{\mathbf{Said}(p,e)\}$. Observe that every translated string is a variable-free formula in first-order logic. The following example illustrates a problem with this translation. Roughly speaking, the problem is that, according to the translation, every statement that follows from the given licenses and grants is said by every principal.

EXAMPLE 4.4. Suppose that Alice cheated on an exam and, if Alice admits that she cheated, then she is trusted. Is Alice trusted? Intuitively, the answer is "no" because Alice has not confessed.

To answer the question using **Query2**, we execute $\mathbf{Query2}(\mathbf{Trusted}(\mathit{Alice}), \emptyset, R, \emptyset)$, where $R = \{\mathbf{Cheated}(\mathit{Alice}), \mathbf{Said}(\mathit{Alice}, \mathbf{Cheated}(\mathit{Alice})) \rightarrow \mathbf{Trusted}(\mathit{Alice})\}$. It is not hard to see that **Query2** returns **false**, indicating that Alice is not trusted. Specifically, the algorithm determines that Alice is trusted only if Alice said she cheated and Alice has not done this.

To answer the question using the revised translation, we determine the validity of the formula $(\mathbf{Cheated}(\mathit{Alice}) \wedge ((\mathbf{true} \Rightarrow \mathbf{Cheated}(\mathit{Alice})) \Rightarrow \mathbf{Trusted}(\mathit{Alice}))) \Rightarrow \mathbf{Trusted}(\mathit{Alice})$. Standard manipulations show that the formula is logically equivalent to $(\mathbf{Cheated}(\mathit{Alice}) \wedge (\mathbf{Cheated}(\mathit{Alice}) \Rightarrow \mathbf{Trusted}(\mathit{Alice}))) \Rightarrow \mathbf{Trusted}(\mathit{Alice})$, which is valid. So, if we use the revised translation, we conclude that Alice is trusted.

If we use the translation in Section 4.1, then we determine that Alice is not trusted. This is because Val "isolates" the **Said** condition from the statements implied by the given grants and the issued licenses. As a result, $\mathbf{Said}(\mathit{Alice}, \mathbf{Cheated}(\mathit{Alice}))$ holds only if the grants issued by Alice, in isolation, imply **Cheated**(*Alice*); that is, $\mathbf{Said}(\mathit{Alice}, \mathbf{Cheated}(\mathit{Alice}))$ holds only if $\mathsf{Val}(\mathbf{true} \Rightarrow \mathbf{Cheated}(\mathit{Alice}))$ is **true**.

Since **true** $\Rightarrow$ **Cheated**$(Alice)$ is not an acceptably valid formula, we conclude that **Said**$(Alice,$ **Cheated**$(Alice))$ does not hold and, thus, **Trusted**$(Alice)$ does not hold. $\square$

## 5. COMPLEXITY

To answer a query $(e, L, R, E)$, we need to determine whether an execution of **Query2**$(e, L, R, E)$ returns **true**. We claimed earlier that the problem of answering queries is, in general, undecidable. We now formalize this claim. Recall that a grant $g$ is restrained if every variable of sort *Resource* mentioned in the antecedent of $g$ is mentioned in the conclusion of $g$. We say that *a grant $g$ is in a set $L$ of licenses* if $(p, g) \in L$ for some principal $p$. A grant $g$ is in $R \cup L$, for some set $R$ of grants, if $g$ is in $R$ or $g$ is in $L$.

THEOREM 5.1. *Determining whether some execution of **Query2**$(e, L, R, E)$ returns **true** is undecidable for the set of queries $(e, L, R, E)$ such that at most one grant in $R \cup L$ is not restrained.*

Let $\mathcal{L}_0$ be the set of queries $(e, L, R, E)$ such that every grant in $R \cup L$ is restrained. In this section, we examine the computational complexity of answering queries for fragments of $\mathcal{L}_0$.

We first show that the problem of answering queries for the full language $\mathcal{L}_0$ is NP hard for two quite different reasons. The first stems from the fact that, if there are $n$ primitive principals, we can construct $2^n$ principals using the $\cup$ operator. The second is that, to answer a query, we might need to determine if exponentially many closed **Said** conditions hold.

We use the following definitions to state our results. $\mathcal{L}_1$ is the set of queries that do not mention the $\cup$ operator. A grant $g$ is *$n$-restricted* if the number of variables of sort *Principal* that are mentioned in the antecedent of $g$ and not in the conclusion of $g$ is at most $n$. $\mathcal{L}_2^n$ is the set of queries $(e, L, R, E)$ such that all grants in $R \cup L$ are $n$-restricted. A call **Holds2**$(d, L, S)$ is *$h$-bounded* if the call tree for every execution of **Holds2**$(d, L, S)$ has height at most $h$. Note that Proposition 3.8 shows that if $L$ is a hierarchical set of licenses, then **Holds2**$(d, L, S)$ is $(2\#(L) + 1)$-bounded. $\mathcal{L}_3^h$ is the set of queries $(e, L, R, E)$ such that if an execution of **Query2**$(e, L, R, E)$ calls **Holds2**$(d, L, S)$, then **Holds2**$(d, L, S)$ is $h$-bounded. The next result shows that deciding if at least one execution of **Query2** returns **true** is hard, even if we restrict to queries in $\mathcal{L}_0$ that satisfy any two of the following: the union operator is not mentioned (i.e., restrict to $\mathcal{L}_1$), the query is $n$-restricted for some fixed $n$, or all calls made during an execution of the query are $h$-bounded for some fixed $h$. (We show shortly that the set of queries in $\mathcal{L}_0$ that satisfy all three restrictions is tractable.)

For a formula $\varphi$, let $|\varphi|$ be the length of $\varphi$ when viewed as a string of symbols. For a set $S$, let $|S|$ be the length of $S$; that is $|S| = \Sigma_{s \in S} |s|$. Finally, we abbreviate *primitivePrin*, the set of primitive principals, as $P_0$.

THEOREM 5.2. *The problem of deciding if some execution of **Query2**$(e, L, R, E)$ returns **true** for $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L} \cap \mathcal{L}'$ is NP-hard for $\mathcal{L}, \mathcal{L}' \in \{\mathcal{L}_1, \mathcal{L}_2^0, \mathcal{L}_3^2\}$.*

If we make all three restrictions (that is, restrict to queries in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$, for some fixed $n$ and $h$), then determining whether a query returns **true** is decidable in polynomial time. However, as we might expect in light of Theorem 5.2, the degree of the polynomial depends on $n$ and $h$, and the polynomial involves constants that are exponential in $n$ and $h$. Note that, for queries in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$, all executions of **Query2** terminate and return the same answer. Termination is fairly easy to show since every call tree of an execution of **Query2**$(e, L, R, E)$ has a finite branching factor if $(e, L, R, E) \in \mathcal{L}_0$, and has finite height if $(e, L, R, E) \in \mathcal{L}_3^h$. The fact that all executions of **Query2**$(e, L, R, E)$ return the same output for all queries $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$ follows easily from Proposition 3.7(b).

THEOREM 5.3. *For fixed $n$ and $h$, if $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$ then determining whether* ***Query2***$(e, L, R, E)$ *returns* **true** *takes time* $O(|L||E| + (|R| + |L|)(|L|^{h-1}(|L| + |R| + |e|)^2))$.

The big-O notation is hiding some rather complex (and uninformative) terms that are functions of $n$ and $h$; we spell these out in the appendix.

In practice, we believe that queries are often in $\mathcal{L}_0$ and, as shown in Proposition 3.8, if we restrict to queries where the set $L$ of licenses has size at most $h$ and is hierarchical (which we expect in practice will often be the case), than all call trees that arise are guaranteed to have height at most $2h+1$. Thus, in practice, we expect that we can restrict to queries in $\mathcal{L}_2^n$ and $\mathcal{L}_3^h$ for relatively small values of $n$ and $h$. Moreover, even for larger values of $n$ and $h$ (say, as large as 10), as long as the union operator does not appear, we expect that queries can be answered efficiently, because the upper bound is quite conservative.

How reasonable is it to restrict to queries in $\mathcal{L}_1$ that do not mention the $\cup$ operator? We believe that XrML without the $\cup$ operator is sufficiently expressive for many applications. To examine the effect of not using the $\cup$ operator, note that principals appear as the first argument in a license, in a **Said** condition, and in a conclusion.

—According to the XrML documentation, the license $(\{p_1, \ldots, p_n\}, g)$ is an abbreviation for the set of licenses $\{(p, g) \mid p \in \{p_1, \ldots, p_n\}\}$. It follows that we can restrict the first argument of licenses to primitive principals and variables without sacrificing any expressive power. (In fact, we can restrict the first argument of licenses to only primitive principals, because **Query** assumes that if $(p, g)$ is a license in $L$, then $p$ is variable-free.)

—We can replace all conditions of the form **Said**$(\{p1, \ldots, p_n\}, e)$, where $p_1, \ldots, p_n$ are primitive principals, by a condition **Said**$(\{p_1, \ldots, p_n\}^*, e)$, where $\{p_1, \ldots, p_n\}^*$ is a new primitive principal, and then expand the set $L$ of issued licenses by adding a new license $(\{p_1, \ldots, p_n\}^*, g)$ for every license $(p, g)$ already in $L$, where $p \in \{p_1, \ldots, p_n\}$. It is not hard to show that this results in at most a quadratic increase in the number of grants. Thus, as long as the first argument to **Said** is variable-free, we can express it without using $\cup$.

—To understand the impact of our restriction on conclusions, we need to consider the meaning of statements such as **Trust**$(\{Alice, Bob\})$ and **Perm**$(\{Alice, Bob\}, issue, g)$. According to the XrML document, **Trust**$(\{Alice, Bob\})$ means Alice and Bob together (i.e., when viewed as a single entity) is trusted; **Perm**$(\{Alice, Bob\}, issue, g)$ means Alice and Bob is permitted to issue $g$. However, the XrML document does not explain precisely what it means for Alice and Bob to be viewed as a single entity. Indeed, it seems to treat this notion somewhat inconsistently (recall the inconsistent use of the subset assumption). There are other difficulties with sets. Notice that if $\{Alice, Bob\}$ is permitted to issue a grant, then presumably $g$ holds if $\{Alice, Bob\}$ issues $g$. However, according to the XrML documentation, the license $(\{Alice, Bob\}, g)$ is simply an *abbreviation* for the set of licenses $\{(\{Alice\}, g), (\{Bob\}, g)\}$. So it is unclear whether a principal that is not a singleton can issue a license. Furthermore, if principals that are not singletons can issue grants and $\{Alice, Bob\}$ is permitted to issue a grant $g$, then it seems reasonable to conclude that $g$ holds if $g$ is issued by both Alice and Bob, but it is not clear whether $g$ holds if it is issued by only Alice (or by only Bob).

There may well be applications for which these notions have an obvious and clear semantics. But we suspect that such applications typically include only a relatively small set of groups of interest. In that case, it may be possible to simply take these groups to be new primitive principals, and express the relationship between the group and its elements in the language. (This approach has the added advantage of forcing license writers to be clear about the semantics of groups.)

In short, we are optimistic that many applications do not need the union function.

## 6. THE ENTIRE XRML LANGUAGE

XrML has several components that are not in our fragment. Most have been excluded simply for ease of exposition. That is, our work can be extended in a straightforward way to a much larger fragment of XrML. In this section we list the main omissions, briefly discussing each one. Giving formal semantics to the entire XrML language remains an open problem.

—XrML supports *patterns*, where a pattern restricts the terms over which a variable ranges. For example, if the variable $x$ is restricted to the pattern "ends in Simpson", then $x$ ranges over the terms that meet this syntactic constraint (e.g., $x$ ranges over $\{HomerSimpson, MargeSimpson, \ldots\}$). Our semantics includes the patterns that correspond to properties in our fragment. Continuing the example, we could capture the pattern "ends in Simpson" by having the property **Simpson** in the language and having the set of grants determine which terms have the property.

XrML also allows a pattern to be a set of patterns. We can express a set of patterns as a conjunction of patterns. Since we can express conjunctions of properties in our fragment, we can also capture sets of the corresponding patterns. Patterns can be written in any language that the writer chooses. The default is to write patterns as XPath expressions. First-order logic is not well-suited to capturing XPath expressions; the situation may be even worse with other languages. Therefore we do not believe our semantics can be easily extended to include all patterns. The significance of this limitation is not yet clear.

—XrML supports *delegable grants*. A delegable grant $g$ can be viewed as a conjunction of a grant $g'$ in our fragment and a set $G$ of grants that, essentially, allow other principals to issue $g'$. For example, the delegable grant "Doctor Alice may view Charlie's medical file and she may also give the right to view the file to her colleague, Doctor Bob" can be viewed as the conjunction of the grant "Doctor Alice may view Charlie's medical file" and the grant "Alice is permitted to issue the grant 'Doctor Bob may view Charlie's medical file' ".

The XrML specification also supports more general types of delegation. For example, in XrML, we can say "Doctor Alice may view Charlie's medical file and may delegate this right to anyone under any condition that she specifies." The extent to which our semantics can capture delegation, as defined in the XrML specification, is an open problem.

—XrML supports *grantGroups*, where a grantGroup is a set of grants. We can extend our syntax to support grantGroups by closing the set of grants (as currently defined) under the union operator. Note that our proposed treatment of grantGroups is quite similar to our current treatment of principals.

—XrML has variables that range over conditions. It is not clear how this capability is intended to be used in practice. Our hope is that the practical applications will translate easily to our fragment. Examining this issue is left as an open problem.

—XrML includes rights, resources, and conditions that are not in our fragment. There should be no difficulty in extending our translation to handle these new features, and proving an analogue of Theorem 4.2. But we might not be able to answer queries in the extended language. The problem is that XrML allows resource terms to be formed by applying functions other than $\cup$. For example, MPEG-21 REL extends XrML by defining a *container* resource that is a sequence of resources. This naturally translates to a function container:$Resource \times Resource \longrightarrow Resource$, so that the container $\langle s_1, s_2, s_3 \rangle$ is translated as $\mathrm{container}(s_1, \mathrm{container}(s_2, s_3))$. Allowing such functions makes the problem of deciding if a conclusion follows from a set of XrML licenses and grants undecidable, for much the same reason that the validity problem for negation-free Datalog with function symbols is undecidable [Nerode and Shore 1997].

—XrML allows an application to define additional principals, rights, resources, and conditions within the XrML framework. Obviously, we cannot analyze terms that have yet to be defined; however, we do not anticipate any difficulty in extending the translation to deal with these terms and getting an analogue of Theorem 4.2.

—XrML allows licenses to be encrypted and supports abbreviations via the *Inventory* component. However, the XrML procedure for determining if a permission follows from a set of licenses assumes that all licenses are unencrypted and all abbreviations have been replaced by the statements for which they stand. In other words, these features are engineering conveniences that are not part of understanding or reasoning about licenses.

## 7. NEGATION

We believe that many license writers will find it important to deny permissions explicitly and to state conclusions based on whether a permission is granted, denied, or neither granted nor denied by a particular principal. For example, Alice's mother might want to say "Alice is not permitted to enter the adult website", a teacher might want to say "if the university does not object, then Alice is permitted to audit the class", and a lawyer might want to say "if the hospital permits an action that the government forbids, then the hospital is not compliant".

We can write these statements in XrML by using special "negated predicates". For example, we can write $\mathbf{Prohibited}(\mathbf{Alice}, \mathbf{enter}, \mathbf{adult\ website})$ to capture "Alice is not permitted to enter the adult website"[1], $\mathbf{NotSaid}(\mathbf{University}, \mathbf{Prohibited}(\mathbf{Alice}, \mathbf{audit}, \mathbf{class}))$ to capture "the university does not say that Alice is not permitted to audit the class" (i.e., the university does not object to Alice auditing the class), and $\mathbf{NotCompliant}(\mathbf{Hospital})$ to capture "the hospital is not compliant". We remark that this approach of using "negated predicates" has appeared before in the literature [Jajodia et al. 1997; Becker and Sewell 2004]; it is essentially the technique used by XACML [Moses 2005], another popular license language.

Adding negated predicates to XrML is straightforward; reasoning about statements in the extended language is not. One problem is that we have to handle statements that are intuitively inconsistent. For example, consider the grants $\mathbf{Perm}(\mathbf{Alice}, \mathtt{issue}, g)$ and $\mathbf{Prohibited}(\mathbf{Alice}, \mathtt{issue}, g)$, which say that Alice is permitted and prohibited to issue the grant $g$. It is not clear what we should conclude from these grants. In particular, it is not clear if Alice should be allowed to issue $g$. (The languages that include negated predicates typically require the policy writer to specify how inconsistencies should be resolved.)

Other problems arise if we extend XrML so that the set of conditions includes $\mathbf{Pr}(p)$ and $\mathbf{NotPr}(p)$, in addition to $\mathbf{Said}(p, e)$ and $\mathbf{true}$.

EXAMPLE 7.1. Suppose that a company allows employees to access their server and allows nonemployees access if they sign a nondisclosure agreement. If Alice cannot prove that she is an employee, can she still get access to the server by signing a nondisclosure agreement? Intuitively, she should be able to, because Alice is either an employee, in which case she has permission, or she is not an employee, in which case she still has permission because she signed the waiver. However, if we express the query in the obvious way (using negated predicates), then Alice is not permitted, because

$$\begin{aligned}&\mathbf{SignedWaiver}(\mathbf{Alice}) \land \forall x(\mathbf{Employee}(x) \Rightarrow \mathbf{Perm}(x, \mathbf{access}, \mathbf{server}))\land\\ &\forall x(\mathbf{NotEmployee}(x) \land \mathbf{SignedWaiver}(x) \Rightarrow \mathbf{Perm}(x, \mathbf{access}, \mathbf{server})) \Rightarrow\\ &\mathbf{Perm}(\mathbf{Alice}, \mathbf{access}, \mathbf{server})\end{aligned}$$

is not valid. □

To address the unintuitive behavior shown in Example 7.1, we could replace the negated predicates by a negation operator, which is the standard approach in logic. Let XrML$^{\neg}$ be XrML extended so that the set of conditions includes $\neg\mathbf{Said}(p, e)$ as well as $\mathbf{Said}(p, e)$, and the set of conclusions includes $\neg\mathbf{Pr}(p)$ and $\neg\mathbf{Perm}(p, r, s)$, as well as $\mathbf{Pr}(p)$ and $\mathbf{Perm}(p, r, s)$. There is no problem extending the semantics of XrML to XrML$^{\neg}$. Moreover, by replacing **NotEmployee** in Example 7.1

[1] Since XrML allows the application to define only additional principals, rights, resources, and conditions, we cannot add **Prohibited** to XrML without extending the framework, but the extension is so minor that we ignore it here; moreover, there are no implications as far as complexity goes.

by $\neg$**Employee**, we get the intuitively correct answer. The downside of allowing negation is intractability. Recall that $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$ is a small fragment of XrML: the licenses in this fragment do not mention the $\cup$ operator, every variable in the antecedent of a grant appears in its conclusion, and the execution tree for all calls to **Holds2** has height at most two. Theorem 5.2 shows that queries in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$ are tractable; however, as we now show, adding negation to this relatively small language makes it intractable.

THEOREM 7.2. *Let $(e, L, R, E)$ be a tuple in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$ extended to include negated* **Said** *conditions and negated conclusions. The problem of deciding whether*

$$\bigwedge_{\ell \in L} \ell^{L,A,S,E} \wedge \bigwedge_{g \in R} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$$

*is valid is NP-hard. This result holds even if $e$, all of the licenses in $L$, and all of the conclusions in $E$ are in XrML, all but one of the grants in $R$ is in XrML, and the one grant that is in XrML$^\neg$ – XrML is of the form $\forall x_1 \ldots \forall x_n (\neg e)$.*

We are currently investigating whether there is a tractable fragment of XrML$^\neg$ that is sufficiently expressive to capture the grants and licenses that are of practical importance. We expect that some ideas from our work on Lithium [Halpern and Weissman 2003] will prove useful in this regard.

## 8. MPEG-21 REL

MPEG-21 is an international standard that is based on XrML. In [Halpern and Weissman 2004], we give semantics to a beta version of MPEG-21. All of the problems discussed in Section 3.2 are present in the beta version. We reported these issues to Xin Wang and Thomas DeMartini of the MPEG-21 working group before the final version was released, and our concerns were addressed in the final version (although not exactly as specified in Section 3.3).

The key differences between XrML and MPEG-21 are as follows.

—MPEG-21 consistently makes the subset assumption; a principal $\{p_1, \ldots, p_n\}$ has all of the properties and permissions of principal $p_i$, for $i = 1, \ldots, n$.

—A **Said** condition takes a *trustRoot* $s$ and a conclusion $e$. No definition of trustRoot is given in the specification; rather, it is assumed that the application will associate with every trustRoot $s$, set $L$ of licenses, and set $R$ of grants a set $G(s, L, R)$ of grants. **Said**$(s, e)$ holds if the set $L$ of issued licenses and $G(s, L, R)$ together imply $e$, where $R$ is the set of grants that implicitly hold.

—Rather than defining an algorithm, MPEG-21 says that $L$ and $R$ imply $e$ if there is a *proof tree* that shows the result holds. Roughly speaking, a proof tree $t$ shows that $L$ and $R$ imply $e$ if (a) $t$ includes a grant $g$ that implies $e$ if certain conditions hold; (b) for each of these conditions, $t$ includes a proof tree showing that the condition does, in fact, hold, and (c) either $g$ is in $R$ or, for some principal $p$, $(p, g)$ is in $L$ and $t$ includes a proof tree showing that $p$ is permitted to issue $g$.

We believe that the translation and corresponding proof of correctness given in Section 4.1 can be modified in a straightforward way to apply to MPEG-21. If this is indeed the case, then an appropriately modified **Query2** can be used to answer queries about licenses and grants that are written in MPEG-21.

## 9. CONCLUDING REMARKS

XrML is a popular language that does not have formal semantics. Since there are no formal semantics, we cannot argue that the XrML algorithm is incorrect, but its behavior on certain input does seem unreasonable. To address the problem, we modified the algorithm, provided formal semantics for an interesting fragment of XrML, and showed that the modified algorithm corresponds to our semantics in a precise sense.

We have examined only a fragment of XrML. A key reason for XrML's popularity is that the framework is extensible; applications can define new components (i.e., principals, rights, resources, and conditions) to suit their needs. We do not believe there will be be any difficulty in giving semantics to the extended language. The real question is whether we can find useful *tractable* extensions. As we have already seen, functions pose no semantic difficulties, but adding them makes the problem of answering queries in XrML undecidable. Another obvious and desirable feature is negation. Currently, XrML does not support negation in either the condition or conclusion of grants. This is a significant expressive weakness. Without negation, license writers cannot forbid an action explicitly nor can they say that a conclusion holds if a permission is denied or unregulated by a particular principal. While it is easy to extend XrML to include negation, doing so without placing further restrictions on the language makes it intractable. We suspect that we can use our earlier work [Halpern and Weissman 2003] to find a fragment of XrML with negation that is tractable and substantially more expressive.

Of course, it remains an open question whether XrML (or some extension of it) is the "best" policy language to use to for rights managment (and, more generally, trust management). Many languages have been proposed to do this, including XACML [Moses 2005], ODRL [Iannella 2001], numerous variants of Datalog [DeTreville 2002; Li et al. 2003; Li et al. 2002; Becker and Sewell 2004; Jim 2001], SPKI/SDSI [Halpern and van der Meyden 2003; Li and Mitchell 2006; Ellison et al. 1999b; 1999a], and our own language Lithium [Halpern and Weissman 2003]. As the references above indicate, a number of these have even been given semantics using first-order or modal logic. Comparing the strengths and weaknesses of all these approaches (and the semantic methods used to capture them) remains an open direction for future research.

Our work emphasizes the need for collaboration between language developers and the formal methods community. Our analysis of XrML demonstrates that a language without formal semantics is prone to ambiguities and inconsistencies, even if that language is carefully crafted and reviewed by industry. The good news is that collaborations are possible. The XrML developers that we contacted answered our questions and listened to our concerns. When they designed the next version of XrML, which is the ISO Standard MPEG-21 REL, they did not make the same mistakes.

### Acknowledgements

Many thanks to Xin Wang and Thomas DeMartini, who answered our questions about the intended meaning of various MPEG-21 components.

## A. PROOFS

PROPOSITION 3.7. *For all closed conditions $d$ and sets $L$ of licenses,*

*(a) every execution of **Holds**$(d, L)$ that terminates returns the same output,*

*(b) every execution of **Holds2**$(d, L, \emptyset)$ that terminates returns the same output,*

*(c) if an execution of **Holds**$(d, L)$ terminates by returning the truth value $t$, then an execution of **Holds2**$(d, L, \emptyset)$ terminates by returning $t$.*

PROOF. Parts (a) and (b) are immediate from the description of the **Holds** and **Holds2**. To prove part (c), say that a call tree for **Holds**$(d, L)$ is *non-repeating* if it is not the case that there exists a path $p$ in the call tree and two nodes $n_1$ and $n_2$ on the path such that both nodes are labeled by the same call to **Holds**. If **Holds**$(d, L)$ terminates, then it has a finite call tree. Moreover, it is easy to see that if there is a finite call tree for **Holds**$(d, L)$, then there is a nonrepeating call tree: If there is a call to **Holds**$(d', L')$ at two nodes on a path, we simply replace the subtree below the first call to **Holds**$(d', L')$ by the subtree below the last call to **Holds**$(d', L')$. A non-repeating call tree for **Holds**$(d, L)$ is essentially a call tree for **Holds2**$(d, L, \emptyset)$; the same calls are made at every step (the third component has to change appropriately). □

For the proofs of Proposition 3.8 and Lemma A.11, we rely on the observation that, if $T$ is the call tree for an execution of **Holds2**$(d, L, S)$, then $T$ can be viewed as an and-or tree, where a node labeled **Holds2**$(d', L, S')$ is an *and* node if $d'$ is a conjunction with at least two conjuncts, an *or* node if $d'$ is a **Said** condition and **Holds2**$(d', L, S')$ makes at least one recursive call, and a leaf if $d'$ is **true** or if $d'$ is a **Said** condition and **Holds2**$(d', L, S')$ makes no recursive calls. For future reference, note that each node in $T$ can be assigned a truth value in an obvious way. An *and* node is assigned "true" if all its children are; an *or* node is assigned "true" if at least one child is; a leaf labeled **Holds2**(**true**$, L, S')$ is assigned "true"; and a leaf labeled **Holds2**(**Said**$(p, e), L, S')$ is assigned "false".

PROPOSITION 3.8. *If $d$ is a closed condition, $L$ is a hierarchical set of licenses, $S$ is a set of closed* **Said** *conditions, and $T$ is the call tree of an execution of **Holds2**$(d, L, S)$, then the height of $T$ is at most $2\#(L) + 1$.*

PROOF. Because $L$ is hierarchical, there exists a strict partial order $\prec$ on licenses such that, if $\ell$ and $\ell'$ are licenses in $L$ and $\ell$ affects $\ell'$, then $\ell \prec \ell'$. A node $v$ in $T$ is a $non-and$ node if $v$ is an *or* node or a leaf. It follows from the description of **Holds2** that every *and* node has at least two children and every child of an *and* node is a $non-and$ node. So, if a path in $T$ from the root to a leaf has $n$ $non-and$ nodes, then that path has at most $2n$ total nodes; thus, it suffices to show that every path in $T$ has at most $\#(L)+1$ $non-and$ nodes. If $L = \emptyset$, then it is immediate from the description of **Holds2** that $T$ has height at most 1. Suppose that $L \neq \emptyset$. Then, for every path $t$ in $T$, either $t$ includes at most 2 $non-and$ nodes, in which case $t$ mentions at most $\#(L) + 1$ $non-and$ nodes, or $t$ includes 2 $non-and$ nodes $v_i$ and $v_j$ such that an *or* node precedes $v_i$, which precedes $v_j$, and no *or* node is between $v_i$ and $v_j$. If $v_i$ has a label of the form **Holds2**$(d_i, L, S_i)$ and $v_j$ has a label of the form **Holds2**$(d_j, L, S_j)$, then it follows from the description of **Holds2** that there are licenses $(p_i, g_i)$ and $(p_j, g_j)$ in $L$ and closed substitutions

$\sigma_i$ and $\sigma_j$ such that the antecedent of $g_i$ under $\sigma_i$ mentions $d_i$; the antecedent of $g_j$ under $\sigma_j$ mentions $d_j$; and $(p_j, g_j)$ affects $(p_i, g_i)$. Thus, $(p_j, g_j) \prec (p_i, g_i)$. It follows that $t$ has at most $\#(L) + 1$ $non - and$ nodes. □

Definition A.1. Suppose that $(e, L, R, E)$ is a query, $X$ is an execution of **Query2**$(e, L, R, E)$, and $A = A(e, L, R, E, X)$. Define

$$\begin{array}{lcl} \mathcal{E}^*(e, L, R) & = & \{\mathbf{Perm}(p, \mathtt{issue}, g) \mid (p, g) \in L\} \cup \{e\}. \\ \mathcal{S}^*(e, L, R, E, X) & = & \{\mathbf{Said}(p, \mathbf{Pr}(p')) \mid p, p' \in P \text{ and } \mathbf{Pr} \in \mathit{primitiveProp}\} \cup \\ & & \{\mathbf{Said}(p, \mathbf{Perm}(p', \mathtt{issue}, g)) \mid p, p' \in P \text{ and } g \in A\}. \end{array}$$

□

Theorem 4.1. *For all strings $s$ in the language and all finite sets $L$ of licenses, $A$ of closed resources, $S$ of closed* **Said** *conditions, and $E$ of closed conclusions, $s^{L,A,S,E}$ is well defined.*

Proof. Let $S_L$ be the set of **Said** conditions that are mentioned in issued grants; that is, $\mathbf{Said}(p, e) \in S_L$ iff there is a license $(p', g) \in L$ such that $g$ mentions $\mathbf{Said}(p, e)$. Let $S_s$ be the set of **Said** conditions mentioned in $s$. Finally, let $S_{L,s} = S_L \cup S_s$. We define a lexicographic order on the tuples $(s, S)$ such that $(s, S) < (s', S')$ iff either (a) $\#(S_{L,s} - S) < \#(S_{L,s} - S')$ or (b) $\#(S_{L,s} - S) = \#(S_{L,s} - S')$ and $|s| < |s'|$. The proof is by induction on this ordering. If $\#(S_{L,s} - S) = 0$ and $|s| = 1$, then $s^{L,A,S,E} = s$, so the translation is well defined. The inductive step is trivial except when $s = \mathbf{Said}(p, e)$ and $s \notin S$.

Suppose that $s$ is of the form $\mathbf{Said}(p, e)$ and $s \notin S$. Recall that

$$\mathbf{Said}(p, e)^{L,A,S,E} = \mathsf{Val}(\bigwedge_{g \in R_p} g^{L,A,S',\emptyset} \Rightarrow e^{L,A,S',\emptyset}),$$

where $R_p = \{g \mid (p', g) \in L \text{ for a } p' \in p\}$ and $S' = S \cup \{\mathbf{Said}(p, e)\}$. Because $L$ is a finite set, $R_p$ is a finite set and because $e$ is a conclusion, $e^{L,A,S',\emptyset}$ is well defined. So, to prove that $\mathbf{Said}(p, e)^{L,A,S,E}$ is well defined, it suffices to show that $g^{L,A,S',\emptyset}$ is well defined for all $g \in R_p$. Suppose that $s \notin S_L$. Then $\#(S_{L,g} - S') = \#(S_L - S')$ since $S_{L,g} = S_L$; $\#(S_L - S') = \#(S_L - S)$ since $s \notin S_L$; $\#(S_L - S) < \#(S_L - S \cup \{s\})$ since $s \notin S_L$; and $\#(S_L - S \cup \{s\}) = \#(S_{L,s} - S)$ since $s \notin S$. So, putting the pieces together, $\#(S_{L,g} - S') < \#(S_{L,s} - S)$ and, by the induction hypothesis, $g^{L,A,S',\emptyset}$ is well defined. Suppose that $s \in S_L$. Then $\#(S_{L,g} - S') = \#(S_L - S')$ since $S_{L,g} = S_L$; $\#(S_L - S') < \#(S_L - S)$ since $s \in S_L - S$; and $\#(S_L - S) = \#(S_{L,s} - S)$ since $s \in L$. Again, putting the pieces together, $\#(S_{L,g} - S') < \#(S_{L,s} - S)$, so $g^{L,A,S',\emptyset}$ is well defined by the induction hypothesis. □

We next prove Theorem 4.2. We actually prove a stronger result, given as Theorem A.9; Theorem A.9(c) is Theorem 4.2. The next five lemmas provide a deeper understanding of the properties of the **Query2**, **Auth2**, and **Holds2** algorithms and the translation, and are used in the proof of Theorem A.9.

Lemma A.2. *Suppose that $(e, L, R, E)$ is a query. Then during an execution $X$ of **Query2**$(e, L, R, E)$*

*(a) every call made to **Query2**, **Auth2**, and **Holds2** takes $L$ as its second argument;*

*(b) every call made to* ***Query2*** *and* ***Auth2*** *takes $R$ as its third argument;*

*(c) if* ***Query2***$(e', L, R, E')$ *is called, then* $e' \in \mathcal{E}^*(e, L, R)$*;*

*(d) if* ***Auth2***$(e', L, R, E')$ *is called, then* $e' \in \mathcal{E}^*(e, L, R)$*; and*

*(e) if* ***Holds***$(d, L, S)$ *is called, then every conjunct of $d$ is in* $\mathcal{S}^*(e, L, R, E, X) \cup \{\mathbf{true}\}$.

Proof. Parts (a) through (d) follow immediately from the descriptions of **Query2**, **Auth2**, and **Holds2**. For part (e), suppose that $\mathbf{Holds}(d, L, S)$ is called. Because $d$ is a closed condition, every conjunct of $d$ is either **true** or of the form $\mathbf{Said}(p, e')$, where $p$ is a closed principal and $e'$ is a closed conclusion. If $e'$ is of the form $\mathbf{Pr}(p')$, then $\mathbf{Said}(p, e')$ is clearly in $\mathcal{S}^*(e, L, R, E, X)$. Otherwise, $e'$ is of the form $\mathbf{Perm}(p', \mathtt{issue}, g)$. Because $e'$ is an input to a call made during $X$ and $g$ is mentioned in $e'$, $g \in A(e, L, R, E, X)$. $\square$

Lemma A.3. *Suppose that $(e, L, R, E)$ is a query such that $e \in E$, $A$ is a set of closed resources, and $S$ is a set of closed* **Said** *conditions. Then $\bigwedge_{\ell \in L} \ell^{L,A,S,E} \wedge \bigwedge_{g \in R} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is not acceptably valid (and hence not valid).*

Proof. Let $m$ be an acceptable model that satisfies $e'^{L,A,S,E}$ iff $e' \neq e$. Recall that, for a grant $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$, $g^{L,A,S,E}$ is a conjunction of formulas of the form

$$(\bigwedge_{e \in E} \neg \mathsf{Val}(e^{L,A,S,E} \Leftrightarrow (e_g\sigma)^{L,A,S,E}) \wedge (d_g\sigma)^{L,A,S,E}) \Rightarrow (e_g\sigma)^{L,A,S,E},$$

where $\sigma$ is a closed substitution. If $e \in E$, then $m$ satisfies $g^{L,A,S,E}$ because, for all substitutions $\sigma$, either $(e_g\sigma)^{L,A,S,E} \neq e^{L,A,S,E}$, in which case $m$ satisfies $(e_g\sigma)^{L,A,S,E}$, or $(e_g\sigma)^{L,A,S,E} = e^{L,A,S,E}$, in which case $\bigwedge_{e \in E} \neg \mathsf{Val}(e^{L,A,S,E} \Leftrightarrow (e_g\sigma)^{L,A,S,E})$ is equivalent to **false**. Since $m$ satisfies every grant, $m$ satisfies $\bigwedge_{\ell \in L} \ell^{L,A,S,E} \wedge \bigwedge_{g \in R} g^{L,A,S,E}$. By construction, $m$ does not satisfy $e^{L,A,S,E}$, so $m$ does not satisfy $\bigwedge_{\ell \in L} \ell^{L,A,S,E} \wedge \bigwedge_{g \in R} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$. $\square$

Lemma A.4. *Suppose that $(e, L, R, E)$ is a query, $A$ is a set of closed resources, and $S$ is a set of closed* **Said** *conditions. Then (a) $e'^{L,A,S,E} = e'^{L,A,S,(E \cup \{e\})}$ for every closed conclusion $e'$ in the language, (b) $g^{L,A,S,E} \Rightarrow g^{L,A,S,(E \cup \{e\})}$ is valid for every grant $g$ in the language, and (c) $\ell^{L,A,S,E} \Rightarrow \ell^{L,A,S,(E \cup \{e\})}$ is valid for every license $\ell$ in the language.*

Proof. Part (a) follows immediately from the translation.

For part (b), let $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$. It is easy to see that $g^{L,A,S,E} \Rightarrow g^{L,A,S,(E \cup \{e\})}$ is valid if, for all closed substitutions $\sigma$, $d_g\sigma^{L,A,S,(E \cup \{e\})} \Rightarrow d_g\sigma^{L,A,S,E}$ is valid. The latter statement holds because the translation of a condition does not depend on the final input argument (i.e., the set of conditions), so $d_g\sigma^{L,A,S,(E \cup \{e\})} = d_g\sigma^{L,A,S,E}$.

For part (c), let $\ell = (p, h)$. If $\mathbf{Perm}(p, \mathtt{issue}, h) \in E \cup \{e\}$ or $(p, h) \notin L$, then $\ell^{L,A,S,(E \cup \{e\})} = \mathbf{true}$, so $\ell^{L,A,S,E} \Rightarrow \ell^{L,A,S,(E \cup \{e\})}$ is valid. If $\mathbf{Perm}(p, \mathtt{issue}, h) \notin E \cup \{e\}$ and $(p, h) \in L$, then $\ell^{L,A,S,E} = \mathbf{Perm}(p, \mathtt{issue}, c_h) \Rightarrow h^{L,A,S,E}$ and $\ell^{L,A,S,(E \cup \{e\})} = \mathbf{Perm}(p, \mathtt{issue}, c_h) \Rightarrow h^{L,A,S,(E \cup \{e\})}$. It follows that $\ell^{L,A,S,E} \Rightarrow \ell^{L,A,S,(E \cup \{e\})}$ is valid if $h^{L,A,S,E} \Rightarrow h^{L,A,S,(E \cup \{e\})}$ is valid. The latter formula is valid by part (b). $\square$

Definition A.5. For a set $A$ of closed resources, an $A$-*closed* substitution $\sigma$ is a closed substitution such that, for all variables $x$ of sort *Resource*, $\sigma(x) \in A$. $\square$

Lemma A.6. *Suppose that $G$ is a set of grants, $L$ is a set of licenses, $A$ is a set of closed resources, $S$ is a set of closed* **Said** *conditions, $E$ is a set of grants, and $e$ is a closed conclusion. Then $\bigwedge_{g\in G} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid iff $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid for some $g \in G$. Moreover, for any grant $g$, $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid iff $e \notin E$ and, for some $A$-closed substitution $\sigma$, the formula $d_g\sigma^{L,A,S,E}$ is acceptably valid and $e_g\sigma = e$.*

Proof. We first show that $\bigwedge_{g\in G} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid iff $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid for some $g \in G$. The "if" direction is trivial. For the "only if" direction, suppose by way of contradiction that $\bigwedge_{g\in G} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid and $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is not acceptably valid for all $g \in G$. Let $m$ be an acceptable model such that, for all closed conclusions $e'$, $m$ satisfies $e'^{L,A,S,E}$ iff $e' \neq e$. Since $\bigwedge_{g\in G} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid, there is a $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in G$ such that $m$ does not satisfy $g^{L,A,S,E}$. By the translation, it follows that there is an $A$-closed substitution $\sigma$ such that $e_g\sigma \notin E$, $d_g\sigma^{L,A,S,E}$ holds in $m$, and $e_g\sigma \neq e$. Because, for all conditions $d'$, $d'^{L,A,S,E}$ can be written as $\mathsf{Val}(\varphi)$ for an appropriate formula $\varphi$, $d_g\sigma^{L,A,S,E}$ is acceptably valid since it holds in an acceptable model. It follows that $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid, which contradicts the assumption.

It remains to show that $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid for a grant $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g)$ iff $e \notin E$ and, for some $A$-closed substitution $\sigma$, the formula $d_g\sigma^{L,A,S,E}$ is acceptably valid and $e_g\sigma = e$. The "if" direction is immediate from the translation. For the "only if" direction, suppose by way of contradiction that $g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$ is acceptably valid and either $e \in E$ or, for each $A$-closed substitution $\sigma$, either $d_g\sigma^{L,A,S,E}$ is not valid or $e_g\sigma \neq e$. Let $m$ be the acceptable model defined above; that is, for all conclusions $e'$, $m$ satisfies $e'^{L,A,S,E}$ iff $e' \neq e$. We can get a contradiction by showing that $m$ satisfies $g^{L,A,S,E}$. If $e \in E$, then $m$ satisfies $g^{L,A,S,E}$ since either $e_g\sigma \in E$ (because $e_g\sigma = e$), or $e_g\sigma$ holds in $m$ (because $e_g\sigma \neq e$). Otherwise, by assumption, either $d_g\sigma^{L,A,S,E}$ is not acceptably valid or $e_g\sigma \neq e$, for each $A$-closed substitution $\sigma$. Note that, because $d_g\sigma^{L,A,S,E}$ (like every formula of the form $d^{L,A,S,E}$ for some condition $d$) is equivalent to a formula of the form $\mathsf{Val}(\varphi)$, then if it is not acceptably valid, it is not true in any acceptable model and, in particular, not in $m$. It then easily follows from the translation that $m$ satisfies $g^{L,A,S,E}$. This gives us the desired contradiction. $\square$

Definition A.7. Let $(e, L, R, E)$ be a query, let $X$ be a terminating execution of **Query2**$(e, L, R, E)$, and let $A = A(e, L, R, E, X)$. Then

$$\begin{aligned} G(e, L, R, E, X) \ = \ & R \cup \{h \mid \text{ for some principal } p,\ (p, h) \in L \text{ and} \\ & ((\textstyle\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,(E\cup\{e\})}) \wedge (\textstyle\bigwedge_{g\in R} g^{L,A,\emptyset,(E\cup\{e\})})) \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_h) \\ & \text{is acceptably valid}\}. \end{aligned}$$

$\square$

Lemma A.8. *Suppose that $(e, L, R, E)$ is a query, $X$ is a terminating execution of* ***Query2***$(e, L, R, E)$*, and $A = A(e, L, R, E, X)$. Then $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E} \Rightarrow$*

*$e^{L,A,\emptyset,E}$ is acceptably valid iff there is a grant $h \in G(e,L,R,E,X)$ such that $h^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid.*

Proof. For the "if" direction, suppose that $h$ is a grant in $G(e,L,R,E,X)$ such that $h^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid. If $h \in R$, then $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid. If $h \in G(e,L,R,E,X) - R$, then there is a principal $p$ such that

(1a) $(p,h) \in L$,

(1b) $\mathbf{Perm}(p,\mathtt{issue},h) \notin E$, and

(1c) $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,(E\cup\{e\})} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p,\mathtt{issue},c_h)$ is acceptably valid.

Let $\varphi = \bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E}$. It follows from (1a) that $\varphi \Rightarrow (p,h)^{L,A,\emptyset,E}$ is acceptably valid. It follows from (1a), (1b), and the translation that $\varphi \Rightarrow (\mathbf{Perm}(p,\mathtt{issue},c_h) \Rightarrow h^{L,A,\emptyset,E})$ is acceptably valid. It follows from Lemma A.4 and (1c) that $\varphi \Rightarrow \mathbf{Perm}(p,\mathtt{issue},c_h)$ is acceptably valid, so $\varphi \Rightarrow h^{L,A,\emptyset,E}$ is acceptably valid. By assumption $h^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid, so $\varphi \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid.

For the "only if" direction, suppose that there is no grant $g \in G(e,L,R,E,X)$ such that $g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid. Let $m$ be an acceptable model that does not satisfy $e^{L,A,\emptyset,E}$ and the formulas in $\{\mathbf{Perm}(p,\mathtt{issue},h)^{L,A,\emptyset,E} \mid (p,h) \in L, \mathbf{Perm}(p,\mathtt{issue},h) \notin E, \text{ and } h \notin G(e,L,R,E,X)\}$. Because $m$ does not satisfy $e^{L,A,\emptyset,E}$, it suffices to show that $m$ satisfies $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E}$. We do this by showing that (1) $m$ satisfies $(p,h)^{L,A,\emptyset,E}$ for every license $(p,h)$ such that $h \notin G(e,L,R,E,X)$, and (2) $m$ satisfies $g^{L,A,\emptyset,E}$ for every grant $g \in G(e,L,R,E,X)$.

For part (1), observe that if $\mathbf{Perm}(p,\mathtt{issue},h) \in E$ or $(p,h) \notin L$, then $(p,h)^{L,A,\emptyset,E} = \mathbf{true}$, so $(p,h)^{L,A,\emptyset,E}$ holds in $m$. If $\mathbf{Perm}(p,\mathtt{issue},h) \notin E$ and $(p,h) \in L$, then $(p,h)^{L,A,\emptyset,E} = \mathbf{Perm}(p,\mathtt{issue},c_h) \Rightarrow h^{L,A,\emptyset,E}$ and, by construction, $m$ does not satisfy $\mathbf{Perm}(p,\mathtt{issue},c_h)$; so $(p,h)^{L,A,\emptyset,E}$ is again true in $m$.

For part (2), let $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in G(e,L,R,E,X)$, and recall that $g^{L,A,\emptyset,E}$ is the conjunction of formulas of the form

$$(\bigwedge_{e\in E} \neg\mathsf{Val}(e^{L,A,\emptyset,E} \Leftrightarrow (e_g\sigma)^{L,A,\emptyset,E}) \wedge (d_g\sigma)^{L,A,\emptyset,E}) \Rightarrow (e_g\sigma)^{L,A,\emptyset,E},$$

where $\sigma$ is an $A$-closed substitution. Clearly, $m$ satisfies $g^{L,A,\emptyset,E}$ iff, for every $A$-closed substitution $\sigma$, $m$ satisfies $((\bigwedge_{e'\in E} \neg\mathsf{Val}(e'^{L,A,\emptyset,E} \Leftrightarrow (e_g\sigma)^{L,A,\emptyset,E}) \wedge (d_g\sigma)^{L,A,\emptyset,E}) \Rightarrow (e_g\sigma)^{L,A,\emptyset,E}$. It is easy to see that the latter statement holds if, for all $A$-closed substitutions $\sigma$, either $e_g\sigma \in E$, $(d_g\sigma)^{L,A,\emptyset,E}$ is not true in $m$, or $(e_g\sigma)^{L,A,\emptyset,E}$ is true in $m$. We claim that this is indeed the case. To prove the claim, suppose by way of contradiction that $e_g\sigma \notin E$, $(d_g\sigma)^{L,A,\emptyset,E}$ is true in $m$, and $(e_g\sigma)^{L,A,\emptyset,E}$ is not true in $m$. Since $(e_g\sigma)^{L,A,\emptyset,E}$ is not true in $m$, either $e_g\sigma = e$ or $e_g\sigma \in \{\mathbf{Perm}(p,\mathtt{issue},h) \mid (p,h) \in L, \mathbf{Perm}(p,\mathtt{issue},h) \notin E, \text{ and } h \notin G(e,L,R,E,X)\}$.

If $e_g\sigma = e$, then we claim that $g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid. To see this note that $g^{L,A,\emptyset,E} \Rightarrow (\bigwedge_{e'\in E} \neg\mathsf{Val}(e'^{L,A,\emptyset,E} \Leftrightarrow (e_g)\sigma^{L,A,\emptyset,E}) \wedge (d_g\sigma)^{L,A,\emptyset,E} \Rightarrow$

$(e_g\sigma)^{L,A,\emptyset,E})$ is acceptably valid. Since $e_g\sigma \notin E$, the formula $\bigwedge_{e'\in E} \neg\mathsf{Val}(e'^{L,A,\emptyset,E} \Leftrightarrow (e_g\sigma)^{L,A,\emptyset,E}) = \mathbf{true}$; so, $g^{L,A,\emptyset,E} \Rightarrow ((d_g\sigma)^{L,A,\emptyset,E} \Rightarrow (e_g\sigma)^{L,A,\emptyset,E})$ is acceptably valid. Since $(d_g\sigma)^{L,A,\emptyset,E}$ is true in $m$ by assumption, and, as we have observed, every formula of the form $d^{L,A,\emptyset,E}$ is equivalent to $\mathsf{Val}(\varphi)$ for some formula $\varphi$, $(d_g\sigma)^{L,A,\emptyset,E}$ is acceptably valid and, as a result, $g^{L,A,\emptyset,E} \Rightarrow (e_g\sigma)^{L,A,\emptyset,E}$ is acceptably valid. By assumption, $e_g\sigma = e$, so $g^{L,A,\emptyset,E} \Rightarrow e^{L,A,\emptyset,E}$ is acceptably valid. Since $g \in G(e,L,R,E,X)$ and, by assumption, none of the grants in $G(e,L,R,E,X)$ imply $e^{L,A,\emptyset,E}$, we have a contradiction.

Finally, suppose that $e_g\sigma \neq e$ and $e_g\sigma = \mathbf{Perm}(p,\mathtt{issue},h)$, where $(p,h) \in L$, $\mathbf{Perm}(p,\mathtt{issue},h) \notin E$, and $h \notin G(e,L,R,E,X)$. We now prove that $g^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p,\mathtt{issue},c_h)$ is acceptably valid, so $h \in G(e,L,R,E,X)$, which contradicts the assumptions. We begin by noting that $g^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow ((\bigwedge_{e'\in E\cup\{e\}} \neg\mathsf{Val}(e'^{L,A,\emptyset,(E\cup\{e\})} \Leftrightarrow (e_g\sigma)^{L,A,\emptyset,(E\cup\{e\})}) \wedge (d_g\sigma)^{L,A,\emptyset,(E\cup\{e\})}) \Rightarrow (e_g\sigma)^{L,A,\emptyset,(E\cup\{e\})})$ is acceptably valid. By assumption, $e_g\sigma \notin E \cup \{e\}$, so $g^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow ((d_g\sigma)^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow (e_g\sigma)^{L,A,\emptyset,(E\cup\{e\})})$ is acceptably valid. Since $e_g\sigma = \mathbf{Perm}(p,\mathtt{issue},h)$, $e_g\sigma^{L,A,\emptyset,(E\cup\{e\})} = \mathbf{Perm}(p,\mathtt{issue},c_h)$, so $g^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow ((d_g\sigma)^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p,\mathtt{issue},c_h))$ is acceptably valid. It remains to be shown that $(d_g\sigma)^{L,A,\emptyset,(E\cup\{e\})}$ is acceptably valid. Because the translation of a condition does not depend on the set of conclusions, it suffices to show that $d_g\sigma^{L,A,\emptyset,E}$ is acceptably valid. But, as we observed above, this follows immediately from the assumption that $d_g\sigma^{L,A,\emptyset,E}$ is true in $m$.

□

THEOREM A.9. *Suppose that $(e,L,R,E)$ is a query, $X$ is a terminating execution of* ***Query2***$(e,L,R,E)$*, and $A = A(e,L,R,E,X)$. Then for all calls of the form* ***Holds2***$(d,L,S)$*,* ***Auth2***$(e',L,R,E')$*, or* ***Query2***$(e',L,R,E')$ *made during execution $X$, including the initial call,*

(a) ***Holds2***$(d,L,S)$ *returns* **true** *iff $d^{L,A,S,E'}$ is acceptably valid, where $E'$ is an (arbitrary) set of closed conclusions;*

(b) ***Auth2***$(e',L,R,E')$ *returns the set $D$ of closed conditions, where $D = \{d \mid e' \notin E'$ and, for some grant $\forall x_1 \ldots \forall x_n(d_g \to e_g) \in G(e',L,R,E',X)$ and closed substitution $\sigma$, $d_g\sigma = d$ and $e_g\sigma = e'\}$; and*

(c) ***Query2***$(e',L,R,E')$ *returns* **true** *iff $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E'} \wedge \bigwedge_{g\in R} \ell^{L,A,\emptyset,E'} \Rightarrow e'^{L,A,\emptyset,E'}$ is acceptably valid.*

PROOF. We prove part (a) by induction on $\#(\mathcal{S}^*(e,L,R,E,X) - S)$, with a subinduction on the structure of $d$. Suppose that $\#(\mathcal{S}^*(e,L,R,E,X) - S) = 0$. If $d = \mathbf{true}$, then $\mathbf{Holds2}(d,L,S) = \mathbf{true}$ and $d^{L,A,S,E'} = \mathbf{true}$. Suppose that $d$ is of the form $\mathbf{Said}(p,e')$. Then, by Lemma A.2, $d \in \mathcal{S}^*(e,L,R,E)$. By assumption, $\#(\mathcal{S}^*(e,L,R,E) - S) = 0$, so $d \in S$. It follows that $\mathbf{Holds2}(d,L,S) = \mathbf{false}$ and $d^{L,A,S,E'} = \mathbf{false}$. Finally, if $d$ is a conjunction, then the result is immediate from the induction hypothesis. For the induction step, the argument used for the base case applies if $d = \mathbf{true}$ or if $d$ is a conjunction of conditions. Suppose that $d$ has the form $\mathbf{Said}(p,e')$. If $d \in S$, then $\mathbf{Holds2}(d,L,S) = \mathbf{false}$ and $d^{L,A,S,E'} = \mathbf{false}$. If $d \notin S$ then, by the description of $\mathbf{Holds2}$, $\mathbf{Holds2}(d,L,S) = \mathbf{true}$ iff there is a grant $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in R_p$ and an $A$-closed substitution $\sigma$ such that $\mathbf{Holds2}(d_g\sigma, L, S \cup \{d\}) = \mathbf{true}$ and $e_g\sigma = e'$. By the induction hypothesis, $\mathbf{Holds2}(d_g\sigma, L, S \cup \{d\}) = \mathbf{true}$ iff $d_g\sigma^{L,A,(S\cup\{d\}),E'}$ is acceptably valid. By the translation, the latter statement holds iff $d_g\sigma^{L,A,(S\cup\{d\}),\emptyset}$ is acceptably valid. So, by Lemma A.6, $\mathbf{Holds2}(d, L, S) = \mathbf{true}$ iff $(\bigwedge_{g\in R_p} g^{L,A,(S\cup\{d\}),\emptyset}) \Rightarrow e^{L,A,(S\cup\{d\}),\emptyset}$ is acceptably valid. It is immediate from the translation that the latter statement holds iff $d^{L,A,S,E'}$ is acceptably valid.

We prove parts (b) and (c) by simultaneous induction on $\#(\mathcal{E}^*(e, L, R) - E')$. If $\#(\mathcal{E}^*(e, L, R) - E') = 0$, then $e' \in \mathcal{E}^*(e, L, R)$ by Lemma A.2, so $e' \in E'$. Because $e' \in E'$, $\mathbf{Auth2}(e', L, R, E') = \emptyset$, so part (b) holds. For part (c), **Query2** begins by calling $\mathbf{Auth2}(e', L, R, E')$, which returns the empty set, and then **Query2** returns **false**. Since $e' \in E'$, it follows from Lemma A.3 that $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E'} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E'} \Rightarrow e'^{L,A,\emptyset,E'}$ is not acceptably valid, so the invariant holds.

Now consider the inductive step. For part (b), suppose that $\mathbf{Auth2}(e', L, R, E')$ is called during the execution of $\mathbf{Query2}(e, L, R, E)$. If $e' \in E'$, then part (b) holds by the same argument as in the base case. If $e' \notin E'$, then **Auth2** returns a set $D$ of closed conditions such that $d \in D$ iff there is a grant $\forall x_1 \ldots \forall x_n(d_h \rightarrow e_h) \in S_L$ and a closed substitution $\sigma$ such that $d_h\sigma = d$ and $e_h\sigma = e$, where

$$\begin{aligned} S_L = \ & R \cup \{h \mid \text{for some principal } p,\ (p, h) \in L \text{ and, during execution } X, \\ & \mathbf{Query2}(\mathbf{Perm}(p, \mathtt{issue}, h), L, R, (E' \cup \{e'\})) \text{ returns } \mathbf{true}\}. \end{aligned}$$

It clearly suffices to show that $S_L = G(e', L, R, E', X)$. By Lemma A.2, $e' \in \mathcal{E}^*(e, L, R)$ and, by assumption, $e \notin E'$. So it follows from the induction hypothesis that

$$\begin{aligned} S_L = \ & R \cup \{h \mid \text{for some principal } p,\ (p, h) \in L \text{ and} \\ & \textstyle\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,(E'\cup\{e'\})} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,(E'\cup\{e'\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_h) \text{ is} \\ & \text{acceptably valid}\}, \end{aligned}$$

which is $G(e', L, R, E', X)$.

For part (c), observe that if $e' \in E'$ then we can use the same reasoning as in the base case to show that the invariant holds. If $e' \notin E'$ then, during execution $X$, $\mathbf{Query2}(e', L, R, E')$ returns **true** iff there is a closed condition $d$ in the output of $\mathbf{Auth2}(e', L, R, E')$ such that **Query2** calls $\mathbf{Holds2}(d, L, \emptyset)$, which returns **true**. By part (b), $\mathbf{Auth2}(e', L, R, E')$ returns a set of conditions that includes $d$ iff there is a grant $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in G(e', L, R, E', X)$ and a closed substitution $\sigma$ such that $d_g\sigma = d$ and $e_g\sigma = e'$. Moreover, since $\mathbf{Holds2}(d, L, \emptyset)$ is called during execution $X$ of $\mathbf{Query2}(e, L, R, E)$, $\sigma$ is $A$-closed. By part (a), $\mathbf{Holds2}(d, L, \emptyset) = \mathbf{true}$ iff $d^{L,A,\emptyset,E'}$ is acceptably valid. So $\mathbf{Query2}(e', L, R, E')$ returns **true** iff there is a grant $g = \forall x_1 \ldots \forall x_n(d_g \rightarrow e_g) \in G(e', L, R, E', X)$ and an $A$-closed substitution $\sigma$ such that $d^{L,A,\emptyset,E'}$ is acceptably valid and $e_g\sigma = e'$. By assumption, $e' \notin E'$; so, by Lemma A.6, $\mathbf{Query2}(e', L, R, E') = \mathbf{true}$ iff $g^{L,A,\emptyset,E'} \Rightarrow e'^{L,A,\emptyset,E'}$ is acceptably valid for some $g \in G(e', L, R, E', X)$. It follows from Lemma A.8 that the latter statement holds iff $\bigwedge_{\ell\in L} \ell^{L,A,\emptyset,E'} \wedge \bigwedge_{g\in R} g^{L,A,\emptyset,E'} \Rightarrow e'^{L,A,\emptyset,E'}$ is acceptably valid. □

THEOREM 5.1. *Determining whether some execution of* ***Query2***$(e, L, R, E)$ *returns* **true** *is undecidable for the set of queries* $(e, L, R, E)$ *such that at most one* *grant in $R \cup L$ is not restrained.*

Proof. We reduce the Post correspondence problem (PCP) [Post 1946] to the problem of determining whether some execution of $\mathbf{Query2}(e, L, R, \emptyset)$ returns **true** for a query $(e, L, R, \emptyset)$, where all but one grant in $R \cup L$ is restrained. Let $\Sigma$ be an alphabet; let $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ be strings over $\Sigma$; and, for all strings $s$ and $s'$, let $s \cdot s'$ be the concatenation of $s$ and $s'$. We want to determine if there are integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $s_{i_1} \cdot \ldots \cdot s_{i_k} = t_{i_1} \cdot \ldots \cdot t_{i_k}$.

To encode the problem as a query, assume that the language includes the primitive principal $p_\sigma$ for each symbol $\sigma \in \Sigma$, the primitive principal $p$, and the property **Pr**. For every string $s$ over $\Sigma$, define a function $G_s$ from grants to grants by induction on the length of $s$. If $s$ has length one ($s \in \Sigma$), then $G_s(g) = \mathbf{Perm}(p_s, \mathtt{issue}, g)$. If $s = \sigma s'$, then $G_s = G_\sigma \circ G_{s'}$. For all grants $g_1$ and $g_2$, define $G(g_1, g_2)$ to be the grant $\mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, g_1)) \rightarrow \mathbf{Perm}(p, \mathtt{issue}, g_2)$.

We claim that there are integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $s_{i_1} \cdot \ldots \cdot s_{i_k} = t_{i_1} \cdot \ldots \cdot t_{i_k}$ iff an execution of $\mathbf{Query2}(\mathbf{Pr}(p), L, R, \emptyset)$ returns **true**, where

$$\begin{aligned} L \;=\; & \{(p, \mathbf{Perm}(p, \mathtt{issue}, G(G_{s_i}(\mathbf{Pr}(p)), G_{t_i}(\mathbf{Pr}(p))))) \mid i = 1, \ldots, n\} \cup \\ & \{(p, \forall x_1 \forall x_2 (\mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, G(x_1, x_2))) \rightarrow \\ & \mathbf{Perm}(p, \mathtt{issue}, G(G_{s_i}(x_1), G_{t_i}(x_2))))) \mid i = 1, \ldots, n\} \end{aligned}$$

and $R = \{\forall x (\mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, G(x, x))) \rightarrow \mathbf{Pr}(p))\}$.

Recall that an execution of $\mathbf{Query2}(e, L, R, \emptyset)$ returns **true** iff an execution of $\mathbf{Auth2}(e, L, R, \emptyset)$ returns a set $D$ of conditions such that an execution of $\mathbf{Holds2}(d, L, \emptyset)$ returns **true** for some condition $d \in D$. It is easy to see that every execution of $\mathbf{Auth2}(e, L, R, \emptyset)$ returns the set $D = \{\mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, G(g, g))) \mid g \text{ is a closed grant}\}$. Moreover, if $d$ is of the form $\mathbf{Said}(p, \mathbf{Perm}(p, \mathtt{issue}, G(g, g)))$, where $g$ is a closed grant, then it is not hard to see that an execution of $\mathbf{Holds2}(d, L, \emptyset)$ returns **true** iff there are integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $g = G_{s_{i_1}}(G_{s_{i_2}}(\ldots G_{s_{i_k}}(\mathbf{Pr}(p)) \ldots))$ and $g = G_{t_{i_1}}(G_{t_{i_2}}(\ldots G_{t_{i_k}}(\mathbf{Pr}(p)) \ldots))$. The latter statements holds iff there are integers $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $s_{i_1} \cdot \ldots \cdot s_{i_k} = t_{i_1} \cdot \ldots \cdot t_{i_k}$. $\square$

Theorem 5.2. *The problem of deciding if some execution of* $\boldsymbol{Query2}(e, L, R, E)$ *returns* **true** *for* $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L} \cap \mathcal{L}'$ *is NP-hard for* $\mathcal{L}, \mathcal{L}' \in \{\mathcal{L}_1, \mathcal{L}_2^0, \mathcal{L}_3^2\}$.

Proof. For the NP hardness results, it suffices to show that the problem of deciding whether $\mathbf{Query2}(e, L, R, E) = \mathbf{true}$ is NP-hard if (a) $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$, (b) $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_3^2$, and (c) $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0$.

For part (a), we show that we can reduce the Hamiltonian path problem to the problem of determining whether $\mathbf{Query2}(e, L, R, E) = \mathbf{true}$, for some $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$. Given a graph $G(V, E)$, where $V = \{v_1, \ldots, v_n\}$, we take $v_1, \ldots, v_n$ to be primitive principles. We also assume that the language has primitive properties **Node**, **Edge**, and **Path**. For each node $v \in V$, let $g_v$ be the grant $\mathbf{Node}(v)$ (recall that this is an abbreviation for $\mathbf{true} \rightarrow \mathbf{Node}(v)$). For each edge $e = (v, v') \in E$, let $g_{(v,v')}$ be the grant $\mathbf{Edge}(\{v, v'\})$ (recall that $\{v, v'\}$ is an abbreviation for $\{v\} \cup \{v'\}$). Finally, let $g$ be the grant $\forall x_1 \ldots \forall x_n (d_1 \wedge d_2 \rightarrow \mathbf{Path}(\{x_1, \ldots, x_n\}))$, where

$$d_1 = \bigwedge_{1\le i\le n} \mathbf{Said}(\mathit{Alice}, \mathbf{Node}(x_i)) \text{ and}$$
$$d_2 = \bigwedge_{1\le i\le n-1} \mathbf{Said}(\mathit{Alice}, \mathbf{Edge}(\{x_i, x_{i+1}\})).$$

Let $L = \{(\mathit{Alice}, g_v) \mid v \in V\} \cup \{(\mathit{Alice}, g_e) \mid e \in E\}$ and let $R = \{g\}$. It is not hard to show that $\mathbf{Query2}(\mathbf{Path}(\{v_1,\ldots,v_n\}), L, R, \emptyset) = \mathbf{true}$ iff $G$ has a Hamiltonian path. To see this, observe that $\mathbf{Auth2}(\mathbf{Path}(\{v_1,\ldots,v_n\}), L, R, \emptyset)$ returns $\{d_1\sigma \wedge d_2\sigma \mid \sigma(x_i) = v_\pi(i), i = 1,\ldots,n$, where $\pi$ is some permutation of $\{1,\ldots,n\}\}$. The condition $d_2\sigma$ holds iff there is a path $x_1\sigma,\ldots,x_n\sigma$. Thus, $\mathbf{Query2}(\mathbf{Path}(\{v_1,\ldots,v_n\}), L, R, \emptyset) = \mathbf{true}$ iff there is a Hamiltonian path in $G$. Moreover, it is clear that $(\mathbf{Path}(\{v_1,\ldots,v_n\}), L, R, \emptyset) \in \mathcal{L}_0 \cap \mathcal{L}_2^0$ and it is not hard to see that $(\mathbf{Path}(\{v_1,\ldots,v_n\}), L, R, \emptyset) \in \mathcal{L}_3^2$, because the antecedent of every issued grant is **true**.

For part (b), we show that we can reduce the 3-satisfiability problem to the problem of determining whether $\mathbf{Query2}(e, L, R, E) = \mathbf{true}$, for $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_3^2$. Let $f = c_1 \wedge \ldots \wedge c_n$ be a formula in propositional logic, where each $c_i$ is a clause with three disjuncts. Let $q_1,\ldots,q_m$ be the primitive propositions mentioned in $f$. We want to determine if $f$ is satisfiable.

To encode the problem as an XrML query, suppose that $p_1,\ldots,p_n,p_t,p_f$ are distinct primitive principals, $\mathbf{Pr}$ is a property, and $x_1,\ldots,x_m$ are distinct variables of sort *Principal*. Let $g_0$ be a fixed closed grant. Given principals $t_1,\ldots,t_m$, we define grants $g_1(t_1),\ldots,g_m(t_1,\ldots,t_m)$ inductively as follows: $g_1(t_1)$ is the grant $\mathbf{true} \rightarrow \mathbf{Perm}(t_i, \mathtt{issue}, g_0)$ and, for $i = 2,\ldots,m$, $g_i$ is the grant $\mathbf{true} \rightarrow \mathbf{Perm}(t_i, \mathtt{issue}, g_{i-1}(t_1,\ldots,t_{i-1}))$. Let $e(t_1,\ldots,t_m)$ be the conclusion $\mathbf{Perm}(t_m, \mathtt{issue}, g_{m-1}(t_1,\ldots,t_{m-1}))$. For ease of exposition, let $e'$ be the conclusion $e(x_1,\ldots,x_m)$. Let $L = \{(p_i, \forall x_1\ldots\forall x_m(e'[x_j/p_t])) \mid q_j$ is a disjunct of $c_i\} \cup \{(p_i, \forall x_1\ldots\forall x_m(e'[x_j/p_f])) \mid \neg q_j$ is a disjunct of $c_i\}$ and let $R = \{\forall x_1\ldots\forall x_m((\bigwedge_{i=1,\ldots,n} \mathbf{Said}(p_i, e')) \rightarrow \mathbf{Pr}(p_t)\}$. We claim that $f$ is satisfiable iff $\mathbf{Query2}(\mathbf{Pr}(p_t), L, R, \emptyset) = \mathbf{true}$. Note that $(\mathbf{P}(p_t), L, R, \emptyset) \in \mathcal{L}_1 \cap \mathcal{L}_0 \cap \mathcal{L}_3^2$, since none of the grants mention a variable of sort *Resource*, the $\cup$ operator is not mentioned in the query, and the antecedent of every issued grant is **true**.

To prove the claim, first note that $\mathbf{Query2}(\mathbf{Pr}(p_t), L, R, \emptyset) = \mathbf{true}$ iff $\bigwedge_{i=1,\ldots,n} \mathbf{Said}(p_i, e')\sigma$ holds for some substitution $\sigma$. It is not hard to see that if $\sigma$ exists, then $f$ is satisfied by the truth assignment that sets $q_i = \mathbf{true}$ if $\sigma$ sets $x_i$ to $p_t$, and sets $q_i$ to **false** otherwise. Similarly, if $f$ is satisfied by a truth assignment $A$, then $\bigwedge_{i=1,\ldots,n} \mathbf{Said}(p_i, e')\sigma$ holds for the substitution $\sigma$ that replaces $x_i$ by $p_t$ if $A$ assigns $x_i$ to **true**, and replaces $x_i$ by $p_f$ otherwise.

For part (c), we show that we can reduce the 3-satisfiability problem to the problem of determining whether $\mathbf{Query2}(e, L, R, E) = \mathbf{true}$, for $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0$. As in part (b), let $f$ be the 3-CNF formula $c_1 \wedge \ldots \wedge c_n$, whose primitive propositions are $q_1,\ldots,q_m$. Define the condition $e(t_1,\ldots,t_m)$ as in part (b); again, take $e'$ to be an abbreviation for $e(x_1,\ldots,x_m)$. Let $p'_1,\ldots,p'_m$ be fresh principals, distinct from $p_1,\ldots,p_n,p_f,p_t$. We claim that $f$ is satisfied iff $\mathbf{Query2}(e(p'_1,\ldots,p'_m), L, R, \emptyset) = \mathbf{true}$, where

$$
\begin{aligned}
L \;=\; & \{(p_i, \forall x_1 \ldots \forall x_m(\mathbf{Said}(p_{i+1}, e'[x_j/p_t]) \rightarrow e'[x_j/p])) \mid \\
& \qquad q_j \text{ is a disjunct of } c_i,\ p \neq p_f, i = 1, \ldots, n-1\} \\
& \cup\{(p_i, \forall x_1 \ldots \forall x_m(\mathbf{Said}(p_{i+1}, e'[x_j/p_f]) \rightarrow e'[x_j/p])) \mid \\
& \quad \neg q_j \text{ is a disjunct of } c_i,\ p \neq p_t, i = 1, \ldots, n-1\} \\
& \cup\{(p_n, \forall x_1 \ldots \forall x_m(e'[x_j/p])) \mid \\
& \quad q_j \text{ is a disjunct of } c_n \text{ and } p \neq p_f, \text{ or } \neg q_j \text{ is a disjunct of } c_n \text{ and } p \neq p_t\} \\
R \;=\; & \{\mathbf{Said}(p_1, e(p'_1, \ldots, p'_m)) \rightarrow e(p'_1, \ldots, p'_m)\}.
\end{aligned}
$$

If $t_1, \ldots, t_m$ are variable-free principals, let $A(t_1, \ldots, t_m)$ be the set of all truth assignments to $q_1, \ldots, q_m$ such that $q_i$ is assigned **true** if $t_i = p_t$ and $q_i$ is assigned **false** if $t_i = p_f$, for $i = 1, \ldots, m$. (If $t_i \notin \{p_t, p_f\}$, then there are no constraints on $q_i$.) Let $A_i(t_1, \ldots, t_m)$ be the set of all truth assignments to $q_1, \ldots, q_m$ under which $c_i \wedge \ldots \wedge c_n$ is **true**. We show by induction on $n-i$ that $A_i(t_1, \ldots, t_m)$ is nonempty iff $\mathbf{Said}(p_i, e(t_1, \ldots, t_m))$ holds. If $n - i = 0$, then $i = n$. It is easy to see that $A_i(t_1, \ldots, t_m)$ is nonempty iff, for some $j = 1, \ldots, m$, either $q_j$ is a disjunct of $c_n$ and $t_j \neq p_f$, or $\neg q_j$ is a disjunct of $c_n$ and $t_j \neq p_t$. For the inductive step, suppose that $n - i > 0$. Clearly, $A_i(t_1, \ldots, t_m)$ is nonempty iff there is an assignment in $A_{i-1}(t_1, \ldots, t_m)$ under which $c_i$ is **true**. If there is at least one such assignment, then $A_{i-1}(t'_1, \ldots, t'_m)$ is nonempty, where $t'_1, \ldots, t'_m$ are variable-free principals such that, for some $j \in \{1, \ldots, m\}$ and for all $i \neq j$, $t'_i = t_i$ and either $q_j$ is a disjunct of $c_i$, $t_j \neq p_f$, and $t'_j = p_t$, or $\neg q_j$ is a disjunct of $c_i$, $t_j \neq p_t$, and $t'_j = p_f$. It follows from the induction hypothesis that $\mathbf{Said}(p_{i-1}, e(t'_1, \ldots, t'_m))$ holds and it follows from $L$ that $\mathbf{Said}(p_i, e(t_1, \ldots, t_m))$ holds as well. If there is no assignment in $A_{i-1}(t_1, \ldots, t_m)$ under which $c_i$ is **true** then, for every disjunct $q_j$ in $c_i$, $t_i = p_f$ and, for every disjunct $\neg q_j$ in $c_i$, $t_j = p_t$. It follows that $A_i(t_1, \ldots, t_m) = \emptyset$ and $\mathbf{Said}(p_i, e(t_1, \ldots, t_m))$ does not hold.

The desired result now follows quickly. It is easy to see that $\mathbf{Query2}(e, L, R, \emptyset) =$ **true** iff $\mathbf{Said}(p_1, e(p'_1, \ldots, p'_m))$ holds. Since none of $p'_1, \ldots, p'_m$ is $p_f$ or $p_t$, by definition, $A(p'_1, \ldots, p'_m)$ consists of all truth assignments. Thus, by the induction argument, it follows that $\mathbf{Query2}(e, L, R, \emptyset) =$ **true** iff $f = c_1 \wedge \ldots \wedge c_n$ is satisfiable. Moreover, it is easy to see that $(e, L, R, \emptyset) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0$, because the query does not mention union and, for every variable $x$ mentioned in a grant $g$ that is in $R \cup L$, $x$ is mentioned in the conclusion of $g$. $\square$

We next prove Theorem 5.3, which considers the complexity of determining whether $\mathbf{Query2}(e, L, R, E)$ returns **true** for $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$. In the statement of the theorem, we viewed $n$ and $h$ as constants. In our proof, we treat them as parameters, so as to bring out their role.

To prove the theorem we need three preliminary lemmas. The first uses the fact that, for every condition $d$, there is a dag (directed acyclic graph) $G_d$ such that $G_d$ represents $d$ and $G_d$ is no larger than $d$. To make this precise, recall that $|s|$ is the length of string $s$ when viewed as a string of symbols. For ease of exposition, we assume that each pair of parenthesis and set braces has length 2, and each comma has length 1. For a graph $G(V, E)$, let $|G| = \#(V) + \#(E)$. It is easy to see that a condition $d$ can be represented as a tree $T_d$, where $|T_d| \leq |d|$. For example, we can represent the condition $d = \mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Smart}(\mathbf{Amy})) \wedge$

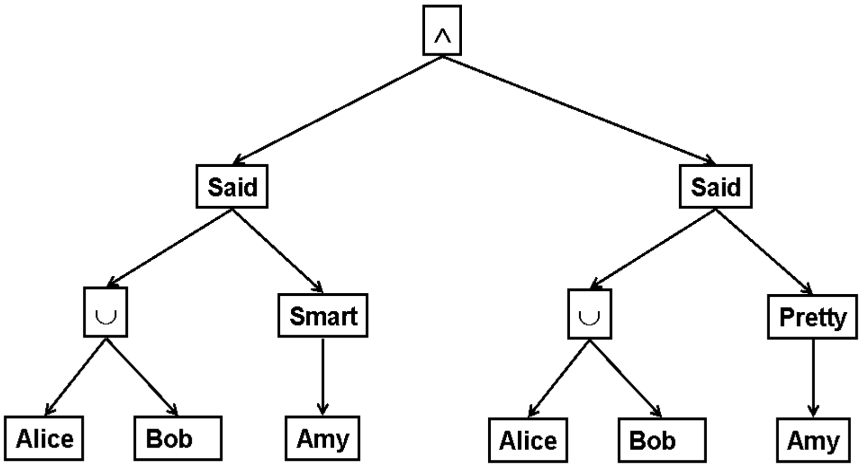


Fig. 7. A tree representing $\mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Smart}(\mathbf{Amy})) \wedge \mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Pretty}(\mathbf{Amy}))$

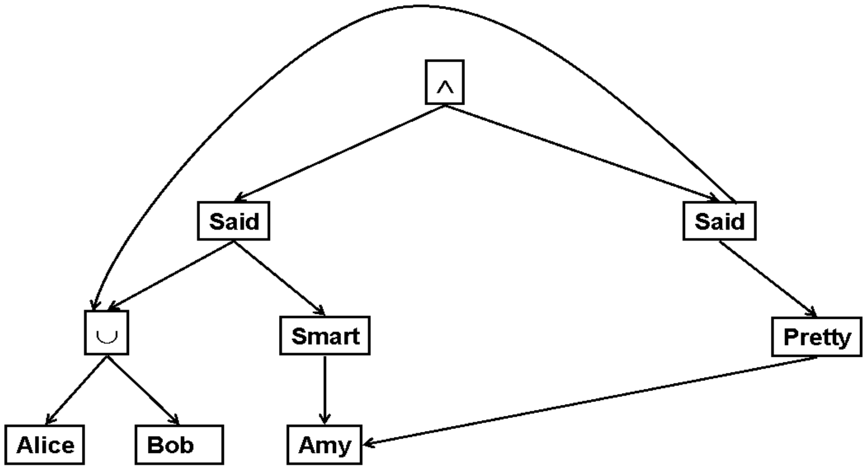


Fig. 8. A dag representing $\mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Smart}(\mathbf{Amy})) \wedge \mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Pretty}(\mathbf{Amy}))$

$\mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Pretty}(\mathbf{Amy}))$ as the tree $T_d$ shown in Figure 7. Note that $|d| = 27$ and, because the tree has 13 nodes and 12 edges, $|T_d| = 25$. By "merging" identical subtrees, we can create a dag representation of $d$ that can be substantially smaller than $|d|$. Continuing our example, the dag $D_d$ in Figure 8 represents the condition $\mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Smart}(\mathbf{Amy})) \wedge \mathbf{Said}(\{\mathbf{Alice}, \mathbf{Bob}\}, \mathbf{Pretty}(\mathbf{Amy}))$ and $|D_d| = 19$.

LEMMA A.10. *Suppose that $T$ is the call tree for an execution of* ***Holds2***$(d, L, \emptyset)$*;*

*every license in $L$ is restrained; the $\cup$ operator is not mentioned in $d$ or in a grant in $L$; and $v$ is a node in $T$ with label* ***Holds2***$(d', L, S)$*. If $G_d$ is a dag representing $d$, then there exists a dag $G_{d'}$ representing $d'$ such that $|G_{d'}| \leq h|L| + |G_d|$, where $h$ is the height of $T$.*

Proof. Because $v$ is a node in $T$, there is a path $v_0, \ldots, v_k$ in $T$ such that $v_0$ is the root of $T$ and $v_k = v$. We prove by induction on $k$ that there is a dag $G_{d'}$ representing $d'$ such that $|G_{d'}| \leq k|L| + |G_d|$. Since $k \leq h$ by assumption, it easily follows that $|G_{d'}| \leq h|L| + |G_d|$.

If $k = 0$, then $v$ is the root of $T$, so $d' = d$. If $k > 0$, then $v$ is the child of a node $v_{k-1}$. Let **Holds2**$(d'', L, S')$ be the label of $v_{k-1}$. The proof is by cases on the structure of $d''$. It follows from the description of **Holds2** that $d''$ is not **true** because $d''$ is not a leaf in $T$. If $d''$ is a conjunction, then $d'$ is a conjunct of $d''$. So the space needed to represent $d'$ is less than the space needed to represent $d''$, thus the result follows easily from the induction hypothesis. Finally, if $d''$ has the form **Said**$(p, e)$, then it follows from the description of **Holds2** that there is a license $(p, g) \in L$, where $g = \forall x_1 \ldots \forall x_m (d_g \rightarrow e_g)$, and a closed substitution $\sigma$ such that $d' = d_g\sigma$ and $e_g\sigma = e$. A dag representing $d_g\sigma$ (i.e., $d'$) can be obtained by taking a dag representing $d_g$ and replacing every variable $x$ by a dag representing $\sigma(x)$. Because every grant in $L$ is restrained, $g$ is restrained, so $\sigma$ assigns every variable of sort *Resource* mentioned in $d_g$ to a term in $e$. Since $\sigma(x)$ is a subterm of $e$ or a primitive principal, given a dag $G_{d_g}$ representing $d_g$ and a dag $G_e$ representing $e$, we can construct a dag $G_{d'}$ representing $d'$ such that $|G_{d'}| \leq |G_{d_g}| + |G_e|$. Since, for every condition $d$, there is a tree representation of $d$ whose size is at most $|d|$, there is a dag $G_{d_g}$ representing $d_g$ such that $|G_{d_g}| \leq |d_g|$. Because $d_g$ is the antecedent of a grant in $L$, $|d_g| < |L|$ so it follows that $|G_{d_g}| < L$. Because $e$ is a subterm of $d'' = \textbf{Said}(p, e)$, and by the induction hypothesis, there is a dag $G_{d''}$ representing $d''$ such that $|G_{d''}| \leq (k-1)|L| + |G_d|$, there is surely a dag $G_e$ representing $e$ such that $|G_e| \leq (k-1)|L| + |G_d|$. Putting this all together, it follows that there is a dag $G_{d'}$ representing $d'$ such that $|G_{d'}| \leq k|L| + |G_d|$. ☐

Lemma A.11. *If* ***Holds2***$(d, L, \emptyset)$ *is $h$-bounded, the $\cup$ operator is not mentioned in $d$ or in a grant in $L$, $L$ is both restrained and $n$-restricted, and $G_d$ is a dag representing $d$, then the output of* ***Holds2***$(d, L, \emptyset)$ *can be determined in time*

$$O(\max(|G_d|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |G_d|)(h + |L|))).$$

Proof. Let $T$ be the call tree for an execution of **Holds2**$(d, L, \emptyset)$. Our goal is to compute the truth value associated with the root of $T$, since that truth value is the output of **Holds2**$(d, L, \emptyset)$.

It is clear that once we have written the call tree, computing the truth value of the root can be done in time linear in the number of nodes in the tree. The obvious way to construct the tree is to start at the root and, for each node $v$, construct the successors of $v$ (if there are any). In constructing the call tree, we assume that the condition $d'$ and the elements of the set $S$ in a node labeled **Holds2**$(d', L, S)$ are described using the dags of Lemma A.10. Consider a node $v$ in $T$ that is labeled **Holds2**$(d', L, S)$ and is neither the root nor a leaf. Since $v$ is not a leaf, $d' \neq$ **true**. If $d'$ is a conjunction, then a bound on the number of conjuncts (and hence on the successors of the node) is $|L|$ since $d'$ is of the form $d_g\sigma$, where $d_g$ is the antecedent of a grant $g$ that is in $L$, and $\sigma$ is a closed substitution. It is easy to see that $d_g$, and hence $d_g\sigma$, has at most $|L|$ conjuncts, and these can be computed in time $O(|L|)$.

Suppose that $d'$ is of the form $\mathbf{Said}(p,e)$. If $d' \in S$, then $v$ is a leaf. Since the height of $T$ is at most $h$, $S$ has at most $h$ elements. It follows from Lemma A.10 that each of these elements can be represented using a dag of size at most $h|L|+|G_d|$, so checking whether $\mathbf{Said}(p,e) \in S$ can be done in time $O(h^2|L| + h|G_d|)$. If $d' \notin S$, then each child of $v$ has the form $d_g\sigma$, where $g = \forall x_1 \ldots \forall x_i(d_g \rightarrow e_g)$ is a grant in $L$ and $\sigma$ is a closed substitution such that $e_g\sigma = e$. Since every grant in $L$ is restrained and $n$-restricted, $d_g$ mentions at most $n$ variables that are not mentioned in $e_g$ and each of these variables is of sort *Principal*. Since $d$ and the grants in $L$ do not mention the $\cup$ operator and $\#(P_0) = |P_0|$, there are at most $|P_0|$ substitutions for each variable and thus $|P_0|^n$ possible substitutions $\sigma$. Finding $\sigma(x)$ for all of the variables $x$ that are mentioned in $e_g$ takes time linear in the size of the dag representing $e$ (since $e_g\sigma = e$). Clearly the dag representing $e$ has size less than that representing $d' = \mathbf{Said}(p,e)$. By Lemma A.10, the latter dag has size at most $h|L| + |G_d|$. Since $\#(L) \leq |L|$, there are at most $|L||P_0|^n$ children of $v$ and computing what they are takes time $O(|L||P_0|^n + (h|L| + |G_d|)(h + |L|))$.

Similarly, the root of $T$ has at most $\max(|G_d|, |L||P_0|^n)$ children since the root has zero children if $d = \mathbf{true}$, less than $|G_d|$ children if $d$ is a conjunction, and at most $|L||P_0|^n$ children if $d$ is a **Said** condition. The children of the root can be computed in time $O(|G_d|)$ if $d$ is a conjunction and in time $O(|L||P_0|^n + |G_d||L|)$ if $d$ is a **Said** condition. This follows from the reasoning given for the case when the node is neither the root nor a leaf modified to account for the fact that $d \notin S$, since $S = \emptyset$, and there is a dag representation of $d$ that has length $|G_d|$.

To determine the number of non-leaf nodes of $T$, observe that, if the root of $T$ has $n$ children and each subtree of $T$ has at most $m$ non-leaf nodes, then $T$ has at most $1 + nm$ non-leaf nodes. It follows that $T$ has at most $1 + 2\max(|G_d|, |L||P_0|^n)(|L||P_0|^n)^{h-2}$ non-leaf nodes, since a tree with outdegree at most $c$ and height $h$ has $c^h/(c-1) \leq 2c^{h-1}$ non-leaf nodes. Thus, it takes time

$$O(\max(|G_d|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |G_d|)(h + |L|)))$$

to compute the children of the $\max(|G_d|, |L||P_0|^n)(|L||P_0|^n)^{h-2}$ non-leaf nodes other than the root. Since this time dominates the time to compute the children of the root, it is also the time required to compute $T$.

Once $T$ is constructed, the truth value of its root can be computed in time linear in the number of nodes of $T$. Thus, $\mathbf{Holds2}(d, L, \emptyset)$ can be computed in time

$$O(\max(|G_d|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |G_d|)(h + |L|))).$$

□

LEMMA A.12. *Suppose that $(e, L, R, E)$ is a query in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$ such that $e \notin E$ and $D$ is the output of $\boldsymbol{Auth2}(e, L, R, E)$. Then*

*(a) $\#(D)$ is at most $\#(P_0)^n(\#(R) + \#(L))$;*

*(b) if $d$ is a closed condition in $D$, then there is a dag $G_d$ representing $d$ such that $|G_d| \leq |R| + |L| + |e|$; and*

*(c) $D$ can be computed in time $O(|L||E \cup \{e\}| + |L|^2\log(|R| + 1) + |L|^2(|L||P_0|^n)^{h+1}h^2)$.*

PROOF. Let $X$ be an execution of $\mathbf{Query2}(e, L, R, E)$ and let $G = G(e, L, R, E, X)$.

For part (a), by Theorem A.9(b), if $e \notin E$, then

$$D = \{d \mid \text{for some grant } \forall x_1 \ldots \forall x_m(d_g \to e_g) \in G \text{ and closed substitution } \sigma, \quad (1)$$
$$d_g\sigma = d \text{ and } e_g\sigma = e\}.$$

Since every grant in $G$ is either in $R$ or $L$, $\#(G) \leq \#(R)+\#(L)$. Moreover, because $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_2^n$, for every grant $g = \forall x_1 \ldots \forall x_m(d_g \to e_g) \in G$, there are at most $n$ variables mentioned in $d_g$ that are not mentioned in $e_g$, and each of these variables is of sort *Principal*. As in the proof of Lemma A.11, it follows that there are at most $\#(P_0)^n$ substitutions of variables in $g$ to closed terms such that $e_g\sigma = e$ because $(e, L, R, E) \in \mathcal{L}_1$. Part (a) follows immediately.

For part (b), let $d$ be a closed condition in $D$. By (1), $d = d_g\sigma$, where $d_g$ is the antecedent of a grant $g \in G$ and $\sigma$ is a closed substitution. By the proof of part (a), $\sigma$ assigns every variable in $d_g$ to a term in $e$ or to a principal in $P_0$. Given dags $G_e$ and $G_{d_g}$ representing $e$ and $d_g$, respectively, we can obtain a dag $G_d$ representing $d$ by replacing every variable in $G_{d_g}$ by either a subgraph of $G_e$ or by some $p \in P_0$. So there is a dag $G_d$ representing $d$ such that $|G_d| \leq |G_{d_g}| + |G_e|$. Recall that, for every string $s$, there is a dag $G_s$ representing $s$ such that $|G_s| \leq |s|$. So there is a dag $G_d$ representing $d$ such that $|G_d| \leq |d_g| + |e|$. Since $d_g$ is the antecedent of a grant in $G$ and every grant in $G$ is a grant in $R$ or $L$, $|d_g| < |R| + |L|$, and we are done.

For part (c), by (1), we can compute $D$ by (i) checking whether $e \in E$; (ii) computing $G$; and (iii) for each grant $g = \forall x_1 \ldots \forall x_m(d_g \to e_g) \in G$, computing $D_g = \{d \mid \text{for some closed substitution } \sigma, d_g\sigma = d \text{ and } e_g\sigma = e\}$. (Observe that these are the same steps taken in $\mathbf{Auth2}$; however, our approach computes $G$ more efficiently.) Step (i) takes time $O(|E|)$. We show below that $G$ can be completed in time $O(|L|^h|P_0|^n(2^{h-1}+|L|^2|P_0|^{n(h-1)})(|P_0|^n+h^2+h|L|)+|L|^2\log(|R|+1)+|L|(|E|+|e|))$. For step (iii), essentially the same arguments as those used in Lemma A.11 show that, given grant $g \in G$, $D_g$ can be computed in time $O(|e| + |e_g| + |P_0|^n|d_g|)$. So, $\{D_g \mid g \in G\}$ can be computed in time $O(|G|(|e|+|P_0|^n))$. Since $|G| \leq |R|+|L|$, the total time needed to compute $D$ is $O(|E| + |L|^h|P_0|^n(2^{h-1} + |L|^2|P_0|^{n(h-1)})(|P_0|^n + h^2 + h|L|) + |L|^2\log(|R| + 1) + |L|(|E| + |e|) + |R|(|e| + |P_0|^n))$.

For step (ii), let $A = A(e, L, R, E, X)$. For all integers $k \geq 0$, define the set $G'_k$ of grants inductively as follows: $G'_0 = R$ and, for $i > 0$, $G'_i = R \cup \{g \mid \text{for some principal } p, (p, g) \in L \text{ and } \bigwedge_{g' \in G'_{i-1}} g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g) \text{ is acceptably valid}\}$. We claim that $G'_{\#(L)} = G$.

To show that $G'_{\#(L)} \subseteq G$, we prove by induction that $G'_i \subseteq G$ for all $i \geq 0$. The base case is immediate because $G'_0 = R$. For the inductive step, it suffices to show that, if there is a license $(p, g) \in L$ and a subset $G' \subseteq G$ such that $\bigwedge_{g' \in G'} g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid, then $g \in G$. Let $\varphi = ((\bigwedge_{\ell \in L} \ell^{L,A,\emptyset,(E\cup\{e\})}) \wedge (\bigwedge_{g \in R} g^{L,A,\emptyset,(E\cup\{e\})}))$. Because $(p, g) \in L$, it is immediate from the definition of $G$ that $g \in G$ if $\varphi \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid. Because $G' \subseteq G$, every grant $g' \in G'$ is either in $R$ or there is a principal $p'$ such that $(p', g') \in L$ and $\varphi \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ is acceptably valid. It follows that $\varphi \Rightarrow \bigwedge_{g' \in G'} g'^{L,A,\emptyset,(E\cup\{e\})}$ is acceptably valid. Since $\bigwedge_{g' \in G'} g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow$

$\mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid, $\varphi \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid.

To show that $G \subseteq G'_{\#(L)}$, we first observe that, for all $i$, $G'_i \subseteq G'_{i+1}$ and, if $G'_i = G'_{i+1}$, then $G'_i = G'_{i+j}$ for all $j > 0$. Since $G'_0 = R$ and $G'_i \subseteq R \cup \{g \mid \text{for some principal } p, (p,g) \in L\}$, it follows that $G'_{\#(L)} = G'_{\#(L)+1}$. To show that $G \subseteq G'_{\#(L)}$, it suffices to show that for all licenses $(p, g) \in L$ such that $\varphi \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid, $g \in G'_{\#(L)}$. Suppose by way of contradiction that there is a license $(p, g) \in L$ such that $\varphi \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid and $g \notin G'_{\#(L)}$. Let $\varphi' = \bigwedge_{g' \in G'_{\#(L)}} g'^{L,A,\emptyset,(E\cup\{e\})}$. Since $G'_{\#(L)} = G'_{\#(L)+1}$, the grant $g \notin G'_{\#(L)+1}$ so, by the definition of $G'_{\#(L)+1}$, the formula $\varphi' \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is not acceptably valid. It follows that there is an acceptable model $m$ that satisfies $\varphi' \wedge \neg\mathbf{Perm}(p, \mathtt{issue}, c_g)$ and is "most forbidding" in the sense that, for all principals $p'$ and grants $g'$, either $m$ does not satisfy $\mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ or the model $m'$ that does not satisfy $\mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ and is otherwise identical to $m$ does not satisfy $\varphi'$. Since $m$ satisfies $\neg\mathbf{Perm}(p, \mathtt{issue}, c_g)$ and $\varphi \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid, $m$ does not satisfy $\varphi$. Because $R \subseteq G'_{\#(L)}$ and $m$ satisfies $\varphi'$, $m$ satisfies $\bigwedge_{g' \in R} g'^{L,A,\emptyset,(E\cup\{e\})}$. So, there is a license $(p', g') \in L$ such that $m$ does not satisfy $(p', g')^{L,A,\emptyset,(E\cup\{e\})}$. If $\mathbf{Perm}(p', \mathtt{issue}, g') \in E \cup \{e\}$, then $(p', g')^{L,A,\emptyset,(E\cup\{e\})} = \mathbf{true}$, so $m$ satisfies $(p', g')^{L,A,\emptyset,(E\cup\{e\})}$. Thus, $\mathbf{Perm}(p', \mathtt{issue}, g') \notin E \cup \{e\}$. But then $(p', g')^{L,A,\emptyset,(E\cup\{e\})} = \mathbf{Perm}(p', \mathtt{issue}, c_{g'}) \Rightarrow g'^{L,A,\emptyset,(E\cup\{e\})}$. Since $m$ does not satisfy this formula, $m$ satisfies $\mathbf{Perm}(p', \mathtt{issue}, c_{g'})$. By the construction of $m$, the model $m'$ that does not satisfy $\mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ and is otherwise identical to $m$ does not satisfy $\varphi'$. So there is a grant $g'' = \forall x_1 \ldots \forall x_n(d_{g''} \rightarrow e_{g''}) \in G'_{\#(L)}$ such that $m'$ does not satisfy $g''^{L,A,\emptyset,(E\cup\{e\})}$. Because $m$ satisfies $g''^{L,A,\emptyset,(E\cup\{e\})}$ and the two models $m$ and $m'$ differ only in their interpretation of $\mathbf{Perm}(p', \mathtt{issue}, c_{g'})$, it follows from the translation of $g''$ that there is a substitution $\sigma$ such that $e_{g''}\sigma = \mathbf{Perm}(p', \mathtt{issue}, g')$, $e_{g''}\sigma \notin E \cup \{e\}$, and $d_{g''}\sigma^{L,A,\emptyset,(E\cup\{e\})}$ is valid. So $g''^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ is acceptably valid. Since $g'' \in G'_{\#(L)}$, $\varphi' \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, c_{g'})$ is acceptably valid, $g' \in G'_{\#(L)+1}$. Because $G'_{\#(L)+1} = G'_{\#(L)}$, the grant $g' \in G'_{\#(L)}$ and, since $m$ satisfies $\varphi'$, $m$ satisfies $g'^{L,A,\emptyset,(E\cup\{e\})}$. So $m$ satisfies $(p', g')^{L,A,\emptyset,(E\cup\{e\})}$, which contradicts the assumptions.

We next consider the complexity of computing $G = G'_{\#(L)}$. Let $L' = \{(p, g) \in L \mid \mathbf{Perm}(p, \mathtt{issue}, g) \notin E \cup \{e\}\}$. Clearly, we can compute $L'$ in time $c_0|L||E \cup \{e\}|$ for some constant $c_0$. For all $k > 1$, let $L'_k = \{(p, g) \in L' \mid g \notin G'_k\}$ and let $G''_k = G'_k - G'_{k-1}$. We plan to compute $G'_k$ inductively, It will be useful in the induction to represent the elements of $G'_k$ in a *splay tree*. (Recall that a splay tree is a form of binary search tree such that $k$ insertions and searches can be done in a tree with at most $n$ nodes in time $O(k \log n)$ [Sleator and Tarjan 1983].) If $G'_k$ is represented as a splay tree, then we can compute $L'_k$ in time $O(|L| \log(|L| + |R|))$ (since $G'_k \subseteq L \cup R$).

For $0 < k < \#(L)$,

$$\begin{aligned} G''_{k+1} = \{g \mid & \text{ for some principal } p,\ (p, g) \in L'_k \text{ and} \\ & \textstyle\bigwedge_{g' \in G''_k} g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g) \text{ is acceptably valid}\}. \end{aligned}$$

By Lemma A.6,

$G''_{k+1} = \cup_{(p,g)\in L'_k} \cup_{g'\in G''_k} \{g \mid g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid$\}$.

Moreover, it follows from Lemma A.6 that, for $(p, g) \in L'$, $g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid iff the formula $d_{g'}\sigma$ is valid for some $A$-closed substitution $\sigma$ such that $e_{g'}\sigma = \mathbf{Perm}(p, \mathtt{issue}, c_g)$, where $g' = \forall x_1 \ldots \forall x_n(d_{g'} \Rightarrow e_{g'})$. Given $(p, g) \in L'$ with $g \notin G'_k$ and $g' \in G''_k$, we can clearly check in time $c_1(|e_{g'}| + |(p, g)|)$ if there exists an $A$-closed substitution $\sigma$ such that $e_{g'}\sigma = \mathbf{Perm}(p, \mathtt{issue}, g)$, where $c_1$ is a constant independent of $k$. If so, as in part (a), there are at most $\#(P_0)^n$ distinct formulas of the form $d_{g'}\sigma$ (since there are at most $\#(P_0)^n$ possible substitutions for the free variables in $d_{g'}$). It follows from Theorem A.9(a) that $d_{g'}\sigma^{L,A,\emptyset,(E\cup\{e\})}$ is valid iff $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset) =$ **true**. We show shortly that there is an execution of $\mathbf{Query2}(e, L, R, E)$ that calls $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset)$, so $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset)$ is $h$-bounded. It follows from Lemma A.11 that we can determine if $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset) =$ **true** in time $c_2 \max(|G_{d_{g'}\sigma}|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |G_{d_{g'}\sigma}|)(h + |L|))$, where $c_2$ is a constant independent of $k$ and $G_{d_{g'}\sigma}$ is a dag representing $d_{g'}\sigma$. As in the proof of part (b), we can obtain $G_{d_{g'}\sigma}$ from a dag $G_{d_{g'}}$ representing $d_{g'}$ by replacing every variable with a principal in $P_0$ or a resource mentioned in $\mathbf{Perm}(p, \mathtt{issue}, g)$. So there is a dag $G_{d_{g'}\sigma}$ representing $d_{g'}\sigma$ such that $|G_{d_{g'}\sigma}| < |d_{g'}| + |g|$. Repeating this process for each of the at most $|P_0|^n$ formulas $d_{g'}\sigma$, it follows that we can check if $g'^{L,A,\emptyset,(E\cup\{e\})} \Rightarrow \mathbf{Perm}(p, \mathtt{issue}, c_g)$ is acceptably valid in time $c_2|P_0|^n \max(|d_{g'}|+|g|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L|+|d_{g'}|+|g|)(h+|L|))$.

Assuming we have already computed $L'_k$ and $G''_k$, we can repeat the process above for all $g' \in G''_k$ and $(p, g) \in L'_k$. It is not hard to show that we can compute $G''_{k+1}$ in time

$$\begin{aligned}
& \textstyle\sum_{g'\in G''_k} \sum_{(p,g)\in L'_k} c_1(|e_{g'}| + |(p, g)|)+ \\
& c_2|P_0|^n \max(|d_{g'}| + |g|, |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |d_{g'}| + |g|)(h + |L|)) \\
\le\ & 2c_1|G''_k||L| + c_2|P_0|^n(|L||P_0|^n)^{h-2}(h + |L|)\cdot \\
& \textstyle\sum_{g'\in G''_k} \sum_{(p,g)\in L}(|d_{g'}| + |g| + |L||P_0|^n)(|L||P_0|^n + h|L| + |d_{g'}| + |g|) \\
\le\ & 2c_1|G''_k||L| + c_2|P_0|^n(|L||P_0|^n)^{h-2}(h + |L|)2|G''_k||L|^2|P_0|^n(|L||P_0|^n + h|L| + |G''_k| + |L|) \\
\le\ & 2c_1|G''_k||L| + 2c_2|G''_k|(|L||P_0|^n)^h(h + |L|)(|L||P_0|^n + h|L| + |G''_k| + |L|) \\
\le\ & c_3|G''_k|(|L||P_0|^n)^h(h + |L|)(|L||P_0|^n + h|L| + |G''_k| + |L|)
\end{aligned}$$

for some constant $c_3$. We can then build the splay tree for $G'_{k+1}$ by inserting the grants in $G''_k$ into the splay tree for $G'_k$; this can be done in time $O(|G''_k| \log(|L| + |R|))$.

Since $\cup_{k=1}^{|L|} G''_k \subseteq L$, the total time to compute $G''_1, \ldots, G''_k$ (ignoring the time to compute the sets $L'$ and $L'_k$, and to build the splay trees for $G'_k$) is at most

$$c_4|L|^2(|L||P_0|^n)^{h+1}h^2$$

for some constant $c_4$; i.e., it is $O(|L|^2(|L||P_0|^n)^{h+1}h^2)$.

Now taking into account the complexity of computing $L'$ and $L'_k$ and to build the splay trees, and using the observation that $\log(a + b) \le \log(a + 1) + \log(b + 1)$, we get that the complexity for computing $G$ is

$$O(|L||E \cup \{e\}| + |L|^2 \log(|R| + 1) + |L|^2(|L||P_0|^n)^{h+1}h^2).$$

It remains to show that if $g' = \forall x_1 \ldots \forall x_n(d_{g'} \rightarrow e_{g'}) \in G'_k - G'_{k-1}$, $(p,g) \in L'$ with $g \notin G'_k$, and $e_{g'}\sigma = \mathbf{Perm}(p, \mathtt{issue}, g)$ and $A$-closed substitution $\sigma$, then there is an execution $X$ of $\mathbf{Query2}(e, L, R, E)$ that calls $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset)$. Because $e \notin E$ by assumption, $\mathbf{Query2}(e, L, R, E)$ calls $\mathbf{Auth2}(e, L, R, E)$, which calls $\mathbf{Query2}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\})$, which calls $\mathbf{Auth2}(\mathbf{Perm}(p, \mathtt{issue}, gR), L, R, E \cup \{e\})$. Since $(p, g) \in L'$, $\mathbf{Perm}(p, \mathtt{issue}, g) \notin E \cup \{e\}$. It follows that $\mathbf{Auth2}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\})$ computes $G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$ and, if $g' \in G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$, then $\mathbf{Auth2}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\})$ returns a set $D$ that includes $d_{g'}\sigma$. After $\mathbf{Auth2}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\})$ returns $D$, it is easy to see that some execution of $\mathbf{Query2}(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\})$ calls $\mathbf{Holds2}(d_{g'}\sigma, L, \emptyset)$. So, in short, it suffices to show that $g' \in G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$. The proof is by induction on $k$. If $k = 0$, then $g' \in R \subseteq G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$. If $k > 0$ then, by the induction hypothesis, $G'_{k-1} \subseteq G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$, so $\bigwedge_{\ell \in L} \ell^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}} \wedge \bigwedge_{g''' \in R} g'''^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}} \Rightarrow \bigwedge g'' \in G'_{k-1} g''^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}}$ is acceptably valid. Since $g' \in G'_k - G'_{k-1}$, there is a grant $g'' \in G'_{k-1}$ and a principal $p'$ such that $(p', g') \in L$ and $g''^{L,A,\emptyset,E\cup\{e\}} \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, g')$ is acceptably valid. Because $g' \in G'_k$ and $g \notin G'_k$, $g \neq g'$ and, thus, it follows from the translation that $g''^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}} \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, g')$ is acceptably valid. Putting the pieces together, there is a principal $p'$ such that $(p', g') \in L$ and $\bigwedge_{\ell \in L} \ell^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}} \wedge \bigwedge_{g''' \in R} g'''^{L,A,\emptyset,E\cup\{e\cup\mathbf{Perm}(p,\mathtt{issue},g)\}} \Rightarrow \mathbf{Perm}(p', \mathtt{issue}, g')$ is acceptably valid, so $g' \in G(\mathbf{Perm}(p, \mathtt{issue}, g), L, R, E \cup \{e\}, X)$. $\square$

We are now ready to prove Theorem 5.3.

THEOREM 5.3. *For fixed $n$ and $h$, if $(e, L, R, E) \in \mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^n \cap \mathcal{L}_3^h$, then determining whether **Query2**$(e, L, R, E)$ returns* **true** *takes time $O(|L||E| + (|R| + |L|)(|L|^{h-1}(|L| + |R| + |e|)^2))$.*

PROOF. Let $D$ be the output of $\mathbf{Auth2}(e, L, R, E)$. It is immediate from the description of **Query2** that $\mathbf{Query2}(e, L, R, E) = \mathbf{true}$ iff there is some condition $d \in D$ such that $\mathbf{Holds2}(d, L, \emptyset) = \mathbf{true}$. So the output of $\mathbf{Query2}(e, L, R, E)$ can be determined in time $T + \#(D)T'$, where $T$ is the time needed to compute $D$ and $T'$ is the time needed to determine the output of $\mathbf{Holds2}(d, L, \emptyset)$ for a condition $d \in D$. By Lemma A.12(c), $\mathbf{Holds2}(d, L, \emptyset)$ for a condition $d \in D$. By Lemma A.12(c), $T = c_1(|L||E \cup \{e\}| + |L|^2 \log(|R|+1) + |L|^2(|L||P_0|^n)^{h+1}h^2)$ for some constant $c_1$. If $n$ and $h$ are treated as constants, then $T = c'_1(|L||E \cup \{e\}| + |L|^2 \log(|R|+1) + |L|^{h+3})$ for some constant $c'_1$; i.e., $T$ is $O(|L||E \cup \{e\}| + |L|^2|R| + |L|^{h+3})$.

By Lemma A.12(a), $\#(D) \leq \#(P_0)^n(\#(R) + \#(L))$. By Lemma A.11, $T'$ is at most $c_2(|G_d| + |L||P_0|^n)(|L||P_0|^n)^{h-2}(|L||P_0|^n + (h|L| + |G_d|)(h + |L|))$, for some constant $c_2$. If $n$ and $h$ are treated as constants, then there is a constant $c'_2$ such that $T'$ is at most

$$\begin{aligned} & c_2'(|G_d| + |L|)|L|^{h-2}(|L| + (|L| + |G_d|)|L|) \\ = \; & c_2'|L|^{h-1}(|G_d| + |L|)(1 + (|L| + |G_d|)) \\ = \; & c_2'|L|^{h-1}(|G_d| + |L|)(2(|G_d| + |L|)) \\ \leq \; & 2c_2'|L|^{h-1}(|G_d| + |L|)^2. \end{aligned}$$

Since, by Lemma A.12(b), $|G_d| \leq |R|+|L|+|e|$, it follows that $T' \leq 2c_2'|L|^{h-1}(2|L|+|R|+|e|)^2$, i.e., $O(|L|^{h-1}(|L|+|R|+|e|)^2)$.

Since $\#(D) \leq \#(P_0)^n(\#(R) + \#(L)) \leq |P_0|^n(|R| + |L|)$, a straightforward computation shows that $T + \#(D)T'$, the time needed to determine whether **Query2**$(e, L, R, E)$ returns **true**, is $O(|L||E| + (|R| + |L|)(|L|^{h-1}(|L| + |R| + |e|)^2))$. □

THEOREM 7.2. *Let $(e, L, R, E)$ be a tuple in $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_2^0 \cap \mathcal{L}_3^2$ extended to include negated* **Said** *conditions and negated conclusions. The problem of deciding whether*

$$\bigwedge_{\ell \in L} \ell^{L,A,S,E} \wedge \bigwedge_{g \in R} g^{L,A,S,E} \Rightarrow e^{L,A,S,E}$$

*is valid is NP-hard. This result holds even if $e$, all of the licenses in $L$, and all of the conclusions in $E$ are in XrML, all but one of the grants in $R$ is in XrML, and the one grant that is in in $XrML^{\neg}$ – XrML is of the form $\forall x_1 \ldots \forall x_n(\neg e)$.*

PROOF. The proof is by reduction of the 3-satisfiability problem. The reduction is identical to the reduction given in the proof for the case of $\mathcal{L}_0 \cap \mathcal{L}_1 \cap \mathcal{L}_3^2$ in Theorem 5.2, except that $R = \{\forall x_1 \ldots \forall x_m((\bigwedge_{i=1,\ldots,n} \mathbf{Said}(p_i, e')) \rightarrow e'), \neg e'\}$. To show that **Query2**$(e, L, R, \emptyset)$ = **true** iff $f$ is valid, we observe that **Query2**$(e, L, R, \emptyset)$ = **true** iff $L$ and $R$ imply **false**, which occurs iff $\bigwedge_{i=1,\ldots,n} \mathbf{Said}(p_i, e')\sigma$ holds for some substitution $\sigma$. The rest of the argument proceeds as in the proof of Theorem 5.2. □

## REFERENCES


BECKER, M. Y. AND SEWELL, P. 2004. Cassandra: Flexible trust management, applied to electronic health records. In *Proc. 17th IEEE Computer Security Foundations Workshop*. 139–154.

CONTENTGUARD. 2001. XrML: The digital rights language for trusted content and services. Available at http://www.xrml.org/.

DETREVILLE, J. 2002. Binder, a logic-based security language. In *Proc. 2002 IEEE Symposium on Security and Privacy*. 95–103.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999a. Simple public key certificate. Available at http://world.std.com/∼cme/spki.txt. Internet RFC 2693.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999b. SPKI certificate theory. Available at http://www.ietf.org/html.charters/spki-charter.html. Internet RFC 2693.

HALPERN, J. Y. AND VAN DER MEYDEN, R. 2003. A logical reconstruction of SPKI. *Journal of Computer Security 11,* 4, 581–614.

HALPERN, J. Y. AND WEISSMAN, V. 2003. Using first-order logic to reason about policies. In *Proc. 16th IEEE Computer Security Foundations Workshop*. 187–201.

HALPERN, J. Y. AND WEISSMAN, V. 2004. A formal foundation for XrML. In *Proc. 17th IEEE Computer Security Foundations Workshop*. 251–263.

IANNELLA, R. 2001. ODRL: The open digital rights language initiative. Available at http://odrl.net/.

Jajodia, S., Samarati, P., and Subrahmanian, V. 1997. A logical language for expressing authorizations. In *Proc. 1997 IEEE Symposium on Security and Privacy*. 31–42.

Jim, T. 2001. Sd3: A trust management system with certified evaluation. In *Proc. 2001 IEEE Symposium on Security and Privacy*. 106–115.

Li, N., Grosof, B. N., and Feigenbaum, J. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur. 6,* 1, 128–171.

Li, N. and Mitchell, J. C. 2006. Understanding spki/sdsi using first-order logic. *International Journal of Information Security 5,* 1, 48–64.

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

Moses, T. 2005. XACML: The eXtensible Access Control Markup Language, Version 2.0. Available at http://www.xacml.org.

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

Nerode, A. and Shore, R. 1997. *Logic for Applications*, 2nd ed. Springer-Verlag, New York.

Post, E. 1946. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society 52*, 264–268.

Sleator, D. and Tarjan, R. 1983. A data structure for dynamic trees. *Journal of Computer and System Sciences 26,* 3, 362–391.