A Logic for Reasoning about Digital Rights

(Extended Abstract)

Riccardo Pucella
Department of Computer Science
Cornell University
Ithaca, NY 14853
riccardo@cs.cornell.edu

Vicky Weissman
Department of Computer Science
Cornell University
Ithaca, NY 14853
vickyw@cs.cornell.edu

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 x before each access to y"), 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 consumer either pays x sometime before the 4th day of the month or the consumer pays x+penalty between the 4th and the 14th."). 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.

To remove these ambiguities, Gunter *et al.* [2001] used techniques from programming language semantics [Hoare 1985] to develop a simple language and a corresponding semantics to express licenses. Their approach can be summarized as follows. The meaning of a license is a set of traces. Each trace 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. Using the language of Gunter *et al.* [2001], we can state a number of licenses precisely.

In addition to unambiguously expressing licenses, we would like to reason about them. In other words, we want to verify that a license has certain properties. Examples of properties include "is a type-B user ever allowed to view the high-resolution image" and "is a user ever allowed to access a work without, at some time, being obligated to pay." Depending on the license, each property may or may not be easy to check. This verification is important, because it increases our confidence that the license matches the informal requirements and that the informal requirements match the owner's intent.

In this paper we present a logic for reasoning about licenses. Why a logic? A logic provides us with a language in which we can write formal *specifications* and prove rigorously that these specifications are met by a set of licenses.[1] In addition, automated tools can be developed to verify, through a process called *model checking* [Clarke, Grumberg, and Peled 1999], that an application does not violate the specifications. Note that the language in which we write our specifications is distinct from the language in which we write the licenses.

In the next section we give the syntax and semantics of our logic. The design 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.) For ease of exposition, we assume 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. We discuss these modifications and other topics in Section 3.

2 The logic

2.1 Syntax

The syntax of our logic has three main categories; formulas ($\varphi, \psi, \ldots$), actions ($\alpha, \beta, \ldots$), and licenses ($\ell, m, \ldots$). Their definitions assume a set $\mathcal{N}$ of license names, a set $\mathcal{W}$ of works, and a set $\mathcal{D}$ of devices. Actions are taken from a set $\mathcal{A} = \{\mathsf{render}[w,d] \ : \ w \in \mathcal{W}, d \in \mathcal{D}\} \cup \{\mathsf{pay}[x] \ : \ x \in \mathbb{R}\} \cup \{\bot\}$, where $\bot$ represents the null or "do nothing" action. In the following formal description, $n \in \mathcal{N}$ and $a \in \mathcal{A}$.

$$\begin{array}{rcl} \varphi & ::= & n : \ell \mid \alpha \mid P\alpha \mid O\alpha \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \\ \alpha & ::= & (a, n) \mid \alpha_1 \wedge \alpha_2 \mid \neg\alpha \\ \ell & ::= & a \mid \ell_1 \, \ell_2 \mid \ell^* \mid \ell_1 \cup \ell_2 \end{array}$$

Intuitively, $n : \ell$ is a formula where ℓ is a regular expression that describes what action sequences are valid for the license labeled n. A boolean combination of primitive actions is also a formula. $P\alpha$ and $O\alpha$ indicate that the action expression α is, respectively, permitted or obligatory. The set of formulas are closed under $\wedge$, $\neg$, $\Box$, $\bigcirc$ and $\mathcal{U}$, which are standard operators from classical and temporal logic [Goldblatt 1992]. We use the obvious 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 $(\bot, n)$ is written $\bot_n$.

[1] As such, the relationship between Gunter *et al.*'s work and ours is analogous to the one between programming language semantics and Dynamic Logic [Harel, Kozen, and Tiuryn 2000].

2.2 Semantics

The semantics of the logic is based 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 integers. A run is a function $r : \mathbb{N} \longrightarrow \mathcal{P}(\mathcal{N} \times \mathcal{L}) \times \mathcal{P}(\mathcal{N} \times \mathcal{A})$. It associates each time with the licenses issued at that time and the actions performed by the client at that time. We impose the restriction that, at most, one action per time *per 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. Finally, when $r(t) = (L, A)$, we write $lic(r, t) = L$ and $act(r, t) = A$.

An interpretation π is a tuple (P, O) where P is a permission assignment and O is an obligation assignment. A permission assignment is a function $P : \mathbb{N} \longrightarrow \mathcal{P}(\mathcal{A} \times \mathcal{N})$ 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 according to license n. In other words, the client is allowed to do an a_n action. Similarly, an obligation assignment is a function $O : \mathbb{N} \longrightarrow \mathcal{P}(\mathcal{A} \times \mathcal{N})$ that is used to give a meaning to obligations. Intuitively, if $(a, n) \in O(t)$, then at time t, the client is obligated to perform action a according to license n. In other words, the client is required to do an a_n action.

To understand the meaning of an action expression, α, we need a way to associate it with name-action pairs. We do this by inductively defining a mapping $[\![\alpha]\!]_A$ from expressions to pairs. Clearly, an action expression (a, n) should be mapped to the pair (a, n). What about conjunction? Intuitively, a client who does one action and another has done both. To capture this, we define a *compound action* to be a set of name-action pairs. The intuition is that all of the actions in a compound action are done concurrently. Thus, the conjunction of two expressions corresponds to the union of two compound sets, one associated with each of the expressions. In keeping with our assumption that a client can do, at most, one action per time per license, we would like the union of the sets to have the property that no two pairs in the set refer to the same license. We will say that a compound action is *legal* if it has this property. Finally, we will associate a negated expression with the set of legal compound actions that refer to the same licenses as the expression, but do not have the same name-action pairs. By restricting negated expressions to only refer to the licenses in the expression, we maintain the intuitive independence between licenses. In other words, statements concerning one set of licenses cannot be used to deduce anything about any other license. Naturally, to take the negation of an expression, we will need to know what licenses are referred to in the expression. In order to do this, we define an associated function $[\![\alpha]\!]_N$ giving the set of licenses referred to by the compound action α. Formally,

$$
\begin{aligned}
[\![(a,n)]\!]_A &= \{\{(a,n)\}\} \\
[\![\alpha_1 \wedge \alpha_2]\!]_A &= \{a_1 \cup a_2 \ : \ a_1 \in [\![\alpha_1]\!]_A, a_2 \in [\![\alpha_2]\!]_A, a_1 \cup a_2 \text{ legal}\} \\
[\![\neg\alpha]\!]_A &= \{\underline{a} \ : \ \underline{a} \notin [\![\alpha]\!]_A, \underline{a} \text{ legal}, \{n \ : \ \exists a.(a,n) \in \underline{a}\} = [\![\alpha]\!]_N\}
\end{aligned}
$$

$$
\begin{aligned}
[\![(a,n)]\!]_N &= \{n\} \\
[\![\alpha_1 \wedge \alpha_2]\!]_N &= [\![\alpha_1]\!]_N \cup [\![\alpha_2]\!]_N \\
[\![\neg\alpha]\!]_N &= [\![\alpha]\!]_N
\end{aligned}
$$

We use the abbreviation $\alpha_1 \vee \alpha_2$ for $\neg(\neg\alpha_1 \wedge \neg\alpha_2)$. It is easy to see that the mapping for disjunction is what one would expect given the above discussion.

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

$r, \pi, t \models n : \ell$ if $(n, \ell) \in lic(r, t)$,

$r, \pi, t \models \alpha$ if $\exists \underline{a} \in [\![\alpha]\!]_A$ with $\underline{a} \subseteq act(r, t)$,

$r, \pi, t \models P\alpha$ if $\exists \underline{a} \in [\![\alpha]\!]_A$ with $\underline{a} \subseteq P(t)$,

$r, \pi, t \models O\alpha$ if $\exists \underline{a} \in [\![\alpha]\!]_A$ with $\underline{a} \subseteq O(t)$,

$r, \pi, t \models \bigcirc\varphi$ if $r, \pi, t + 1 \models \varphi$,

$r, \pi, t \models \Box\varphi$ if for all $t' \geq t$, $r, \pi, t' \models \varphi$,

$r, \pi, t \models \varphi \mathcal{U} \psi$ if $\exists t' \geq t$ such that $r, \pi, t' \models \psi$ and $r, \pi, t'' \models \varphi$ for all t'' with $t' > t'' \geq t$,

$r, \pi, t \models \neg\varphi$ if $r, \pi, t \not\models \varphi$,

$r, \pi, t \models \varphi \wedge \psi$ if $r, \pi, t \models \varphi$ and $r, \pi, t \models \psi$.

We should point out that we use the operators $\neg$ and $\wedge$ in two distinct ways: as operators acting on formulas, representing the negation and conjunction of formulas, and as operators acting on actions, representing the formation of compound actions. Since actions are themselves formulas, there is some ambiguity whether for example the operator $\wedge$ in $\alpha_1 \wedge \alpha_2$ acts on the formulas α_1 and α_2, or on the actions α_1 and α_2. That this ambiguity is not a problem, as our semantics is coherent: it will assign the same meaning to both interpretations. Specifically, we have:

Proposition 2.1: *For all actions α_1 and α_2 and for all r,π,t, the following holds:*

- $r, \pi, t \not\models \alpha_1$ *iff there exists* $\underline{a} \in [\![\neg\alpha_1]\!]_A$ *with* $\underline{a} \subseteq act(r, t)$, *and*
- $r, \pi, t \models \alpha_1$ *and* $r, \pi, t \models \alpha_2$ *iff there exists* $\underline{a} \in [\![\alpha_1 \wedge \alpha_2]\!]_A$ *with* $\underline{a} \subseteq act(r, t)$.

Nothing in the above semantics forces the interpretation π to have anything to do with the run r. Intuitively, we want the interpretation and the run to agree on what is permitted, what is obligatory, and what is neither permitted nor obligatory. To define this requirement formally, we first give a mapping to relate licenses to sequences of actions. We then use this mapping to state the conditions under which an interpretation is consistent with a run. By consistent, we mean that any permission or obligation specified by the run is also specified by the interpretation. Finally, we show how to find the consistent interpretation that does not permit or obligate an action, unless the run does.

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; and a trace s_1 is said to be a *prefix* of trace s_2 if there is some trace s such that $s_1 \cdot s = s_2$. Given a license ℓ, we construct a function $[\![\ell]\!]_L$ by induction on the

[2] Gunter *et al.* [2001] use the term *reality* for this concept, although their formal definition is different.

structure of ℓ:

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

The function $[\![\ell]\!]_L$ gives the set of traces allowed by the license. We define the function $[\![\ell]\!]_I$ to provide the infinitary version of the sequences corresponding to ℓ. Formally, $[\![\ell]\!]_I = \{s \ :\ s \in [\![\ell]\!]_L, s \text{ infinite}\} \cup \{s \cdot \perp^\infty \ :\ s \in [\![\ell]\!]_L, s \text{ finite}\}$. Finally, a sequence of action s is said to be *viable* for ℓ if s is a prefix of some trace in $[\![\ell]\!]_I$.

We are now ready to define what it means for an interpretation to be *consistent* with a run r. Given a license (n, ℓ) issued at time t_1 in a run r, the *action-sequence* of n up to time t_2, denoted $r[n, t_1, t_2]$, 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 $\pi = (P, O)$ is *consistent* with a run r if and only if for all actions a, for all licenses (n, ℓ) issued at time t, and for all times $t' \geq t$,

- if $r[n, t, t'] \cdot a$ is viable for ℓ, then $(a, n) \in P(t)$, and
- if a is the only action such that $r[n, t, t'] \cdot a$ is viable for $[\![\ell]\!]_I$, then $(a, n) \in O(t)$

In other words, an interpretation is consistent with a run r if it enforces all the permissions and obligations required by the licenses issued in the run.

We are interested in interpretations that enforce exactly the permissions and obligations required by the licenses issued in the run. To obtain one, define an ordering on interpretations such that $(P, O) \leq (P', O')$ if, for all times t, $P(t) \subseteq P'(t)$ and $O(t) \subseteq O'(t)$. Thus, $(P, O) \leq (P', O')$ if for all times t, every action permitted by P is also permitted by P', and every action obligated by O is also obligated by O'. This ordering can easily be seen to give the set of interpretations the structure of a lattice. We say that an element π of a set S of interpretations is minimal in S if $\pi \leq \pi'$ for all $\pi' \in S$.

Proposition 2.2: *For every run r, there exists a unique interpretation π minimal in the set of interpretations consistent with r.*

We can now define a new satisfiability relation based on the minimal interpretation. We define the satisfiability relation $r, t \models_c \varphi$ as an abbreviation for $r, \pi_r, t \models \varphi$, where π_r is the minimal interpretation consistent with run r. Following the lead of the logics presented by Halpern and van der Meyden [2001a, 2001b], we call the $\models_c$ semantics the *closed semantics*.[3] For the remainder of

[3]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 an action is not explicitly permitted in the run, then it is not permitted at all. 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 the run does not include an explicit permission for that action.

this paper, we will only be concerned with the closed semantics. If a formula φ is true at all times in a run r, we say φ is *valid* in r, written $r \models_c \varphi$. If φ is valid in all runs r, we simply say φ is valid, written $\models_c \varphi$. One can verify that under the closed semantics, P and O are dual operators:

Proposition 2.3: *For all r, t, α, $r, t \models_c O\alpha$ if and only if $r, t \models_c \neg P \neg \alpha$.*

This duality is similar to the duality between permission and obligation in most accounts of deontic logic [Meyer and Wieringa 1993].

3 Discussion

One motivation for defining a logic was to give us a language in which to write specifications for licenses. Having defined our logic, the next step is to look at some specifications that are of practical interest and show how they can be represented in our framework. For example, a provider may want to require that a work w is never rendered on a device d if doing so would violate a license ℓ. This specification can be written as:

$$\models_c n : \ell \Rightarrow \Box(\mathsf{render}_n[w, d] \Rightarrow P(\mathsf{render}_n[w, d])).$$

Alternatively, a provider may require that a client is never allowed to render a particular work w on either device d_1 or d_2. This can be written as:

$$\models_c n : \ell \Rightarrow \neg(\mathsf{render}_n[w, d_1] \vee \mathsf{render}_n[w, d_2]).$$

A client may also want to reason about a set of licenses. For instance, the client may want to know that two payments x_1 and x_2 are never due at the same time for two licenses ℓ_1 and ℓ_2. In our logic, the client would check if:

$$\models_c (n_1 : \ell_1 \wedge n_2 : \ell_2) \Rightarrow \Box\neg O(\mathsf{pay}_{n_1}[x_1] \wedge \mathsf{pay}_{n_2}[x_2]).$$

In a practical setting, the usefulness of the logic depends on our ability to check efficiently that a given formula holds in a particular model. Standard model checking techniques can be used to determine if a specification in our logic holds for a run prefix. Although the details are beyond the scope of this paper, we will sketch out the main ideas. Because we use a regular language to express licenses, each license can be represented as an automaton. The model is built by "connecting" these automata, according to the run. As for the specification, it can be written in a temporal logic that has an atomic proposition for each name-action pair. An atomic proposition corresponding to an action a and a license name n is only true in the states for which a is permitted by n. The action is obligatory if it is the only one permitted in the state for the given license. The complexity of this procedure will be addressed in future work.

In this paper we have introduced a framework for precisely stating and rigorously proving properties of licenses. While useful in its own right, the logic provides the foundation on which more expressive rights management logics can be built. For this reason, the logic has been kept simple. For example, the logic does not support multiple clients or multiple providers. It is straightforward, however, to extend the logic to support this by adding appropriate parameters to actions and licenses. 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 report on this extension in future work. Another way in which the logic has been kept simple is that it uses a regular language to express licenses. We can modify the logic to use a different rights language, for instance *DigitalRights* [Gunter, Weeks, and Wright 2001] or *XrML* [ContentGuard, Inc. 2000]. A trace-based semantics, however, is needed for any such language, in the same way that we provided a semantics for our licenses via the $[\![\ell]\!]_I$ mapping. This modification would provide a common ground for comparing different rights languages. 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; van der Meyden 1990].

Acknowledgments

We would like to thank Joe Halpern, Carl Lagoze and Sandy Payette for their helpful comments. Support for this work came from the ONR under grants N00014-00-1-0341, N00014-01-1-0511, and N00014-01-1-0795, and from the NSF under grant IIS-9905955 (Project Prism).

References

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`.

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.

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.

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.

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

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

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.