A Logic for Reasoning about Digital Rights*

Riccardo Pucella
Cornell University
Ithaca, NY 14853
riccardo@cs.cornell.edu

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

Abstract

We present a logic for reasoning about licenses, which are "terms of use" for digital resources. The logic provides a language for writing both properties of licenses and specifications that govern a client's actions. We discuss the complexity of checking properties and specifications written in our logic and propose a technique for verification. A key feature of our approach is that it is essentially parameterized by the language in which the licenses are written, provided that this language can be given a trace-based semantics. We consider two license languages to illustrate this flexibility.

1 Introduction

In the world of digital rights management, licenses are agreements between the distributors and consumers of digital resources. A license is issued by an owner to a prospective client. It states the exact conditions under which a particular resource may be used, including a complete description of how compensation may be given. Licenses can be viewed as a subset of authorization policies, policies that dictate what actions a system's principal can perform at any given time. Licenses are an essential part of any rights management system, because they tell the consumer, as well as the enforcement mechanism, which uses are legitimate.

Licenses must be written in some language. Although many licenses are very simple (e.g., "consumer must pay a fee before each access to an on-line journal"), more complicated ones, in particular ones involving time, are also common (e.g., "for each month from 1/1/01 to 1/1/02 the mortgage requires either a $1500 payment between the first and fourth of the month or a $1525 payment between the fourth and the fourteenth"). The language must be expressive enough to capture these types of licenses. Languages such as *DPRL* [Ramanujapuram and Ram 1998], *XrML* [ContentGuard, Inc. 2000], and *ODRL* [IPR Systems Pty Ltd 2001] have been developed to state a wide range of licenses. These languages, however, do not have formal semantics. Instead, they rely on intuitions behind their syntax, and on informal descriptions of expected behavior. As a consequence, licenses that "seem right" are enforced without anyone knowing precisely what is intended or exactly what is allowed.

Gunter *et al.* [2001] used techniques from programming language semantics [Hoare 1985] to remove these ambiguities. In their approach, the meaning of a license is a set of traces. Each trace

*This paper is essentially the same as one that appeared in the *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 282–294, 2002.

represents a sequence of actions allowed by the license. A correct enforcement mechanism permits any sequence of action specified by the license and forbids any other. To illustrate their idea, Gunter *et al.* defined a simple language with semantics that could be used to state a number of licenses precisely.

In addition to unambiguously expressing licenses, we would like to reason about them. In general, we are interested in two classes of questions: does a set of licenses have certain properties and does a client's actions with respect to a set of licenses meet particular specifications. Note that we make a distinction between the characteristics inherent in a set of licenses (properties, sometimes referred to as license properties for emphasis). and those whose truth depends on the client's actions (specifications, sometimes referred to as client behavior specifications for emphasis) Examples of properties include "a religious work may only be viewed during the hour before sunset" and "if a user accesses a work, then the user is obligated to pay for the access at some time." Depending on the licenses, each property may or may not be easy to check. Continuing the last example, an owner may allow a client to defer payment in so many situations that it is not clear that there will ever be an occasion when the client must pay. Alternatively, a license may permit free access to some resources, however, the license has so much "red tape" that the client cannot determine if the desired resource actually is free. As for specifications, examples include "the client never uses a resource illegally" and "the client is never obligated to pay interest on her credit card debt". The difficulty of specification checking is based on the licenses and the client's actions. Verifying properties and specifications is important, because it increases our confidence that the licenses match the informal requirements and that the informal requirements match the owner's intent.

In this paper we present a logic for reasoning about licenses that provides us with a language in which we can state properties and specifications precisely. The logic is essentially a temporal logic. It allows us to make statements about issued licenses, assuming the licenses are written in some particular language that is distinct from our logic. For ease of exposition, we assume until Section 4 that licenses are written in a very simple, regular language and that the application has only one client and one provider. Our framework can be modified in a straightforward manner to reason about different license languages. It is also easy to extend the logic to multiple clients and providers.

As the examples suggest, license properties and client behavior specifications typically involve the client's permissions and obligations to do certain actions. We take a very simple view of permissions and obligations. In particular, we focus exclusively on the client's viewpoint. Inspired by Gunter *et al.*, we interpret licenses as describing a set of legal sequences of actions. A client is permitted to do an action if that action is part of a sequence of actions that is legal according to the actions she has already done and the licenses issued. If there is only one such action for a particular license, then the client is obligated to do that action.

To illustrate our notions of permission and obligation, consider the mortgage example in which the client must pay either \$1500 between the first and fourth or \$1525 between the fourth and the fourteenth of every month from 1/1/01 to 1/1/02. For the first month, there are two legal action sequences. The client could pay \$1500 before the fourth. Alternatively, the client could pay \$1525 between the fourth and the fourteenth. Since there is a legal action sequence in which the client pays before the fourth and one in which the client does not, we say that the client is permitted, but not obligated, to make the earlier payment. If the client doesn't make the earlier payment, then the only legal sequence she can be following is the second one. In this case, she is obligated to complete that sequence by paying \$1525 before the fourteenth.

Why are we designing *a logic* for reasoning about licenses? A logic provides us with a formal language in which to write properties and specifications. In addition, it allows us to check in a provably correct way that a property or specification holds for a particular set of licenses and, in the case of specification, a client's behavior. We can automate the analysis, by developing model checking techniques. It turns out that standard model checking procedures (as given in [Clarke, Grumberg, and Peled 1999]) apply to our framework. These procedures can form the foundation of enforcement mechanisms that are well-grounded in formal methods.

The design of our logic was strongly influenced by the work of Halpern and van der Meyden [2001a, 2001b] on reasoning about SPKI/SDSI. It is also reminiscent of deontic logic approaches, which aim at reasoning about ideal and actual behavior [Meyer and Wieringa 1993]. Deontic logic has been used extensively to analyze the structure of normative law and normative reasoning in law. (For examples, please see [Wieringa and Meyer 1993] and the references therein.)

In the next section, we introduce our logic. Section 3 examines the complexity of checking that a license property or client behavior specification holds. In Section 4, we show that our logic can be adapted to different license languages, by replacing our regular language with a variant of *DigitalRights* [Gunter, Weeks, and Wright 2001]. We discuss related work in Section 5. Proofs of our technical results can be found in the appendix.

2 The logic

We want to reason about licenses and client's actions with respect to licenses. To do this, we introduce a logic, $\mathcal{L}^{lic}$, that allows us to talk about licenses and actions. Formulas in $\mathcal{L}^{lic}$ include permission and obligation operators, as well as temporal operators, because we want to write formulas that represent interesting properties and specifications; the ones that state the conditions under which actions are permitted or obligatory. In this section, we give the syntax for our logic, followed by its semantics.

2.1 Syntax

The syntax of $\mathcal{L}^{lic}$ has three categories; formulas $(\varphi, \psi, \dots)$, actions $(\alpha, \dots)$, and licenses $(\ell, \dots)$. Their definitions assume a set $Names$ of license names, a set $Works$ of works (i.e. resources), and a set $Devices$ of devices (i.e. ways to access resources). Actions are taken from a set $Act = \{\mathsf{render}[w, d] \ : \ w \in Works, d \in Devices\} \cup \{\mathsf{pay}[x] \ : \ x \in \mathbb{R}\} \cup \{\bot\}$, where $\bot$ represents the null or "do nothing" action. (For simplicity, we consider only render and pay actions, as was done in [Gunter, Weeks, and Wright 2001].) Also, we let Lic be the set of licenses ℓ. In the following formal description, $n \in Names$ and $a \in Act$.

$$\begin{array}{rcl} \varphi & ::= & n : \ell \mid \alpha \mid P\alpha \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \\ \alpha & ::= & (a, n) \mid (\overline{a}, n) \\ \ell & ::= & a \mid \ell_1\, \ell_2 \mid \ell^* \mid \ell_1 \cup \ell_2 \end{array}$$

Intuitively, $n : \ell$ means "the license whose legitimate action sequences are described by the regular expression ℓ is being issued now and will be referred to by the name n." The primitive action (a, n) means "action a is performed with respect to license named n". The action $(\overline{a}, n)$ represents any action-name pair where the action is not a, but the license name is n. $P\alpha$ indicates that the action expression α is permitted. The set of formulas are closed under $\wedge$, $\neg$, $\Box$, $\bigcirc$ and $\mathcal{U}$, which are well-known operators from classical and temporal logic [Goldblatt 1992].[1] We use the standard abbreviations $\varphi \vee \psi$ for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \Rightarrow \psi$ for $\neg\varphi \vee \psi$, and $\Diamond\varphi$ for $\neg\Box\neg\varphi$. Also, we abbreviate the action (a, n) as a_n. For instance, $(\mathsf{render}[w, d], n)$ is written $\mathsf{render}_n[w, d]$, and $(\perp, n)$ is written $\perp_n$.

We use the abbreviation $O(a, n)$ to stand for $\neg P(\overline{a}, n)$. As we shall see later, the interpretation of $O(a, n)$ is that the client is obligated to perform action a with respect to the license named n.

To illustrate how our logic can be used in practice, consider the following scenario. Suppose an owner of an on-line journal requires a fee to be paid before each access. This license ℓ can be written in our logic as:

$$\ell = ((\mathsf{pay}[\mathsf{fee}](\perp)^*\mathsf{render}[\mathsf{journal}, d]) \cup \perp)^*,$$

where d is the device that the client uses to access the journal. Assuming the license is labeled n, the property that the client is not obligated to access the journal immediately after paying the fee can be written as:

$$\mathsf{pay}_n[\mathsf{fee}] \Rightarrow \bigcirc(\neg O\mathsf{render}_n[\mathsf{journal}, d]).$$

The specification that the client doesn't violate the license can be written as the family of formulas:

$$n : \ell \Rightarrow \Box[(\alpha \Rightarrow (P\alpha)) \wedge ((O\alpha) \Rightarrow \alpha)],$$

where $\alpha \in \{\mathsf{pay}_n[\mathsf{fee}], \mathsf{render}_n[\mathsf{journal}, d], \perp_n\}$. In other words, the client only does legitimate actions and does every action that is required by the license once it is issued. As a final example, we can write that, during one time period, the client pays \$1500 on the mortgage m, but doesn't pay the journal fee as:

$$\mathsf{pay}_m[1500] \wedge \overline{\mathsf{pay}_n[\mathsf{fee}]}.$$

2.2 Semantics

To formalize the intuitions given above, we base our semantics on the notion of a run. When defining a run, we make the standard assumption that time is discrete and can, in fact, be represented using nonnegative integers. A run r associates each time t with a pair (L, A), where L is the set of named licenses issued at that time (a named license is a pair (n, ℓ) of a name n and a license ℓ), and A is a function giving, for each license name n, an action $A(n)$ performed by the client at that time (or $\perp$ if no action was performed with respect to n). Formally, a run is a function $r : \mathbb{N} \longrightarrow \wp(Names \times Lic) \times Act^{Names}$ such that no name is paired with more than one license throughout the entire run. Recall that Act^{Names} is the set of all functions from $Names$ to Act. Our approach imposes the restriction that, at most, one action per time *per named license* can occur. We do not need this limitation, but it simplifies the exposition. In essence, we are trading the ability to handle the class of licenses where a client must do multiple actions simultaneously for a simple definition of a license where concurrent actions are not handled. For notational convenience, given a run r and time t with $r(t) = (L, A)$, we define $lic(r, t)$ to be the set of named licenses issued in run r at time t, that is, $lic(r, t) = L$; similarly, we define $act(r, t)$ to be the set of action and license name pairs performed in run r at time t, that is, $act(r, t) = \{(A(n), n) \ : \ n \in Names\}$.

[1] Recall that $\Box\varphi$ means "φ holds now and at all future times", $\bigcirc\varphi$ means "φ holds at the next time", and $\varphi_1 \mathcal{U} \varphi_2$ means "φ_2 eventually holds and, until it does, φ_1 holds".

Finally, we say that a license $(n : \ell)$ is *active* at time t in run r if there exists a time $t' \leq t$ such that $(n : \ell) \in lic(r, t')$

While a run captures the client's actions, an interpretation states what is permitted. Formally, a permission interpretation P is a function $P : \mathbb{N} \longrightarrow \wp(Act \times Names)$ that is used to give a meaning to permissions. Intuitively, if $(a, n) \in P(t)$ then at time t, the client is permitted to perform action a with respect to license name n. In other words, the client is allowed to do an a_n action.

We want the interpretation of permissions to match the permissions implied by the run. To define this requirement formally, we first give a mapping that relates licenses to action sequences. We then use this mapping to find the permission interpretation that permits an action if and only if the run implies the permission.

Following the lead of Gunter *et al.* [2001], we associate each license with a set of traces. In our discussion, a *trace* refers to a sequence of actions.[2] The notation $s_1 \cdot s_2$ denotes the concatenation of two sequences of actions s_1 and s_2 where $s_1 \cdot s_2 = s_1$ if s_1 is infinite. A trace s_1 is said to be a *prefix* of trace s_2 if there is some trace s such that $s_1 \cdot s = s_2$.

We construct a function $\mathcal{L}[\![\ell]\!]$ by induction on the structure of a given license ℓ:

$$\begin{array}{rcl}
\mathcal{L}[\![a]\!] & = & \{a\} \\
\mathcal{L}[\![\ell_1\ \ell_2]\!] & = & \{s_1 \cdot s_2 \ :\ s_1 \in \mathcal{L}[\![\ell_1]\!] \text{ and } s_2 \in \mathcal{L}[\![\ell_2]\!]\} \\
\mathcal{L}[\![\ell_1 \cup \ell_2]\!] & = & \mathcal{L}[\![\ell_1]\!] \cup \mathcal{L}[\![\ell_2]\!] \\
\mathcal{L}[\![\ell^*]\!] & = & \bigcup_{n \geq 0} \{s_1 \cdot \ldots \cdot s_n \ :\ s_i \in \mathcal{L}[\![\ell]\!]\}.
\end{array}$$

The function $\mathcal{L}[\![\ell]\!]$ gives the set of traces allowed by the license. We define the function $\mathcal{I}[\![\ell]\!]$ to provide the infinitary version of the sequences corresponding to ℓ, by essentially appending infinitely many $\perp$ actions at the end of each sequence. Formally, $\mathcal{I}[\![\ell]\!] = \{s \cdot \perp^\infty \ :\ s \in \mathcal{L}[\![\ell]\!]\}$. Finally, a sequence of action s is said to be *viable* for ℓ if s is a prefix of some trace in $\mathcal{I}[\![\ell]\!]$.

We are now ready to define the interpretation P_r corresponding to run r. Given a named license (n, ℓ) issued at time t_1 in a run r, the *action-sequence* of n up to time t_2, denoted $r[n, t_2]$, is the sequence $a_0 a_1 \cdots a_{t_2 - t_1 - 1}$ such that:

$$a_i = \begin{cases} a & \text{if } (a, n) \in act(r, t_1 + i) \\ \perp & \text{otherwise.} \end{cases}$$

Since we restricted a run to only allow one action per license per time unit, the notion of an action-sequence is well-defined. The interpretation P_r corresponding to a run r is defined as follows. For all times $t \geq 0$, $P_r(t)$ is the smallest set such that for all license names $n \in Names$ and actions $a \in Act$, $(\perp, n) \in P(t)$ if the license (n, ℓ) is not active and $(a, n) \in P(t)$ if the license is active and $r[n, t] \cdot a$ is viable for ℓ.

To understand the meaning of an action expression, α, we need a way to associate it with name-action pairs. We do this by defining a mapping $\mathcal{A}[\![\alpha]\!]$ from expressions to sets of pairs. Clearly, an action expression (a, n) should be mapped to the pair (a, n). The complement action $(\overline{a}, n)$ is mapped to the set of actions different from a, but associated with the same license name n. Formally,

$$\begin{array}{rcl}
\mathcal{A}[\![(a, n)]\!] & = & \{(a, n)\} \\
\mathcal{A}[\![(\overline{a}, n)]\!] & = & \{(b, n) \mid b \neq a\}.
\end{array}$$

[2] Gunter *et al.* use the term *reality* for this concept, although their formal definition is different.

Contrary to intuition, we do not associate the complement of a name-action pair with the largest set of name action pairs that does not include it. This mapping has unfortunate consequences, because it ignores the intuitive independence between licenses. For example, it allows us to deduce that the client can do any action with respect to any license other than the mortgage, if the client is permitted to not make a mortgage payment. Statements concerning one set of licenses should not be used to deduce anything about any other license.

As an example of our approach, recall the situation in which the client pays \$1500 on the mortgage, but doesn't pay the journal fee. The action expressions α_1 and α_2 used to express these actions are $\mathsf{pay}_m[1500]$ and $\neg\mathsf{pay}_n[\text{fee}]$, respectively. Applying the above definition, $\mathcal{A}[\![\alpha_1]\!] = \{(\mathsf{pay}[1500], m)\}$, and $\mathcal{A}[\![\alpha_2]\!] = \{(a, n) \ : \ a \neq \mathsf{pay}[\text{fee}]\}$. Hence, the actions α_1 and α_2 mean that "the client is paying \$1500 with respect to m and doing some action other than paying the fee with respect to n".

We now define what it means for a formula φ to be true (or satisfied) at a run r at time t, written $r, t \models \varphi$, by induction on the structure of φ:

$$r, t \models n : \ell \quad \text{if } (n, \ell) \in lic(r, t),$$

$$r, t \models \alpha \quad \text{if } \exists (a, n) \in \mathcal{A}[\![\alpha]\!] \text{ s.t. } (a, n) \in act(r, t),$$

$$r, t \models P\alpha \quad \text{if } \exists (a, n) \in \mathcal{A}[\![\alpha]\!] \text{ s.t. } (a, n) \in P_r(t),$$

$$r, t \models \bigcirc\varphi \quad \text{if } r, t+1 \models \varphi,$$

$$r, t \models \Box\varphi \quad \text{if for all } t' \geq t,\ r, t' \models \varphi,$$

$$r, t \models \varphi\mathcal{U}\psi \quad \text{if } \exists t' \geq t \text{ s.t. } r, t' \models \psi \text{ and } r, t'' \models \varphi \text{ for all } t'' \text{ with } t' > t'' \geq t,$$

$$r, t \models \neg\varphi \quad \text{if } r, t \not\models \varphi,$$

$$r, t \models \varphi \wedge \psi \quad \text{if } r, t \models \varphi \text{ and } r, t \models \psi.$$

If a formula φ is true at all times in a run r, we say φ is *valid* in r and write $r \models \varphi$. If φ is valid in all runs r, we simply say φ is valid and write $\models \varphi$. [3]

Various properties of permission (P) and obligation ($\neg P(\overline{a}, n)$) follow from the above semantics. In particular, we can see that $O(a, n)$ is true in a run r at time t if and only if (a, n) is the only action-name pair in $P_r(t)$. In other words, an action is obligated if and only if it is the only permitted action. This is a consequence of the following proposition:

Proposition 2.1: *For all action expressions* (a, n)*, the formula* $P(a, n) \vee P(\overline{a}, n)$ *is valid.*

[3] In an earlier version of this paper [Pucella and Weissman 2002], we considered two related semantics for formulas, in the spirit of the logics presented by Halpern and van der Meyden [2001a, 2001b]. The first semantics, called the open semantics, was defined with respect to an arbitrary interpretation P. The second semantics, called the closed semantics, was defined from the open semantics by taking the minimal interpretation, as we do in this paper. Intuitively, the closed semantics assumes that the run contains all the information relevant to interpret the formulas. This is often referred to as the *closed-world assumption*. In other words, if a permission is not implied by the run, then it is not permitted. In contrast, the open semantics admits that the run may not encode all the information, and therefore one cannot infer that an action is not permitted simply because it is not implied by the run.

Hence, if $P(\overline{a}, n)$ is not true at a point, $P(a, n)$ must be true. Another consequence of the above proposition is that $O(a, n) \Rightarrow P(a, n)$ is valid. These properties show that our operators P and O, although defined exclusively from the traces of the licenses issues in a run, satisfy some of the classical properties of deontic logic operators, as given for instance in [Follesdal and Hilpinen 1981]. These properties are a consequence of our prescribed semantics and, as such, suggest a certain deontic interpretation. In particular, the validity of $O(a, n) \Rightarrow P(a, n)$ indicates that obligation should be read as "must" and not as "ought". It also reflects the fact that we cannot express conflicting prohibitions and obligations in our framework.

2.3 Encoding finite runs and licenses

In this section, we show that any run can be "encoded" as a formula in our logic, provided that the run is finite. By finite, we intuitively mean that nothing happens after a given time, and each time instant, only finitely many licenses are issued and non-$\perp$ actions are performed. Formally, a run r is finite if there exists a natural number t_f such that :

- for all $t \leq t_f$, $lic(r, t)$ is finite,
- for all $t \leq t_f$, $\{n : (a, n) \in act(r, t), a \neq \perp\}$ is finite,
- for all $t > t_f$, $lic(r, t) = \emptyset$, and
- for all $t > t_f$, $(a, n) \in act(r, t)$ implies $a = \perp$.

For convenience, we write $\bigcirc^k \varphi$ for the formula $\bigcirc \cdots \bigcirc \varphi$ that has k occurences of the $\bigcirc$ operator before φ. Given a finite run r, define N_r to be the set of license names issued in r. Formally, $N_r = \{n : \exists t, \ell.(n, \ell) \in lic(r, t)\}$. Define

$$\psi_r = \psi_0 \wedge \bigcirc \psi_1 \wedge \bigcirc^2 \psi_2 \wedge \cdots \wedge \bigcirc^{t_f} \psi_{t_f} \wedge \bigcirc^{t_f+1} \Box \psi_e,$$

where t_f is the last time "something happened" in the run, ψ_e is $\bigwedge_{n \in N_r} (\perp, n)$, and ψ_t, which encodes the state of the run at time t, is:

$$\psi_t = \bigwedge_{\substack{(a,n) \in act(r,t) \\ n \in N_r}} (a, n) \wedge \bigwedge_{(n,\ell) \in lic(r,t)} n : \ell.$$

Finally, let N_φ be the set of license names appearing in formula φ, defined in the obvious way. The following proposition formalizes the fact that ψ_r captures the important aspects of the run r.

Proposition 2.2: *If r is a finite run and $N_\varphi \subseteq N_r$, then $r, t \models \varphi$ iff $\models \psi_r \Rightarrow \bigcirc^t \varphi$.*

It is interesting to note that ψ_r does not specify explicitly the permissions implied by the run. Intuitively, this is because the information encoded in ψ_r is sufficient for the permissions to be uniquely determined. To formalize this intuition, we show the more general result that issuing a license results in the client's actions implying a particular set of permissions.

We use some notation from the theory of regular languages to formalize the general result. Specifically, we let ϵ represent the empty action sequence and we extend the set of licenses to include 0 and 1 where $\mathcal{L}[\![0]\!] = \emptyset$ and $\mathcal{L}[\![1]\!] = \{\epsilon\}$. We also define complementary functions $S(\ell)$ and $D_a(\ell)$ where ℓ is a regular expression. For any action sequence $a_0, a_1, \ldots, a_n \in \mathcal{L}[\![\ell]\!]$,

$S(\ell)$ is the set of actions containing a_0 and $D_{a_0}(\ell)$ is a regular expression such that $a_1, \ldots, a_n \in \mathcal{L}[\![D_{a_0}(\ell)]\!]$. Formally, $S(0) = \emptyset$, $S(1) = \emptyset$, $S(a) = \{a\}$, $S(\ell_1\ell_2) = S(\ell_1)$ if $\epsilon \notin \mathcal{L}[\![\ell_1]\!]$ and $S(\ell_1) \cup S(\ell_2)$ otherwise, $S(\ell_1 \cup \ell_2) = S(\ell_1) \cup S(\ell_2)$, and $S(\ell^*) = S(\ell)$. $D_a(\ell)$ is called the Brzozowski derivative of ℓ with respect to a [Brzozowski 1964]. Its formal definition is: $D_a(a) = 1$, $D_a(b) = 0$, $D_a(\ell_1\ell_2) = D_a(\ell_1)\ell_2$ if $\epsilon \notin \mathcal{L}[\![\ell_1]\!]$ and $(D_a(\ell_1)\ell_2) \cup (D_a(\ell_2))$ otherwise, $D_a(\ell_1 \cup \ell_2) = D_a(\ell_1) \cup D_a(\ell_2)$, and $D_a(\ell^*) = D_a(\ell)\ell$.

Given these definitions, we inductively define a family of formulas for each named license (n, ℓ). For any action sequence $a_0 a_1 \cdots a_n \in \mathcal{L}[\![\ell]\!]$, the formulas say that a_0 is permitted and if the client does the action sequence $a_0 \cdots a_{i-1}$, then the client is permitted to do a_i in i time steps. Formally:

$$\begin{aligned} \varphi^0_{n,\ell} &= \bigwedge_{a \in S(\ell)} P(a, n) \\ \varphi^{i+1}_{n,\ell} &= \bigwedge_{a \in S(\ell)} \Big(P(a, n) \wedge \big((a, n) \Rightarrow \bigcirc \varphi^i_{n, D_a(\ell)} \big) \Big). \end{aligned}$$

The following proposition formalizes the intuition that by issuing a license, we force the client's actions to imply a particular set of permissions.

Proposition 2.3: *For any license ℓ, the formulas $n : \ell \Rightarrow \varphi^i_{n,\ell}$ are valid, for $i = 0, 1, 2, \ldots$.*

Hence, if the formula ψ_r represents the finite run r in the sense of Proposition 2.2, then every named license (n, ℓ) issued in run r will imply the formulas $\varphi^i_{n,\ell}$, as per Proposition 2.3. Because the conjunction of the actions specified in ψ_r and the formula $\varphi^i_{n,\ell}$ implies the permissions that hold for run r for i time steps, Proposition 2.2 is true even though ψ_r does not specify permissions explicitly.

3 Satisfiability and verification

In this section, we examine the complexity of reasoning using $\mathcal{L}^{lic}$ and discuss a technique for automatically checking if a client behavior specification is satisfied in a given run. As we mentionned in the introduction, we are fundamentally interested in two classes of questions does a set of licenses have certain properties and does a client's actions with respect to a set of licenses meet particular specifications. The first question can be rephrased as "does a set of licenses imply a property, regardless of what the client does, which licences are issued, and when the licenses are issued?". In other words, the first question corresponds to asking if a formula in our logic is valid (i.e., true in all runs). The second question can be rephrased as "does a specification hold for a given sequence of client actions and licenses issued?" In other words, the second question corresponds to asking if a formula in our logic is true in a given run.

To answer the first question, we investigate the complexity of our satisfiability problem (i.e. the problem of determining for any given $\mathcal{L}^{lic}$ formula φ if there exists a run r and a time t such that $r, t \models \varphi$). We can reduce the satisfiability problem for our logic to the satisfiability problem for a "simpler" logic, Linear Temporal Logic (LTL), which is well-known in the formal verification community. LTL is essentially a propositional logic with temporal operators. To distinguish the

LTL operators from the temporal operators in $\mathcal{L}^{lic}$, we use CTL syntax for LTL. Specifically, an LTL formula F is defined as:

$$F ::= p \mid F_1 \wedge F_2 \mid \neg F \mid \mathbf{X}F \mid \mathbf{G}F \mid F_1 \mathbf{U} F_2$$

where p is a primitive proposition, $\mathbf{X}F$ means that F holds at the next time, $\mathbf{G}F$ means that F holds now and at all future times, and $F_1\mathbf{U}F_2$ means that F_2 eventually holds and, until it does, F_1 holds. Models for LTL are linear structures of the form $M = (S, L)$, where $S = \{s_0, s_1, s_2, \dots\}$ is a set of states and L assigns to every state in S the primitive propositions that are true in that state. The definition of the satisfiability of an LTL formula F in a linear structure M at state s, written $M, s \models_L F$, is straightforward. We refer to [Clarke, Grumberg, and Peled 1999] for more detail. The key property of LTL that we will use is that the satisfiability problem for LTL is PSPACE-complete [Sistla and Clarke 1985].

It is straightforward to encode a formula F in LTL as a formula φ in $\mathcal{L}^{lic}$ in such a way that F is satisfiable if and only if φ is satisfiable. Therefore, the satisfiability problem for $\mathcal{L}^{lic}$ is PSPACE-hard. What is more interesting is that there is a polynomial reduction from the satisfiability problem for $\mathcal{L}^{lic}$ to the satisfiability problem for LTL. At the heart of this reduction is a way to encode our logic into LTL.

The first step of the reduction is to show that if a formula φ is satisfiable in $\mathcal{L}^{lic}$, then it can be translated into a satisfiable formula φ^T in LTL. We will do this directly, by showing that we can in fact transform the run r in which φ is true into a linear structure M_r in which φ^T is true. Let Φ_0 be the set of primitive propositions that we will use in our formula encoding, inclduing primitive propositions $\mathsf{issued}(n, \ell)$ for every name n and license ℓ, and $\mathsf{done}(a, n)$, $\mathsf{permitted}(a, n)$ and $\mathsf{obligated}(a, n)$ for each action a and name n.

Given a run r, we construct a linear model $M_r = (S, L)$ where $S = \{s_0, s_1, s_2, \dots\}$. For each state s_t, which corresponds to the run at time t, $L(s_t)$ is defined as the smallest set such that:

- if $(n, \ell) \in lic(r, t)$, then $\mathsf{issued}(n, \ell) \in L(s_t)$,
- if $(a, n) \in act(r, t)$, then $\mathsf{done}(a, n) \in L(s_t)$,
- if $(a, n) \in P_r(t)$, then $\mathsf{permitted}(a, n) \in L(s_t)$,
- if $(a, n) \in P_r(t)$ is the only action associated with license name n in $P_r(t)$, then $\mathsf{obligated}(a, n) \in L(s_t)$.

Given this structure M_r, it should be clear how to translate a $\mathcal{L}^{lic}$ formula φ true in r into a formula φ^T true in M_r. In particular, the following translation works:

- $(n : \ell)^T = \mathsf{issued}(n, \ell)$.
- $(a, n)^T = \mathsf{done}(a, n)$ and $(\overline{a}, n)^T = \neg\mathsf{done}(a, n)$.
- $(P(a, n))^T = \mathsf{permitted}(a, n)$ and $(P(\overline{a}, n))^T = \neg\mathsf{obligated}(a, n)$.
- $(\varphi_1 \wedge \varphi_2)^T = \varphi_1^T \wedge \varphi_2^T$ and $(\neg\varphi)^T = \neg\varphi^T$.
- $(\bigcirc\varphi)^T = \mathbf{X}\varphi^T$, $(\Box\varphi)^T = \mathbf{G}\varphi^T$, and $(\varphi_1 \mathcal{U} \varphi_2)^T = \varphi_1^T \mathbf{U} \varphi_2^T$.

It is straightforward to see that the above translations preserve the truth of the formula. In fact, something stronger holds, which will be useful later in this section:

Proposition 3.1: $r, t \models \varphi$ *iff* $M_r, s_t \models_L \varphi^T$.

This means that if φ is satisfiable in our logic, then φ^T is satisfiable in LTL. However, the converse does not hold. In particular, φ^T may be satisfiable in an LTL structure that does not correspond to any run. We somehow need a way to restrict the LTL structures considered, to ensure that they correspond to runs in $\mathcal{L}^{lic}$. Intuitively, we need to account in LTL for the notions that are implicit in the $\mathcal{L}^{lic}$ semantics. In particular, we must enforce our requirements that two actions are never done for the same license at the same time, two licenses are never labeled with the same name, an obligation implies exactly one action is permitted for the license, a client is only permitted to do actions other than $\perp$ for active licenses, and issuing a license implies various facts as discussed in Section 2.3. It is easy to state all but the last of these in LTL.

Since we will only need to satisfy the above restrictions as they pertain to a given formula φ, we enforce those restrictions over the actions, license names, and licenses appearing in φ. In general, let A be a finite set of actions, N be a finite set of license names, and L be a finite set of named licenses. The restriction that at most one action is done per license name per time is expressed by the following LTL formula $\mathsf{Done}_{A,N}$:

$$\mathbf{G} \bigwedge_{\substack{a \in A \\ n \in N}} \left(\mathsf{done}(a, n) \Rightarrow \bigwedge_{\substack{a' \in A \\ a' \neq a}} \neg \left(\mathsf{done}(a', n)\right) \right).$$

The restriction that a license name in N is never associated with more than one license in L is expressed by the LTL formula Issued_L:

$$\mathbf{G} \bigwedge_{(n,\ell) \in L} \left(\mathsf{issued}(n, \ell) \Rightarrow \bigwedge_{\substack{(n',\ell') \in L \\ n' = n}} \mathbf{G} \neg (\mathsf{issued}(n', \ell')) \right).$$

The restriction that obligation is an abbreviation for only being allowed to do one action with respect to a license is expressed by the LTL formula $\mathsf{Obl}_{A,N}$:

$$\mathbf{G} \bigwedge_{\substack{a \in A \\ n \in N}} \left(\begin{array}{l} \mathsf{obligated}(a, n) \Leftrightarrow \\ \quad \left(\begin{array}{l} \mathsf{permitted}(a, n) \wedge \\ \bigwedge_{\substack{a' \in A \\ a' \neq a}} \neg (\mathsf{permitted}(a', n))) \end{array} \right) \end{array} \right).$$

The restriction that a client can only do $\perp$ actions with respect to an unissued license is expressed by the LTL formula $\mathsf{Unissued}_L$:

$$\bigwedge_{(n,\ell) \in L} (\mathsf{obligated}(\perp, n)\ \mathbf{U}\ \mathsf{issued}(n, \ell)).$$

To state the consequences of issuing a named license (n, ℓ), we first construct a nondeterministic finite automaton (NFA) that accepts the same language as ℓ (when ℓ is viewed as a regular expression), and encode the transition relation of the automaton as an LTL formula. Formally, we construct the ϵ-free NFA representing ℓ as $A_n = (Q_n, \Delta_n, S_n, F_n)$ where Q_n is the set of states, Δ_n is the transition function, S_n are the start states, and F_n are the final states. For convenience, we will write $\Delta_n(q)$ for $\{a : \exists q' \in Q_n.(q, a, q') \in \Delta_n\}$ and $\Delta_n(q, a)$ for $\{q' : (q, a, q') \in \Delta_n\}$. We assume that we have primitive propositions in Φ_0 to represent the states of the automaton, namely $\mathsf{instate}(n, q)$ for all $q \in Q_n$, and a primitive proposition $\mathsf{over}(n)$ to represent the fact that we have stopped taking transitions in the automaton (for instance, because the client performed an action that was not permitted). The "effect" of taking a transition (from a finite set A of actions) in a state q of A_n can be represented by the following LTL formula $\mathsf{Trans}_{A,q}$:

$$\begin{array}{l}\mathsf{instate}(n,q) \Rightarrow \\ \left(\begin{array}{l} \bigwedge_{a \in \Delta_n(q)} (\mathsf{permitted}(a,n)) \wedge \\ \bigwedge_{a \in \Delta_n(q)} \left(\begin{array}{l}\mathsf{done}(a,n) \Rightarrow \\ \quad \bigvee_{q' \in \Delta(q,a)} \mathbf{X}(\mathsf{instate}(n,q'))\end{array}\right) \wedge \\ \bigwedge_{\substack{a \in A \\ a \notin \Delta_n(q)}} (\neg\mathsf{permitted}(a,n)) \wedge \\ \bigwedge_{\substack{a \in A \\ a \notin \Delta_n(q)}} \mathsf{done}(a,n) \Rightarrow \mathbf{X}(\mathsf{over}(n)) \end{array}\right).\end{array}$$

We also need a statement to the effect that the automaton A_n can only be in one state at any given time, or in a state satisfying over. This is expressed by the following LTL formula States:

$$\begin{array}{l}\left(\mathsf{over}(n) \Rightarrow \bigwedge_{q \in Q_n} \neg\mathsf{instate}(n,q)\right) \wedge \\ \bigwedge_{q \in Q_n} \left(\begin{array}{l}\mathsf{instate}(n,q) \Rightarrow \\ \quad \left(\neg\mathsf{over}(n) \wedge \bigwedge_{\substack{q' \in Q_n \\ q' \neq q}} \neg\mathsf{instate}(n,q')\right)\end{array}\right).\end{array}$$

The encoding of the NFA A_n is then expressed by the following LTL formula $\mathsf{NFA}_{n,\ell,A}$, which asserts the initial states of the automaton, as well as encoding all the transitions, including the transitions from the states where $\mathsf{over}(n)$ holds:

$$\begin{array}{l}\left(\bigvee_{q \in S_n} \mathsf{instate}(n,q)\right) \wedge \mathbf{G}(\mathsf{States}) \wedge \\ \mathbf{G}\left(\begin{array}{l}\bigwedge_{q \in Q_n} \mathsf{Trans}_{A,q} \wedge \\ (\mathsf{over}(n) \Rightarrow (\mathsf{obligated}(\bot,n) \wedge \mathbf{X}(\mathsf{over}(n))))\end{array}\right).\end{array}$$

The restriction that issuing a license implies the consequences described by the corresponding NFA is therefore expressed by the LTL formula $\mathsf{Lic}_{L,A}$:

$$\mathbf{G} \bigwedge_{(n,\ell) \in L} (\mathsf{issued}(n,\ell) \Rightarrow \mathsf{NFA}_{n,\ell,A}).$$

Note that the formula corresponding to the NFA construction guarantees that only the $\perp$ action is allowed for a completed license.

We now associate with every $\mathcal{L}^{lic}$ formula φ the LTL formula φ^I that captures all the implicit restrictions required for our treatment of φ. Recall from Section 2.3 that N_φ represents the set of license names appearing in φ. In a similar way, define A_φ to be the set of actions explicitely appearing in φ, and define L_φ to be the set of named licenses appearing in φ (i.e., occurrences of the $n : \ell$ formula). We take φ^I to be:

$$\mathsf{Done}_{A_\varphi,N_\varphi} \wedge \mathsf{Issued}_{L_\varphi} \wedge \mathsf{Obl}_{A_\varphi,N_\varphi} \wedge \mathsf{Unissued}_{L_\varphi} \wedge \mathsf{Lic}_{L_\varphi,A_\varphi}.$$

We can formally verify that the formula φ^I does indeed capture the implicit restrictions imposed by the semantics of $\mathcal{L}^{lic}$, as far as they pertain to formula φ. We can show:

Proposition 3.2: *If $M, s \models_L \varphi^T \wedge \varphi^I$, then there exists a run r such that $r, 0 \models \varphi$.*

Propositions 3.1 and 3.2 can be used to derive the following characterization of the complexity of the logic:

Theorem 3.3: *The satisfiability problem for $\mathcal{L}^{lic}$ is PSPACE-complete.*

Since a formula φ is valid if and only if $\neg\varphi$ is not satisfiable, a corollary of Theorem 3.3 is that determining if a formula φ of our logic is valid is also a PSPACE-complete problem.

It is much easier to answer our second question. The above discussion in fact hints at a suitable approach: we reduce the model-checking problem for our logic to one for LTL and then apply existing verification technology developed for LTL. More specifically, we translate the run (and associated minimial interpretation P_r) into a linear structure with a state for each time and atomic propositions for the licenses issued, client actions, permissions and obligation.

We restrict our attention to finite runs, as defined in Section 2.3, because we want to give an algorithm for deciding if a formula holds in a given model. (In practice, we expect to have a description of client behavior for a period of time and we want to establish permissions or obligations given that behavior; this can be modeled with a finite run.) The idea is simply to use the construction of the LTL structure M_r as given earlier, and use Proposition 3.1. The only problem is that the construction of M_r assumes that we have the permission interpretation P_r. To construct M_r efficiently, we need a way to compute P_r efficiently. For each named license (n, ℓ) (finitely many by assumption), we construct an NFA that accepts the language represented by ℓ. We associate a subset of the NFA's states with every time t after the license is issued. Specifically, the NFA's initial states are associated with the time when the license is issued. The states associated with any later time $t+1$ is the set of states that can be reached by one transition from a state associated with time t. For every time t after the license is issued, the set of permitted actions $P_{r,n}(t)$ is the set of possible transitions from the states associated with t. Finally, for any time t, $P_r(t)$ is the union of $P_{r,n}(t)$ for all licenses named n issued by time t. This procedure constructs $P_{r,n}(t)$ in polynomial time with respect to the size of the run.

Proposition 3.4: *There exists a polynomial time algorithm for computing the interpretation P_r corresponding to a finite run r.*

Combining the computation of P_r from r with the construction of the model M_r given earlier and applying known LTL model-checking techniques, model checking can be done reasonably efficiently, at least for a small specification φ:

Theorem 3.5: *There exists an algorithm for deciding if a formula φ is true in a finite run r at time t. Furthermore, the algorithm runs in polynomial time with respect to the size of the model r and in exponential time with respect to the size of the formula φ.*

A straightforward modification to the above procedure would allow us to check the validity of a formula φ in a run r (i.e., check that φ holds throughout the run).

Proposition 3.6: $r \models \varphi$ *iff* $M_r, s_0 \models_L \mathbf{G}(\varphi^T)$.

Finally, note that the model M_r is constructed without regard to the formula φ whose truth value we want to check. Therefore, we can construct M_r once and use it to model-check different formulas, each translated to LTL, against the run r.

4 Handling different license languages

In discussing our logic thus far, we have assumed that the licenses are written in a regular language. Although a regular language has the benefits of being well-known, simple, and fairly expressive, it is not difficult to imagine settings in which another license language is more appropriate. A key feature of our logic is that it can be adapted in a straight-forward way to reason about licenses that are written in any language that has trace-based semantics. To illustrate this flexibility, we will modify our logic to handle the licenses presented in Gunter *et al.* [2001].

For ease of exposition, we consider a restricted version of *DigitalRights* [Gunter, Weeks, and Wright 2001].[4] The syntax of licenses is given by the following grammar:

$$
\begin{aligned}
e \quad ::= \quad & (\mathsf{for}\ p \mid \mathsf{for}\ [\mathsf{upto}]\ m\ p) \\
& \mathsf{pay}\ x\ (\mathsf{upfront} \mid \mathsf{flatrate} \mid \mathsf{peruse}) \\
& \mathsf{for}\ W\ \mathsf{on}\ D
\end{aligned}
$$

where p is a period of time (a number of time units), x is a payment amount, W is a subset of works and D is a subset of devices. The terms $\mathsf{upfront}$, $\mathsf{flatrate}$ and peruse refer to the payment schedule. The $\mathsf{upfront}$ schedule requires payment at the beginning of the time period. The $\mathsf{flatrate}$ and peruse schedules require payment at the end of the time period. The difference between the two is that the payment for $\mathsf{flatrate}$ does not depend on the number of renderings, while the one for peruse does. If we let H be a payment schedule ($\mathsf{upfront}$, $\mathsf{flatrate}$ or peruse), then a license of the form $\mathsf{for}\ p\ \mathsf{pay}\ x\ H\ \mathsf{for}\ W\ \mathsf{on}\ D$ means that for the time period indicated by p, the client is required to pay x, according to schedule H, in order to render any of the works in W on a device in D. Instead of beginning with $\mathsf{for}\ p$, a license can start with $\mathsf{for}\ m\ p$. If the license starts with $\mathsf{for}\ m\ p$, then the body of the license is valid for m time periods of length p, but can be canceled at the end of any period.

As an example, consider the license

$$\mathsf{for}\ 3\ 100\ \mathsf{pay}\ 10.00\ \mathsf{flatrate}\ \mathsf{for}\ W\ \mathsf{on}\ D$$

[4]The original *DigitalRights* allows one to specify the time at which a client can activate a license. Roughly speaking, we could capture this in our model by adding license activation as an action.

where W is a set of works and D is a set of devices. This license allows the client to render any work in W on a device in D by paying a flat rate of 10.00 at the end of every 100 time units, for 3 such time periods.

We can incorporate this license language in our logic by replacing our syntax for licenses (ℓ) with expressions in the above language. To define the function $\mathcal{L}[\![-]\!]$, which interprets licenses as sets of traces in the semantics of our logic, we adapt the semantics of [Gunter, Weeks, and Wright 2001]. (The main difference is that we have a fixed time granularity, whereas the original semantics uses real numbers as time stamps for events.)

To build up the function $\mathcal{L}[\![-]\!]$, we first assign sets of traces to the simplest licenses, those that are valid for a single period. The set of traces that allow for a payment of x to view works from W on devices from D, for a period of p time units depends on the payment schedule. The traces for an up front schedule is defined as:

$$\begin{aligned} \mathit{UpFront}(x,p,W,D) = \{\mathsf{pay}[x]a_1 \cdots a_{p-1} \mid{} & a_i \text{ is either } \bot \text{ or } \mathsf{render}[w,d] \\ & \text{for some } w \in W \text{ and } d \in D\}. \end{aligned}$$

The traces for a flat rate schedule is defined as:

$$\begin{aligned} \mathit{FlatRate}(x,p,W,D) = \{a_0 \cdots a_{p-2}\mathsf{pay}[x] \mid{} & a_i \text{ is either } \bot \text{ or } \mathsf{render}[w,d] \\ & \text{for some } w \in W \text{ and } d \in D\}. \end{aligned}$$

The set of traces for a per use schedule is defined as:

$$\begin{aligned} \mathit{PerUse}(x,p,W,D) = \{a_0 \cdots a_{p-2}\mathsf{pay}[nx] \mid{} & a_i \text{ is either } \bot \text{ or } \mathsf{render}[w,d] \\ & \text{for some } w \in W \text{ and } d \in D, \\ & \text{and } n = |\{a_i \mid a_i \neq \bot\}|\}. \end{aligned}$$

Given two sets of traces S_1 and S_2, we define $S_1 \cdot S_2$ as the set $\{s_1 \cdot s_2 \mid s_1 \in S_1, s_2 \in S_2\}$. In other words, $S_1 \cdot S_2$ is the set of all concatenation of traces from S_1 and S_2. We write S^n for $\underbrace{S \cdot S \cdot \ldots \cdot S}_{n}$.

Using the above definitions, we define the function $\mathcal{L}[\![-]\!]$ as:

$$\begin{aligned} \mathcal{L}[\![\mathsf{for}\ p\ z]\!] &= \mathcal{M}[\![z]\!](p) \\ \mathcal{L}[\![\mathsf{for}\ m\ p\ z]\!] &= (\mathcal{M}[\![z]\!](p))^m \\ \mathcal{L}[\![\mathsf{for\ upto}\ m\ p\ z]\!] &= \bigcup_{n=0}^{m} (\mathcal{M}[\![z]\!](p))^n, \end{aligned}$$

where $\mathcal{M}[\![-]\!]$ generates the traces for a single time period:

$$\begin{aligned} \mathcal{M}[\![\mathsf{pay}\ x\ \mathsf{upfront\ for}\ W\ \mathsf{on}\ D]\!](p) &= \mathit{UpFront}(x,p,W,D) \\ \mathcal{M}[\![\mathsf{pay}\ x\ \mathsf{flatrate\ for}\ W\ \mathsf{on}\ D]\!](p) &= \mathit{FlatRate}(x,p,W,D) \\ \mathcal{M}[\![\mathsf{pay}\ x\ \mathsf{peruse\ for}\ W\ \mathsf{on}\ D]\!](p) &= \mathit{PerUse}(x,p,W,D). \end{aligned}$$

As expected, the semantics of the logic defined in Section 2 carries over verbatim with the above changes.

The *DigitalRights* language given above is not more expressive than the regular one that we introduced in Section 2. It is easy to see that for any license e in *DigitalRights*, the set of traces $\mathcal{L}[\![e]\!]$

can be expressed by a regular language. Because the sets $UpFront(x, p, W, D)$, $FlatRate(x, p, W, D)$, and $PerUse(x, p, W, D)$ are finite for any p, x, W and D, it is trivial to express them using a regular language. The concatenation operation $S_1 \cdot S_2$ preserves regularity, as does union, therefore it is possible to express any license expressed in *DigitalRights* as a regular one. There are, however, advantages to using the *DigitalRights* language. The translation of a *DigitalRights* license yields a large regular expression that may be significantly less efficient to verify than the original license. Another benefit is that the *DigitialRights* language is easier to understand.

It should be noted that every license language is not necessarily subsumed by the language of regular expressions. To see this, consider a license in some license language that can be canceled whenever the number of renderings equals the number of payments. The set of traces corresponding to such a license is not regular, by a well-known result from formal language theory (see for instance [Hopcroft and Ullman 1969]). Therefore, any language that can be used to state this license is not equivalent to any sublanguage of the regular expressions.

5 Related work

The inspiration for our work comes from the field of program verification, where one finds logics such as Hoare Logic [Hoare 1969] and Dynamic Logic [Harel, Kozen, and Tiuryn 2000] to reason about properties of programs. Our logic is similar to those, in the sense that our formulas contain explicit licenses, in much the same way that theirs contain explicit programs. Logics of this type are often referred to as *exogenous*. In contrast, *endogenous* logics do not explicitly mention programs; to analyze a program with such a logic, one builds a model for that specific program, and uses the logic to analyze the model. One advantage of using an exogenous logic is that it allows the behavior of two programs to be compared within the logic. In our case, it allows us to compare the effect of different licenses within the logic. An endogenous logic, however, permits more efficient verification procedures. To get this benefit, our verification procedures in Section 3 essentially convert formulas from our logic into formulas of an endogenous logic, viz. temporal logic.

Although our logic is an exogenous logic inspired by Dynamic Logic, its models are quite different. In Dynamic Logic, programs guide the state transitions in the model. Licenses, on the other hand, do not affect states. Instead, they are used to specify permissions and obligations. The models of our logic are primarily influenced by the work of Halpern and van der Meyden [2001b] on formalizing SPKI [Ellison, Frantz, Lampson, Rivest, Thomas, and Ylonen 1999]. SPKI is used to account for access rights based on certificates received. Similarly, we base the right to do actions on the licenses received. In fact, we could imagine licenses being implemented with SPKI certificates.

Permissions and obligations are key concepts in our approach. These notions are typically studied in the philosophical literature under the heading of *deontic logic* [Meyer and Wieringa 1993]. Early accounts of deontic logic failed to differentiate between actions and assertions, leading to many paradoxical and counterintuitive propositions (see for instance [Follesdal and Hilpinen 1981]). The idea of separating actions from assertions has lead to a recasting of deontic logic as a variant of Dynamic Logic [Meyer 1988; Meyden 1990]. Models for deontic dynamic logics specify explicitly either which states represent the violation of an obligation or a permission or which transitions are permitted or forbidden. In [Meyer 1988], a special formula V is introduced in the logic, and any state that satisfies V is deemed a violation. Intuitively, an action a is permitted in a state if it is possible to reach a state via a where V does not hold. Conversely, an action is obligatory if performing any other action leads to a state where V holds. In [Meyden 1990], it is the transitions between states that are deemed permitted or forbidden. $\mathcal{L}^{lic}$ is different from these approaches, because we derive our permissions and obligations from the licenses issued in the run. This indirection means that we do not have to explicitly model the permissions and obligations. In addition, we can easily change the model to account for different licenses.

Finally, deontic logic has been used to reason about contracts. This is intriguing, because a license can be viewed as a restricted form of contract. Research in this direction includes work by Lee [1988], which focuses on developing a logical language based on predicate logic with temporal operators. Deontic operators are handled using a specific predicate to represent a violation (in this context, defaulting on a contract). Unfortunately, the logic is not meant to reason about contracts written in some language. Instead, the models for the logic *represent* the contracts to be analyzed. In other words, for each contract that he wants to study, Lee builds a specific model encoding violations at the appropriate states.

6 Conclusion

In this paper we have introduced a framework for precisely stating and rigorously proving properties of licenses. We also have illustrated how our logic can be modified to reason about licenses that are written in any language with a trace-based semantics. This flexibility provides us with a common ground in which to compare different rights languages with trace-based semantics. We intend to report on these comparisons in the future. While useful in its own right, the logic is a simple foundation on which more expressive rights management logics can be built. For example, the logic can be modified in a straightforward manner to support multiple clients and multiple providers. Multiple providers is an especially interesting case, because it allows us to study the management of licensing rights, the rights required for one provider to legitimately offer another provider's work to a client. We plan to examine various extension in the near future. There remain interesting questions about the foundation of $\mathcal{L}^{lic}$, such as axiomatizations for the logic. Finally, as mentioned previously, our operators P and O have a distinctly deontic flavor. It would be interesting to establish a correspondence between our approach and existing deontic frameworks, in particular deontic logics of actions [Khosla and Maibaum 1987; Meyer 1988; Meyden 1990].

Acknowledgments

We would like to thank Joe Halpern, Carl Lagoze and Sandy Payette for their helpful comments. Joshua Guttman and the CSFW anonymous referees made suggestions that greatly improved the presentation. Support for this work came from the ONR under grants N00014-00-1-0341 and N00014-01-1-0511, from the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and from the NSF under grant IIS-9905955 (Project Prism).

A Proofs

Proposition 2.1: *For all action expressions (a, n), the formula $P(a, n) \vee P(\overline{a}, n)$ is valid.*

Proof: The validity of this formula is a consequence of the fact that $P_r(t)$ contains at least one action corresponding to every license name n. Given a run r and a time t, and consider the action expression (a, n). We know there must exist an action-name pair (b, n) in $P_r(t)$. Two cases arise. If $a = b$, then (a, n) is in both $\mathcal{A}[\![(a, n)]\!]$ and $P_r(t)$, and thus $r, t \models P(a, n)$. If $a \neq b$, then (b, a) is in both $\mathcal{A}[\![(\overline{a}, n)]\!]$ and $P_r(t)$, and thus $r, t \models P(\overline{a}, n)$. Therefore, we have $r, t \models P(a, n) \vee P(\overline{a}, n)$. Since the above holds for all r and t, $\models P(a, n) \vee P(\overline{a}, n)$. ∎

Proposition 2.2: *If r is a finite run and $N_\varphi \subseteq N_r$, then $r, t \models \varphi$ iff $\models \psi_r \Rightarrow \bigcirc^t \varphi$.*

To simplify the proof, we introduce the following notation. Given runs r, r', times t, t', and a subset N of $Names$, define $(r, t) \leq_N (r', t')$ if for all $i \geq 0$, $lic(r, t + i) \subseteq lic(r', t' + i)$ and $(act(r, t + i) \cap (Act \times N)) = (act(r', t' + i) \cap (Act \times N))$. Intuitively, $(r, t) \leq_N (r', t')$ if every license issued by r (starting at time t) is also issued in r' (starting at time t'), and moreover the two runs agree on the actions corresponding to license names in N. The following lemmas capture the relevant properties of the $\leq_N$ relation. Recall that N_φ is the set of license names appearing in formula φ.

Lemma A.1: *For any φ such that $N_\varphi \subseteq N_r$, if $(r, 0) \leq_{N_r} (r', t')$, then $r, i \models \varphi$ iff $r', t' + i \models \varphi$ for all $i \geq 0$.*

Proof: By induction on the structure of φ. We prove the nontrivial cases here. Consider $\varphi = n : \ell$. If $r, i \models n : \ell$, then $(n, \ell) \in lic(r, i) \subseteq lic(r', t' + i)$, and hence $r', t' + i \models n : \ell$. Conversely, if $r', t' + i \models n : \ell$, then since $N_\varphi \subseteq N_r$, license name n must appear in r, and by definition of $(r, 0) \leq_{N_r} (r', t')$ and the fact that license names can be associated with only one license in a run, it must be the case that $(n, \ell) \in lic(r, i)$. Hence, $r, i \models n : \ell$. The cases for (a, n) and $(\overline{a}, n)$ follow from r and r' agreeing on the actions for license names $n \in N_\varphi \subseteq N_r$. For $P(a, n)$ and $P(\overline{a}, n)$, because r and r' agree on the licenses issued with name $n \in N_\varphi \subseteq N_r$, and because r and r' agree on the actions pertaining to license name n, P_r and $P_{r'}$ agree on the permissions with respect to license name n, from which the result follows. The remaining cases are a straightforward application of the inductive hypothesis. ∎

Lemma A.2: $r', t' \models \psi_r$ *iff* $(r, 0) \leq_{N_r} (r', t')$.

Proof: We know by definition that $r', t' \models \psi_r$ if and only if $r', t' \models \psi_0$, $r', t' + 1 \models \psi_1$, $\ldots$, $r', t' + t_f \models \psi_{t_f}$, and $r, t' + t \models \psi_e$ for all $t > t_f$. Given the definition of $\psi_0, \ldots, \psi_{t_f}$ and ψ_e, this is equivalent to $lic(r, 0) \subseteq lic(r', t')$, $\ldots$, $lic(r, t_f) \subseteq lic(r', t' + t_f)$, $lic(r, t) = \emptyset \subseteq lic(r', t' + t)$ for $t > t_f$, and moreover $r(i)$ and $r'(t' + i)$ agree on the actions pertaining to license names $n \in N_r$ for all $i \geq 0$. This just says that $(r, 0) \leq_{N_r} (r', t')$. ∎

Proof: (Proposition 2.2) Note that $r, t \models \varphi$ iff $r, 0 \models \bigcirc^t \varphi$. Thus, it is sufficient to show that $r, 0 \models \varphi$ iff $\models \psi_r \Rightarrow \varphi$.

First, assume that $(r, 0) \models \varphi$. Let r', t' be an arbitrary run and time. If $r', t' \models \psi_r$, then by Lemma A.2, $(r, 0) \leq_{N_r} (r', t')$. Since $N_\varphi \subseteq N_r$, Lemma A.1 implies that $r', t' \models \varphi$. This establishes that $r', t' \models \psi_r \Rightarrow \varphi$. Since r', t' was arbitrary, $\models \psi_r \Rightarrow \varphi$ holds.

For the converse direction, assume that $\models \psi_r \Rightarrow \varphi$. In particular, $r, 0 \models \psi_r \Rightarrow \varphi$. Since $(r, 0) \leq_{N_r} (r, 0)$, Lemma A.2 implies that $r, 0 \models \psi_r$, and hence $r, 0 \models \varphi$. ∎

Proposition 2.3: *For any license ℓ, the formulas $n : \ell \Rightarrow \varphi^i_{n,\ell}$ are valid, for $i = 0, 1, 2, \ldots$.*

Proof: The proof relies on a suitable application of standard properties of regular expressions, and much formal symbolic manipulation. We sketch the argument here. First, extend the definition of S to handle more than a single action. Let $S^k(\ell)$ (for $k \geq 1$) be the function that returns the set of all prefixes of length k of action sequences associated with ℓ. Formally, $S^1(\ell) = S(\ell)$, and $S^{k+1} = \{a\sigma \ : \ a \in S(\ell), \sigma \in S^k(D_a(\ell))\}$.

Given this definition, we can verify that the formula $\varphi^{i+1}_{n,\ell}$ is equivalent to $\varphi^i_{n,\ell} \wedge \varphi^{i \mapsto i+1}_{n,\ell}$, where $\varphi^{i \mapsto i+1}_{n,\ell}$ is the formula

$$\bigwedge_{\substack{a_0 \cdots a_{i+1} \in \\ S^{i+2}(\ell)}} \left(\begin{array}{l} ((a_0, n) \wedge \bigcirc(a_1, n) \wedge \cdots \\ \wedge \bigcirc^i(a_i, n)) \Rightarrow \bigcirc^{i+1} P(a_{i+1}, n) \end{array} \right).$$

Let r, t be an arbitrary run and time. We show by induction that $r, t \models n : \ell \Rightarrow \varphi^i_{n,\ell}$ for all $i \geq 0$. Assume $r, t \models n : \ell$, that is, $(n, \ell) \in \mathit{lic}(r, t)$. The base case of the induction is verified by noticing that $\varphi^0_{n,\ell} = \bigwedge_{a \in S(\ell)} P(a, n)$, and by the definition of $P_r(t)$, for all $a \in S(\ell)$, $(a, n) \in P_r(t)$, so that $r, t \models P(a, n)$. The induction step follows by a similar reasoning. Assume $r, t \models \varphi^i_{n,\ell}$. Given the above equivalence, it is sufficient to show that $r, t \models \varphi^{i \mapsto i+1}_{n,\ell}$ to establish the result. For any $a_0 \cdots a_{i+1} \in S^{i+2}(\ell)$, if $r, t \models (a_0, n) \wedge \bigcirc(a_1, n) \wedge \cdots \wedge \bigcirc^i(a_i, n)$, then $r, t \models (a_0, n)$, $r, t+1 \models (a_1, n), \ldots, r, t+i \models (a_i, n)$. Since $a_0 \cdots a_i a_{i+1} \in S^{i+2}(\ell)$, it is viable for ℓ, and hence $(a_{i+1}, n) \in P_r(t+i+1)$, that is, $r, t+i+1 \models P(a_{i+1}, n)$, or $r, t \models \bigcirc^{i+1} P(a_{i+1}, n)$, as required. Since this is true for all sequences in $S^{i+2}(\ell)$, we have $r, t \models \varphi^{i \mapsto i+1}_{n,\ell}$, establishing our result. ∎

Proposition 3.1: *$r, t \models \varphi$ iff $M_r, s_t \models_L \varphi^T$.*

Proof: We prove by induction on the structure of φ that for all t, $r, t \models \varphi$ iff $M_r, s_t \models_L \varphi^T$. We give a few representative cases here, the remaining cases being similar.

Consider $\varphi = n : \ell$. For any t, we have $r, t \models n : \ell$ iff $(n, \ell) \in \mathit{lic}(r, t)$ iff $\mathsf{issued}(n, \ell) \in L(s_t)$ (by construction of $L(s_t)$) iff $M_r, s_t \models_L \mathsf{issued}(n, \ell)$.

Consider $\varphi = P(\overline{a}, n)$. For any t, we have $r, t \models P(\overline{a}, n)$ iff $(b, n) \in P_r(t)$ for some $b \neq a$ iff $\mathsf{obligated}(a, n)$ is *not* in $L(s_t)$ (since (a, n) cannot be the unique action in $P_r(t)$) iff $M_r, s_t \models_L \neg\mathsf{obligated}(a, n)$.

Consider $\varphi = \bigcirc\varphi'$. For any t, we have $r, t \models \bigcirc\varphi'$ iff $r, t+1 \models \varphi'$ iff $M_r, s_{t+1} \models_L (\varphi')^T$ (by hypothesis) iff $M_r, s_t \models_L \mathbf{X}(\varphi')^T$, and $\mathbf{X}(\varphi')^T = \varphi^T$. ∎

Proposition 3.2: *If $M, s \models_L \varphi^T \wedge \varphi^I$, then there exists a run r such that $r, 0 \models \varphi$.*

Proof: Without loss of generality, $M = (S, L)$ with $S = \{s_0, s_1, \ldots\}$, and $s = s_0$. (If not, $s = s_t$ for some t, and take $M' = (S', L)$ where $S' = \{s_t, s_{t+1}, \ldots\}$, and we can check that $M', s_0 \models_L \varphi^T \wedge \varphi^I$.) Construct the run r_M as follows: for all $t \geq 0$, $r_M(t) = (L_M(t), A_M(t))$, where $L_M(t) = \{(n, \ell) \ : \ \mathsf{issued}(n, \ell) \in L(s_t)\}$, and $A_M(t)(n) = a$ if $\mathsf{done}(a, n) \in L(s_t)$, and $A_M(t)(n) = \perp$ otherwise. This is a well-defined run, because M_r, s_0 satisfies $\mathsf{Done}_{A_\varphi, N_\varphi}$ and $\mathsf{Issued}_{L_\varphi}$. We next check that for all $t \geq 0$, $P_{r_M}(t) = \{(a, n) \ : \ \mathsf{permitted}(a, n) \in L_M(s_t)\}$. The details are routine, if tedious. Essentially, every path through the automaton encoded in $\mathsf{NFA}_{\mathsf{n},\ell,\mathsf{A}_\varphi}$ corresponds to a viable trace of the license ℓ from the point where the license is issued. A straightforward proof by induction establishes that $r_M, 0 \models \varphi$. ∎

Theorem 3.3: *The satisfiability problem for $\mathcal{L}^{lic}$ is PSPACE-complete.*

Proof: For the lower bound, we show that we can reduce the satisfiability problem for LTL to the satisfiability problem for $\mathcal{L}^{lic}$. Let F be a formula of LTL, over primitive propositions $\Phi_f = \{p_1, \ldots, p_n\}$. We first rewrite F into a formula φ_F of $\mathcal{L}^{lic}$, by picking an arbitrary non-$\perp$ action in Act (call it $\star$) and a name n_p for every $p \in \Phi_f$, and replacing every primitive proposition p in F by the action expression $(\star, n_p)$, and replacing **G**, **X**, and **U** by $\Box$, $\bigcirc$, and $\mathcal{U}$ respectively. Assume F is satisfiable in a linear structure $M = (S, L)$ at state s_i, where $S = (s_0, s_1, \ldots)$. Let r_M be the run defined by $r_M(t) = (\emptyset, A(t))$, where $A(t)$ maps name n_p to action $\star$ if $p \in L(s_t)$, and to $\perp$ otherwise, and maps all other names to $\perp$. It is easy to check that φ_F is satisfiable in r_M at time i. Similarly, if φ_F is satisfiable in a run r at time t, we can convert r into a linear structure $M_r = (S, L)$, where $p \in L(s_t)$ iff $(\star, n_p) \in act(r, t)$, and it is easy to check that F is satisfiable in M_r at state s_t. Since the satisfiability problem for LTL is PSPACE-complete, the above reduction means that the satisfiability problem for $\mathcal{L}^{lic}$ is PSPACE-hard.

For the upper bound, we show that we can reduce the satisfiability problem for $\mathcal{L}^{lic}$ to the satisfiability problem for LTL in polynomial time. In particular, we show that φ is satisfiable in $\mathcal{L}^{lic}$ iff $\varphi^T \wedge \varphi^I$ is satisfiable in LTL. Let φ be a formula satisfied in run r at time t. By Proposition 3.1, $M_r, s_t \models_L \varphi^T$. By construction, it is clear that $M_r, s_t \models_L \varphi^I$ (only one action per license per time, no two licenses with the same name ever issued, and so on). Hence, $M_r, s_t \models_L \varphi^T \wedge \varphi^I$. Conversely, assume that $\varphi^T \wedge \varphi^I$ is satisfiable in a linear structure M. By Proposition 3.2, there exists a run r such that $r, 0 \models \varphi$, i.e., φ is satisfiable in $\mathcal{L}^{lic}$. Finally, one can check that the size of the formula $\varphi^T \wedge \varphi^I$ is polynomial in the size of φ. ∎

Proposition 3.4: *There exists a polynomial time algorithm for computing the interpretation P_r corresponding to a finite run r.*

Proof: It is clearly sufficient to define P_r for non-$\perp$ actions only, by taking $\perp$ to be the default value of P_r. Let L_r be the set of named licenses issued in run r. We define, for every named license $(n, \ell) \in L_r$, a function $P_{r,n}$ that gives for every time t the set of actions permitted by the named license (n, ℓ) at time t. Clearly, we can then take $P_r(t) = \bigcup_{(n,\ell) \in L_r} P_{r,n}(t)$.

Consider a named license $(n, \ell) \in L_r$, and assume (n, ℓ) is issued at time t_0 in r. Let $A = (Q, I, \Delta, F)$ be the ϵ-free NFA corresponding to the regular expression ℓ, where Q is the set of states, I is the set of initial states, Δ is the transition relation, and F is the set of final states. We can construct A in time polynomial in the size of ℓ, using [Hromkovic, Seibert, and Wilke 1997], where $|Q|$ is linear in the size of ℓ and $|\Delta|$ is less than quadratic.

We can now define the function $P_{r,n}$. For $t < t_0$, we can take $P_{r,n}(t) = \{\perp\}$. For $t \geq t_0$, we need to take the license into consideration. First, define the sequence of sets $S_0, S_1, \ldots, S_{m-t_0}$ where m is the length of run r. These sets represents the sets of states of the NFA obtained by following the actions related to license name n prescribed by the run. Formally, define S_i inductively as:

$$\begin{aligned} S_0 &= I \\ S_{i+1} &= \{s' \,:\, (s, a, s') \in \Delta \text{ for some} \\ &\qquad\qquad s \in S_i \text{ and } (a, n) \in act(r, t_0 + i)\}. \end{aligned}$$

With these sets, we define $P_{r,n}(t_0 + i) = \bigcup_{s \in S_i} \{a \,:\, \exists s'.(s, a, s') \in \Delta\}$, that is, the set of actions that can be performed according to license ℓ starting from any of the states in S_i. One can check that the sets S_i can be constructed in polynomial time, and therefore that $P_{r,n}$, and hence P_r, can be constructed in polynomial time. ∎

Theorem 3.5: *There exists an algorithm for deciding if a formula φ is true in a finite run r at time t. Furthermore, the algorithm runs in polynomial time with respect to the size of the model r and in exponential time with respect to the size of the formula φ.*

Proof: Given a run r, we can compute P_r in polynomial time by Proposition 3.4, and construct the model M_r in time polynomial in the size of r. We can translate φ into φ^T in time polynomial in the size of the formula. We use Proposition 3.1 to reduce the problem to the model-checking problem for LTL, which can be solved in time polynomial in the size of the M_r and exponential in the size of φ (see, for instance, [Vardi 1997]). ∎

Proposition 3.6: $r \models \varphi$ *iff* $M_r, s_0 \models_L \mathbf{G}(\varphi^T)$.

Proof: By definition, $r \models \varphi$ iff for all times t, $r, t \models \varphi$. By Proposition 3.1, this holds iff for all states s_t of M_r, $M_r, s_t \models \varphi^T$, which just means that $M_r, s_0 \models \mathbf{G}\varphi^T$. ∎

References

Brzozowski, J. A. (1964). Derivatives of regular expressions. *Journal of the ACM 11*(4), 481–494.

Clarke, E. M., O. Grumberg, and D. Peled (1999). *Model Checking*. The MIT Press.

ContentGuard, Inc. (2000). XrML: Extensible rights Markup Language. Available from `http://www.xrml.org`.

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

Follesdal, D. and R. Hilpinen (1981). Deontic logic: An introduction. In R. Hilpinen and D. Reidel (Eds.), *Deontic Logic: Introductory and Systematic Readings*, pp. 1–35. Dordrecht.

Goldblatt, R. (1992). *Logics of Time and Computation*. CSLI Lecture Notes, No. 7. CSLI.

Gunter, C. A., S. T. Weeks, and A. K. Wright (2001). Models and languages for digital rights. In *Hawaii International Conference on Systems Sciences*.

Halpern, J. Y. and R. van der Meyden (2001a). A logic for SDSI's linked local name spaces. *Journal of Computer Security 9*(1,2), 47–74.

Halpern, J. Y. and R. van der Meyden (2001b). A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pp. 59–70. IEEE Computer Society Press.

Harel, D., D. Kozen, and J. Tiuryn (2000). *Dynamic Logic*. Cambridge, Massachusetts: The MIT Press.

Hoare, C. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM 12*, 576–580, 583.

Hopcroft, J. E. and J. D. Ullman (1969). *Formal languages and their relation to automata*. Reading Mass.: Addison Wesley.

Hromkovic, J., S. Seibert, and T. Wilke (1997). Translating regular expressions into small epsilon-free nondeterministic finite automata. In *Symposium on Theoretical Aspects of Computer Science*, Volume 1200 of *Lecture Notes in Computer Science*, pp. 55–66.

IPR Systems Pty Ltd (2001). Open Digital Rights Language (ODRL). Available from `http://odrl.net`.

Khosla, S. and T. S. E. Maibaum (1987). The prescription of description of state based systems. In B. Banieqbal, H. Barringer, and A. Pnueli (Eds.), *Temporal Logic in Specification*, Volume 398 of *Lecture Notes in Computer Science*, pp. 243–294. Springer.

Lee, R. M. (1988). A logic model for electronic contracting. *Decision Support Systems 4*, 27–44.

Meyden, R. van der (1990). The dynamic logic of permission. In *Proceedings of the Fifth IEEE Conference on Logic in Computer Science (LICS'90)*, pp. 72–78.

Meyer, J.-J. C. (1988). A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic 29*(1), 109–136.

Meyer, J.-J. C. and R. J. Wieringa (1993). Deontic logic: A concise overview. In J.-J. C. Meyer and R. J. Wieringa (Eds.), *Deontic Logic in Computer Science: Normative System Specification*, Chapter 1, pp. 3–16. John Wiley & Sons.

Pucella, R. and V. Weissman (2002). A logic for reasoning about digital rights (extended abstract). In *Proceedings of the Workshop on Issues in Computer Security (WITS'02)*.

Ramanujapuram, A. and P. Ram (1998). Digital content and intellectual property rights. *Dr. Dobb's Journal 23*(12), 20–27.

Sistla, A. and E. Clarke (1985). The complexity of propositional linear temporal logic. *Journal of the ACM 32*, 733–749.

Vardi, M. Y. (1997). Alternating automata: Checking truth and validity for temporal logics. In *Proceedings of the 14th International Conference on Automated Deduction*, Volume 1249 of *Lecture Notes in Computer Science*, pp. 191–206. Springer.

Wieringa, R. J. and J.-J. C. Meyer (1993). Applications of deontic logic in computer science: A concise overview. In J.-J. C. Meyer and R. J. Wieringa (Eds.), *Deontic Logic in Computer Science: Normative System Specification*, Chapter 2, pp. 17–40. John Wiley & Sons.