Reasoning about Dynamic Policies

Riccardo Pucella and Vicky Weissman

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

Abstract. People often need to reason about policy changes *before* they are adopted. For example, suppose a website manager knows that users want to enter her site without going through the welcome page. To decide whether or not to permit this, the wise manager will consider the consequences of modifying the policies (e.g., would this allow users to bypass advertisements and legal notices?). Similiarly, people often need to compare policy sets. For example, consider a person who wants to buy health insurance. Before choosing a provider, the customer will want to compare the different policies. In other words, the customer wants to reason about the effect of choosing one policy set over another. We introduce a logic, based on propositional dynamic logic, in which these tasks can be done. We give a sound and complete axiomatization for our logic, and also show that it is decidable. More precisely, the satisfiability problem is decidable in nondeterministic exponential time.

1 Introduction

Many applications include a set of statements, called policies, that say what is and what is not permitted. Policies arise in many different settings. They can be access control policies, describing which agents are permitted to access resources. They can be legal policies, describing what actions are legally permitted, in a normative sense. An important observation is that an application's set of policies might not be static. They often change over time, particularly in response to a user's request. A user not only asks for policy changes, she usually compares the policies of different applications and chooses the one that's best for her. Even before a policy set can be changed or rejected outright, a system designer needs to create the original set. This might involve comparing different options with respect to what they allow, as well as how difficult they are to implement. Choosing whether or not to modify a policy set, deciding to accept or reject one, and creating policies are nontrivial tasks. To get a sense of what needs to be done in practice, consider the following examples.

Example 1.1. Suppose Alice has a junior library card that lets her into the junior section of the library and nowhere else. Alice asks her librarian Libby for an adult card, because she wants to read the books on Classical Philosophy that are kept in the library's adult nonfiction section. Should Libby change the library's policies so that Alice may act as an adult patron? To answer this question wisely, Libby needs to determine the consequences of her change. If the only consequence is that Alice may access the adult fiction and adult

I. Walukiewicz (Ed.): FOSSACS 2004, LNCS 2987, pp. 453–467, 2004.
© Springer-Verlag Berlin Heidelberg 2004

nonfiction collections, then it seems reasonable for Libby to grant the request. On the other hand, if the adult card would allow Alice to enter the library's section on erotic literature, then Libby might look for another solution.

Example 1.2. A company wants to offer its employees health insurance. The providers under consideration are Aetna and Blue Cross Blue Shield. To make an informed decision, the company needs to determine which actions are permitted under Aetna's policies that are not allowed under Blue Cross Blue Shield's and vice-versa.

Example 1.3. A software company is building a new application. The policies that govern the application need to enforce the *principle of least privilege* [24, p.242], which says that each agent has only those permissions that are necessary to do her job. Alice is told to create the policies. To do this, she needs to build a policy set and then check that it meets the principle of least privilege. Once she has found an appropriate set, she gives it to Bob, whose job is to implement the policies correctly. Bob creates a new policy set that is relatively easy to implement and seems equivalent to the one Alice gave him. Before implementing the new set, however, Bob needs to verify that his set allows exactly the same actions as Alice's.

These examples demonstrate a need for a language in which people can compare policies and reason about suggested changes. There are many languages for articulating and reasoning about policies. A survey by Wieringa and Meyer [29] provides some examples. Others may be found in a variety of Computer Science communities, including computer systems security [7,13,8], automated legal reasoning [19], database integrity [23], and digital rights management [27,17,2,11]. All of these languages were created to determine which permissions follow from a single, fixed set of policies. They simply were not designed to address the issues highlighted by our examples. In particular, they cannot express that one policy set is equivalent to another, or that one is strictly more permissive.

In this paper, we introduce a logic in which we can reason about non-static (i.e., dynamic) policies. The logic is based on Dynamic Logic of Permission (DLP) defined by van der Meyden [21], which is itself based on Propositional Dynamic Logic (PDL) [9]. DLP is used to reason about a fixed policy set that governs an application whose behavior is modeled by a transition system. For example, in DLP, we can formulate the query 'Is Alice permitted to enter the adult fiction section'; then, we can answer the query based on the particular application and policy set. DLP is a very expressive logic. It was developed to support the kind of reasoning found in intelligent legal information systems. To do so, it considers permissions to be associated with transitions (any given state transition is either permitted or forbidden), and provides two different operators to query whether actions are permitted: an action is permitted if there is a possible execution of the action using only permitted transitions, and it is freely permitted if all possible executions of the action use only permitted transitions. For many computer applications, this distinction is not necessary. (Indeed, the examples we use in this paper do not use free permissions.) However, by extending DLP, our logic remains appropriate for reasoning about policies in legal information systems. We extend DLP by adding the ability to mention and to modify the policies of the application *in the formulas*. This lets

us write queries such as 'Assuming we change the policy so that Alice is treated as an adult, may she enter the adult fiction section'. Moreover, we can determine the truth of such conditional queries with respect to the particular application and the original policy set.

The uses for our logic go well beyond reasoning about basic conditionals. In our logic, we can update a policy set (i.e., add or remove policies) within a query at arbitrary points. This allows us to reason about the execution of a scenario which begins under one policy set and completes under a modified version. For example, suppose a university has a policy p that says no one can pass her thesis defense, unless she has fulfilled her minor requirements. After witnessing several students with finished theses, scrambling to meet minor requirements, the university decides that the policy should be changed. The new policy says that minor requirements must be met for a student to pass her preliminary exam; since passing the preliminary exam is already a requirement for passing the defense, the univerisity removes p from its policy set. Now, under either policy set, a student cannot pass her defense unless she has completed her minor requirements. However, a student with fortunate timing can avoid the requirement (she passes the preliminary exam under the old policy and defends under the new). We can use our framework to detect this type of consequence.

The rest of the paper is organized as follows. In the next section we review transition systems. Then, we present both the syntax and the semantics of our logic. We finish the section by applying our logic to the situations in Examples 1.1, 1.2, and 1.3. In Section 3 we give a sound and complete axiomatization for our logic. The satisfiability problem is considered in Section 4, where we show that our logic is decidable. In fact, the satisfiability problem is decidable in nondeterministic exponential time. (We suspect that the problem is decidable in deterministic exponential time, which is the complexity of the satisfiability problem for PDL.) Related work is discussed in Section 5 and we conclude in Section 6. For reasons of space, the proofs are left to the full paper.

2 A Logic for Reasoning about Dynamic Policies

Application Model. We assume the application is modeled by a set of states, a set of labelled transitions, and a set of policies. A state is a snapshot of the application in time. A state can, for instance, record the value of all the variables in the application. Transitions between the states represent progress of the application. Each transition is labelled by an action; intuitively, a transition between states s and s' labelled with an action a means that by performing a in s, the application might progress to s'. Note that actions can be nondeterministic, in the sense that more than one transition from the same state can be labelled with the same action. The set of policies tells us which transitions are permitted.

As an example of these concepts, suppose Alice wants a file and can obtain it either by downloading it from the network or copying it from a disk. We can capture this scenario in a model that has three states, s_1, s_2, and s_3, where Alice wants the file in s_1 and has the network version of the file in s_2 and the disk version of the file in s_3. The model has two transitions, t and t', where t goes from s_1 to s_2 and is labeled 'download from network', while t' goes from s_1 to s_3 and is labeled 'download from disk'. Now suppose t' is permitted and t is not, according to the application's policies. (For instance,

the policy may want to restrict access to the network.) Then if Alice wants the file, she is permitted to copy it from the disk, but is not permitted to download it from the network.

Syntax. We now introduce the syntax of $\mathrm{DLP}_{\mathrm{dyn}}$, which is our logic for reasoning about dynamic policies. (We assume the policies are part of an application whose behavior is modeled by a transition system.) $\mathrm{DLP}_{\mathrm{dyn}}$ is an extension of DLP, which is itself an extension of PDL. As in PDL, we assume a set of primitive actions, Act_0, and then provide combinators for building more complex actions from the primitive ones.

Syntax for Actions:

$a \in Act_0$	primitive action
$\alpha, \beta ::=$	action
$\quad a$	$\quad$ primitive action
$\quad \alpha; \beta$	$\quad$ sequential
$\quad \alpha \cup \beta$	$\quad$ alternative
$\quad \alpha^*$	$\quad$ repetition

The action $\alpha; \beta$ represents the sequential composition of α and β; it means 'first execute α, then execute β'. The action $\alpha \cup \beta$ represents the nondeterministic choice of α or β; it means 'either execute α or execute β'. Finally, the action α^* represents the repeated execution of action α, some nondeterministically chosen number of times (possibly zero).

As with actions, the formulas of our logic are written by combining primitives. In this case, however, the primitives are propositions from a set Φ_0.

Syntax for Formulas:

$p \in \Phi_0$	primitive proposition
$\varphi, \psi, \rho ::=$	formula
$\quad p$	$\quad$ primitive proposition
$\quad \neg\varphi$	$\quad$ negation
$\quad \varphi \wedge \psi$	$\quad$ conjunction
$\quad \langle\alpha\rangle\varphi$	$\quad$ effect of action α
$\quad \mathsf{Perm}(\alpha)\varphi$	$\quad$ permission
$\quad \mathsf{FreePerm}(\alpha)\varphi$	$\quad$ free-choice permission
$\quad \mathsf{Grant}(\rho_1, \rho_2)\varphi$	$\quad$ granting permissions
$\quad \mathsf{Revoke}(\rho_1, \rho_2)\varphi$	$\quad$ revoking permissions

The negation ($\neg$) and conjunction ($\wedge$) operators are the standard ones from propositional logic. We abbreviate $\neg(\neg\varphi \wedge \neg\psi)$ as $\varphi \vee \psi$ and abbreviate $\neg\varphi \vee \psi$ as $\varphi \Rightarrow \psi$. Also, we define $true$ to be the formula $p \vee \neg p$, where p is a fixed primitive proposition in Φ_0. $false$ is $\neg true$. We define the sublanguage Φ_p of *propositional formulas* of our logic; it is the set of primitive propositions in Φ_0 closed under negation and conjunction. We let ρ range over propositional formulas in Φ_p.

The PDL operator $\langle\alpha\rangle\varphi$ says by doing α, the application can progress to a state satisfying φ. We abbreviate $\neg\langle\alpha\rangle\neg\varphi$ as $[\alpha]\varphi$. Observe that the formula $[\alpha]\varphi$ means after any execution of α, the formula φ is true.

The DLP operators, which we write as $\mathsf{Perm}(\alpha)\varphi$ and $\mathsf{FreePerm}(\alpha)\varphi$, capture two different types of permissions. The formula $\mathsf{Perm}(\alpha)\varphi$ means there is at least one execution of α that is both permitted and leads to a state where φ is true. For example, consider the formula $\mathsf{Perm}(\text{download} \cup \text{copy})\text{haveFile}$. It says that there is a way to get the file legitimately either by downloading it from the network or copying it from the disk, however, it does not say which of the two actions is permitted. The formula $\mathsf{FreePerm}(\alpha)\varphi$ means *all* executions of α that lead to a state satisfying φ are permitted. For example, consider the formula $\mathsf{FreePerm}(\text{download} \cup \text{copy})\text{haveFile}$. It says that every way of obtaining the file by downloading it from the network or copying it from the disk is legitimate.

Finally, we introduce the operators $\mathsf{Grant}(\rho_1, \rho_2)\varphi$ and $\mathsf{Revoke}(\rho_1, \rho_2)\varphi$. The formula $\mathsf{Grant}(\rho_1, \rho_2)\varphi$ means φ holds, if we assume every transition from a state satisfying ρ_1 to a state satifying ρ_2 is permitted. Conversely, the formula $\mathsf{Revoke}(\rho_1, \rho_2)\varphi$ means φ holds, if we assume that every transition from a state satisfying ρ_1 to a state satifying ρ_2 is *not* permitted. The only restriction on these operators is that ρ_1 and ρ_2 must be propositional formulas. Roughly speaking, this limitation means that we cannot easily reason about permissions that are defined in terms of other permissions. For example, we cannot say 'φ holds if whenever someone is permitted to download a file from the network, she is permitted to copy it from the disk'. (We believe that none of our results fundamentally depend on this restriction.)

Semantics. The semantics of our logic is based on Kripke structures, which are the formal models of the applications. Intuitively, a Kripke structure encodes a transition system, along with the characteristics of each state (i.e., which primitive propositions are true in each state). A Kripke structure $M = (S, \pi, \tau)$ is a set of states S, an interpretation π used to interpret the primitive propositions, and an interpretation τ used to interpret the primitive actions. More specifically, for a primitive proposition p, $\pi(p)$ is the set of states where p holds, and for a primitive action a, $\tau(a)$ is the set of transitions $s_1 s_2$ that could occur by doing a.

We associate every (not necessarily primitive) action α with a set of finite traces, where a trace is a sequence of states. Roughly speaking, a trace is in the set if there is an execution of the action that travels through each of the states in the trace, in turn. The set of traces $\tau_s(\alpha)$ includes every trace that could be encountered during an execution of α from state s. The following table defines this notion formally

Sequences of States Associated with Actions: $\tau_s(\alpha)$

$$\tau_s(a) \triangleq \{s_1 s_2 \in \tau(a) \mid s_1 = s\}$$
$$\tau_s(\alpha;\beta) \triangleq \{\sigma_\alpha s' \sigma_\beta \mid \sigma_\alpha s' \in \tau_s(\alpha), s'\sigma_\beta \in \tau_{s'}(\beta)$$
$$\tau_s(\alpha \cup \beta) \triangleq \tau_s(\alpha) \cup \tau_s(\beta)$$
$$\tau_s(\alpha^*) \triangleq \{ss\} \cup \tau_s(\alpha) \cup \tau_s(\alpha;\alpha) \cup \tau_s(\alpha;\alpha;\alpha) \cup \ldots$$

This definition of τ_s essentially yields the trace semantics of PDL [25].

To establish the truth of our formulas, we need to keep track of which transitions are assumed to be permitted. We store this information in a *policy set* P, which is simply a set of transitions. A transition is assumed to be permitted, according to P, if and only if it

is in P. If a transition is in P, then we say it is *P-green*. Otherwise, we say the transition is *P-red*. More generally, a sequence of transitions is P-green, if every transition in the sequence is P-green. Otherwise, the sequence is P-red. Notice that this definition suggests that an action sequence is illegal if any action in the sequence is illegal.

A formula φ is true (or satisfied) in a state s of a model M given a policy set P, written $(M, s, P) \models \varphi$, if it is true according to the following definition, where σ_f denotes the final element of σ for any nonempty finite sequence σ.

Satisfaction Relation: $(M, s, P) \models \varphi$

$(M, s, P) \models p$	if $s \in \pi(p)$
$(M, s, P) \models \neg\varphi$	if $(M, s, P) \not\models \varphi$
$(M, s, P) \models \varphi \wedge \psi$	if $(M, s, P) \models \varphi$ and $(M, s, P) \models \psi$
$(M, s, P) \models \langle\alpha\rangle\varphi$	if for some $\sigma \in \tau_s(\alpha)$, $(M, \sigma_f, P) \models \varphi$
$(M, s, P) \models \mathsf{Perm}(\alpha)\varphi$	if for some P-green $\sigma \in \tau_s(\alpha)$, $(M, \sigma_f, P) \models \varphi$
$(M, s, P) \models \mathsf{FreePerm}(\alpha)\varphi$	if for all $\sigma \in \tau_s(\alpha)$ such that $(M, \sigma_f, P) \models \varphi$, σ is P-green
$(M, s, P) \models \mathsf{Grant}(\rho_1, \rho_2)\varphi$	if $(M, s, P \cup P^{\rho_1,\rho_2}) \models \varphi$
$(M, s, P) \models \mathsf{Revoke}(\rho_1, \rho_2)\varphi$	if $(M, s, P \setminus P^{\rho_1,\rho_2}) \models \varphi$

where $P^{\rho_1,\rho_2} \triangleq \{s_1 s_2 \mid (M, s_1, P) \models \rho_1, (M, s_2, P) \models \rho_2\}$

A formula φ is true at a state s of a model M, written $(M, s) \models \varphi$, if for any policy set P, $(M, s, P) \models \varphi$. We can easily check that a propositional formula does not require the set of policies to determine its truth value. Formally, if ρ is a propositional formula, then $(M, s, P) \models \rho$ for some P if and only if $(M, s) \models \rho$. It follows that P^{ρ_1,ρ_2} is $\{(s_1, s_2) \mid (M, s_1) \models \rho_1, (M, s_2) \models \rho_2\}$, because ρ_1 and ρ_2 are propositional formulas. If $(M, s) \models \varphi$ for all $s \in S$, then we say φ is *valid in* M, and write $M \models \varphi$. Finally, if $M \models \varphi$ for all Kripke structures M, we say φ is *valid*, and write $\models \varphi$. We now revisit the examples given in the introduction.

Example 2.1. In Example 1.1, we present a scenario in which the librarian Libby needs to decide whether or not to give Alice an adult patron card. To make this example more concrete, suppose that having an adult card means Alice may do any primitive action in a set Act_A. We now show that we can use our logic to help Libby make an informed decision. To do this, suppose

- M is the model that represents the library system and P is the library's current policy set.
- A state in M satisfies the primitive proposition 'Alice acted as an adult' if and only if every transition into the state is labeled with an action in Act_A.
- For ease of exposition, we assume that either all transitions into a state are labeled with an action in Act_A or none are. (Note that if this is not true, we could easily create a model, equivalent to M, that satisfies the condition.)

An action α that Alice may not do according to P would be allowed according to the modified policy set, if for some state s in M

$$(M, s, P) \models \neg\mathsf{Perm}(\alpha)\mathit{true} \wedge \mathsf{Grant}(\mathit{true}, \text{'Alice acted as an adult'})\mathsf{Perm}(\alpha)\mathit{true}.$$

By considering each action α of interest, we can determine the consequences of Libby granting Alice's request.

Example 2.2. Suppose the company in Example 1.2 suspects that every permission they care about is either granted by the Blue Cross Blue Shield policies, or is not granted by either set of policies. To test this hypothesis:

- Let P_A and P_B be the Aetna and Blue Cross Blue Shield policies, respectively.
- Let M be a Kripke structure capturing the states of the application and the possible transitions. For example, a state could represent the flu season, and a transition from the state could represent Alice getting a free flu shot. We assume that every state can be uniquely described by a propositional formula; in other words, for every state s, there is a propositional formula ρ_s which is true only at s.
- Let $\mathsf{Pol}_A(\varphi)$ be an abbreviation for $\mathsf{Grant}(\rho_{s_1}, \rho_{s'_1}) \ldots \mathsf{Grant}(\rho_{s_k}, \rho_{s'_k})\varphi$ where $P_A = \{s_1 s'_1, \ldots, s_k s'_k\}$. Let $\mathsf{Pol}_B(\varphi)$ be the corresponding abbreviation based on P_B.
- Let φ_{desP} be a formula that represents the desired permissions. As a simple example, φ_{desP} could be $\mathsf{Perm}(\text{Alice gets free flu shot})\mathit{true}$, which means there is a way for Alice to get a free flu shot.

The company's hypothesis is correct if

$$M \models (\mathsf{Pol_A}(\varphi_{\mathrm{desP}})) \Rightarrow (\mathsf{Pol_B}(\varphi_{\mathrm{desP}})).$$

Example 2.3. Consider Example 1.3.

- Let P_{LP} be the set of policies that Alice created to enforce the principle of least privilege.
- Let M be a Kripke structure capturing the states of the application and the possible transitions. As in the previous example, we assume that every state can be uniquely described by a propositional formula; in other words, for every state s, there is a propositional formula ρ_s which is true only at s.
- Let $\mathsf{Pol}_P(\varphi)$ be an abbreviation for $\mathsf{Grant}(\rho_{s_1}, \rho_{s'_1}) \ldots \mathsf{Grant}(\rho_{s_k}, \rho_{s'_k})\varphi$, for any policy set $P = \{s_1 s'_1, \ldots, s_k s'_k\}$.
- Let φ_{jobP} be a formula that represents the permissions required for users to do their job. As a simple example, φ_{jobP} could be $\mathsf{Perm}(\text{edit user's own files})\mathit{true}$, which means users have a way to edit their own files.

We want to verify that P_{LP} satisfies the principle of least privilege. However, this is a bit tricky, because there are at least two interpretations of the principle. The first says that P_{LP} satisfies the principle of least privilege if we cannot remove any policy from P_{LP} and still allow the users to do their job. According to this definition, P_{LP} satisfies the principle of least privilege if and only if

$$M \models \mathsf{Pol}_{P_{LP}} \left(\varphi_{\mathrm{jobP}} \wedge \bigwedge_{ss' \in P_{LP}} \mathsf{Revoke}(\rho_s, \rho_{s'}) \neg \varphi_{\mathrm{jobP}} \right).$$

A second interpretation is the stronger statement that P_{LP} satisfies the principle of least privilege if it lets the users do their job, and there is no smaller set of policies that does. Assuming that the Kripke structure M is finite, we can formalize this interpretation as follows. The policy set P_{LP} satisfies the principle of least privilege if and only if

$$M \models \mathsf{Pol}_{P_{LP}}(\varphi_{\mathrm{jobP}}) \wedge \bigwedge_{P \in \mathcal{P}_M} \mathsf{Pol}_P(\neg\varphi_{\mathrm{jobP}}),$$

where $\mathcal{P}_M = \{P \mid P \text{ is a policy set over } M, |P| < |P_{LP}|\}$, the set of all policy sets with fewer elements than P_{LP}. The key observation is not that there are many interpretations of the principle of least privilege, but that we can capture the different interpretations in our framework.

Before leaving this section, we should emphasize that $\mathsf{Grant}(\rho_1, \rho_2)\varphi$ means 'φ holds under the assumption that every *single* transition from a state satisfying ρ_1 to a state satisfying ρ_2 is permitted'. This does not mean that we assume all *sequences* of transitions from states satisfying ρ_1 to states satisfying ρ_2 are permitted. This consequence of our logic seems particularly desirable. To see why consider the statement 'any transition from a state in which Alice is in school to one in which she is home is permitted'. It might follow from the statement that Alice may bike home from school or even take a cab. However, we should not conclude from the statement that Alice is allowed to bike from school to the docks, convince some disreputable people to buy her beer, stagger home, and then beat-up her brother, despite the fact that the action sequence begins with Alice at school and ends with Alice at home.

3 A Sound and Complete Axiomatization

In this section we present a sound and complete axiomatization for our logic. Recall that a formula φ is *provable* if it can be proven using the axiom system's axioms and rules of inferences. If every provable formula is valid, then the axiom system is *sound*. If every valid formula is provable, then the axiom system is *complete*.

Our axiom system **AX** can be divided into six parts. The first set of axioms accounts for propositional reasoning.

Axioms for Propositional Reasoning:

Taut. All instances of propositional tautologies
MP. From φ and $\varphi \Rightarrow \psi$ infer ψ

As an example, an instance of **Taut** is $\varphi \vee \neg\varphi$, for any formula φ. Axiom **Taut** can be replaced by a sound and complete axiomatization for propositional tautologies, such as the one given in Mendelson [20].

The second set of axioms accounts for the PDL modality $\langle\,\rangle$.

Axioms for $\langle\,\rangle$:

A1. $\langle\alpha\rangle \mathit{false} \Leftrightarrow \mathit{false}$

A2. $\langle\alpha;\beta\rangle\varphi \Leftrightarrow \langle\alpha\rangle\langle\beta\rangle\varphi$
A3. $\langle\alpha\cup\beta\rangle\varphi \Leftrightarrow \langle\alpha\rangle\varphi \vee \langle\beta\rangle\varphi$
A4. $\langle\alpha^*\rangle\varphi \Leftrightarrow \varphi \vee \langle\alpha;\alpha^*\rangle\varphi$
A5. $\langle\alpha\rangle(\varphi\vee\psi) \Leftrightarrow \langle\alpha\rangle\varphi \vee \langle\alpha\rangle\psi$
A6. $\varphi \wedge [\alpha^*](\varphi \Rightarrow [\alpha]\varphi) \Rightarrow [\alpha^*]\varphi$
A7. From φ infer $[\alpha]\varphi$

This is essentially the axiomatization for PDL due to Segerberg [28]. Axioms **A1** through **A5** and axiom **A7** are straightforward. Axiom **A6** is an induction axiom that captures the infinitary behavior of the * operator.

The third set of axioms accounts for the DLP modalities Perm and FreePerm.

Axioms for Perm **and** FreePerm**:**

P1. $\mathsf{Perm}(\alpha)\varphi \Rightarrow \langle\alpha\rangle\varphi$
P2. $\mathsf{Perm}(\alpha;\beta)\varphi \Leftrightarrow \mathsf{Perm}(\alpha)\mathsf{Perm}(\beta)\varphi$
P3. $\mathsf{Perm}(\alpha\cup\beta)\varphi \Leftrightarrow \mathsf{Perm}(\alpha)\varphi \vee \mathsf{Perm}(\beta)\varphi$
P4. $\mathsf{Perm}(\alpha^*)\varphi \Leftrightarrow \varphi \vee \mathsf{Perm}(\alpha;\alpha^*)\varphi$
P5. $\mathsf{Perm}(\alpha)(\varphi\vee\psi) \Leftrightarrow \mathsf{Perm}(\alpha)\varphi \vee \mathsf{Perm}(\alpha)\psi$
P6. $\varphi \wedge \neg(\mathsf{Perm}(\alpha^*)\neg(\varphi \Rightarrow \neg\mathsf{Perm}(\alpha)\neg\varphi)) \Rightarrow \neg\mathsf{Perm}(\alpha^*)\neg\varphi$
P7. $[\alpha]\neg\varphi \Rightarrow \mathsf{FreePerm}(\alpha)\varphi$
P8. $\mathsf{FreePerm}(\alpha;\beta)\varphi \Leftrightarrow \mathsf{FreePerm}(\alpha)\langle\beta\rangle\varphi \wedge [\alpha]\mathsf{FreePerm}(\beta)\varphi$
P9. $\mathsf{FreePerm}(\alpha\cup\beta)\varphi \Leftrightarrow \mathsf{FreePerm}(\alpha)\varphi \wedge \mathsf{FreePerm}(\beta)\varphi$
P10. $\mathsf{FreePerm}(\alpha^*)\varphi \Leftrightarrow \mathsf{FreePerm}(\alpha;\alpha^*)\varphi$
P11. $\mathsf{FreePerm}(\alpha)(\varphi\vee\psi) \Leftrightarrow \mathsf{FreePerm}(\alpha)\varphi \vee \mathsf{FreePerm}(\alpha)\psi$
P12. $[\alpha^*]\mathsf{FreePerm}(\alpha)\langle\alpha^*\rangle\varphi \Rightarrow \mathsf{FreePerm}(\alpha^*)\varphi$
P13. $\mathsf{FreePerm}(\alpha)\varphi \wedge \langle\alpha\rangle\varphi \Rightarrow \mathsf{Perm}(\alpha)\varphi$
P14. $\mathsf{Perm}(\alpha)\varphi \wedge [\alpha](\varphi \Rightarrow \psi) \Rightarrow \mathsf{Perm}(\alpha)\psi$
P15. $\mathsf{FreePerm}(\alpha)\psi \wedge [\alpha](\varphi \Rightarrow \psi) \Rightarrow \mathsf{FreePerm}(\alpha)\varphi$

These axioms are due to van der Meyden [21]. Axioms **P1–P6** and **P7–P12** correspond closely to axioms **A1–A6**, indicating the tight relationship between the PDL and DLP modalities. (This relationship is further clarified by Csirmaz [3].) Note that axiom **P6** uses the dual of $\mathsf{Perm}(\alpha)\varphi$, written as $\neg\mathsf{Perm}(\alpha)\neg\varphi$. Axioms **P13–P15** capture the interactions between the different modalities.

The fourth set of axioms concerns the behavior of the Grant operator.

Axioms for Grant**:**

G1. $\mathsf{Grant}(\rho_1,\rho_2)(\varphi\wedge\psi) \Leftrightarrow \mathsf{Grant}(\rho_1,\rho_2)\varphi \wedge \mathsf{Grant}(\rho_1,\rho_2)\psi$
G2. $\mathsf{Grant}(\rho_1,\rho_2)\neg\varphi \Leftrightarrow \neg\mathsf{Grant}(\rho_1,\rho_2)\varphi$
G3. $\mathsf{Grant}(\rho_1,\rho_2)\langle\alpha\rangle\varphi \Leftrightarrow \langle\alpha\rangle\mathsf{Grant}(\rho_1,\rho_2)\varphi$
G4. $\mathsf{Grant}(\rho_1,\rho_2)\mathsf{Grant}(\rho_3,\rho_4)\varphi \Leftrightarrow \mathsf{Grant}(\rho_3,\rho_4)\mathsf{Grant}(\rho_1,\rho_2)\varphi$
G5. From $\rho_3 \Rightarrow \rho_1$ and $\rho_4 \Rightarrow \rho_2$ infer $\mathsf{Grant}(\rho_1,\rho_2)\mathsf{Grant}(\rho_3,\rho_4)\varphi \Leftrightarrow \mathsf{Grant}(\rho_1,\rho_2)\varphi$
G6. $\mathsf{Grant}(\mathit{false},\rho)\varphi \Leftrightarrow \varphi$
G7. $\mathsf{Grant}(\rho,\mathit{false})\varphi \Leftrightarrow \varphi$
G8. $\mathsf{Grant}(\rho_1,\rho_2)p \Leftrightarrow p$ for primitive propositions p

G9. $\mathsf{Grant}(\rho_1 \vee \rho_2, \rho_3)\varphi \Leftrightarrow \mathsf{Grant}(\rho_1, \rho_3)\mathsf{Grant}(\rho_2, \rho_3)\varphi$
G10. $\mathsf{Grant}(\rho_1, \rho_2 \vee \rho_3)\varphi \Leftrightarrow \mathsf{Grant}(\rho_1, \rho_2)\mathsf{Grant}(\rho_1, \rho_3)\varphi$
G11. $\mathsf{Grant}(\rho_1, \rho_2)(\rho_1 \wedge \langle a \rangle \rho_2) \Rightarrow \mathsf{Grant}(\rho_1, \rho_2)(\mathsf{Perm}(a)\rho_2)$ for primitive actions a
G12. $\mathsf{Grant}(\rho_1, \rho_2)\mathsf{Perm}(\alpha)\varphi \Leftrightarrow \mathsf{Grant}(\rho_1, \rho_2)\mathsf{Perm}(\alpha)\mathsf{Grant}(\rho_1, \rho_2)\varphi$
G13. $\mathsf{Grant}(\rho_1, \rho_2)\mathsf{FreePerm}(\alpha)\varphi \Leftrightarrow \mathsf{Grant}(\rho_1, \rho_2)\mathsf{FreePerm}(\alpha)\mathsf{Grant}(\rho_1, \rho_2)\varphi$
G14. From φ infer $\mathsf{Grant}(\rho_1, \rho_2)\varphi$

Axioms **G1–G3** capture the behavior of Grant under conjunctions, negations, and the PDL modality. Axiom **G4** says that the order in which permissions are granted is irrelevant. The inference rule **G5** allows a permission to be disregarded if it is already implied by a permission that was granted earlier in the analysis. Axioms **G4** and **G5** together imply that if ρ_1, ρ_2 are respectively equivalent to ρ_3, ρ_4, then $\mathsf{Grant}(\rho_1, \rho_2)\varphi \Leftrightarrow \mathsf{Grant}(\rho_3, \rho_4)\varphi$. Axioms **G6** through **G8** say that an occurrence of the Grant operator can be removed, if it clearly doesn't affect the truth of the formula. Axioms **G9** and **G10** capture the fact that in some sense permission grants are cumulative. Finally, Axioms **G11** through **G13** capture the relationship between granting permissions and the other permission modalities.

The fifth set of axioms concerns the behavior of the Revoke operator.

Axioms for Revoke**:**

R1. $\mathsf{Revoke}(\rho_1, \rho_2)(\varphi \wedge \psi) \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_2)\varphi \wedge \mathsf{Revoke}(\rho_1, \rho_2)\psi$
R2. $\mathsf{Revoke}(\rho_1, \rho_2)\neg\varphi \Leftrightarrow \neg\mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R3. $\mathsf{Revoke}(\rho_1, \rho_2)\langle\alpha\rangle\varphi \Leftrightarrow \langle\alpha\rangle\mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R4. $\mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Revoke}(\rho_3, \rho_4)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_3, \rho_4)\mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R5. From $\rho_3 \Rightarrow \rho_1$ and $\rho_4 \Rightarrow \rho_2$ infer
$\mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Revoke}(\rho_3, \rho_4)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R6. $\mathsf{Revoke}(\mathit{false}, \rho)\varphi \Leftrightarrow \varphi$
R7. $\mathsf{Revoke}(\rho, \mathit{false})\varphi \Leftrightarrow \varphi$
R8. $\mathsf{Revoke}(\rho_1, \rho_2)p \Leftrightarrow p$ for primitive propositions p
R9. $\mathsf{Revoke}(\rho_1 \vee \rho_2, \rho_3)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_3)\mathsf{Revoke}(\rho_2, \rho_3)\varphi$
R10. $\mathsf{Revoke}(\rho_1, \rho_2 \vee \rho_3)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Revoke}(\rho_1, \rho_3)\varphi$
R11. $\mathsf{Revoke}(\rho_1, \rho_2)(\rho_1 \wedge [a]\rho_2) \Rightarrow \mathsf{Revoke}(\rho_1, \rho_2)(\neg\mathsf{Perm}(a)\rho_2)$
for primitive actions a
R12. $\mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Perm}(\alpha)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Perm}(\alpha)\mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R13. $\mathsf{Revoke}(\rho_1, \rho_2)\mathsf{FreePerm}(\alpha)\varphi \Leftrightarrow \mathsf{Revoke}(\rho_1, \rho_2)\mathsf{FreePerm}(\alpha)\mathsf{Revoke}(\rho_1, \rho_2)\varphi$
R14. From φ infer $\mathsf{Revoke}(\rho_1, \rho_2)\varphi$

These axioms are essentially **G1–G14**, with Grant replaced by Revoke. The only exception is **R11**, which says that an action corresponding to a revoked transition is not permitted.

Finally, the last set of axioms capture the interaction between permission grants and permission revocations.

Interaction Axioms for Grant **and** Revoke**:**

I1. $\mathsf{Grant}(\rho_1, \rho_2)\mathsf{Revoke}(\rho_3, \rho_4)\varphi \Leftrightarrow$
$\mathsf{Revoke}(\rho_3, \rho_4)\mathsf{Grant}(\rho_1, \rho_2 \wedge \neg\rho_4)\mathsf{Grant}(\rho_1 \wedge \neg\rho_3, \rho_2)\varphi$
I2. $\mathsf{Revoke}(\rho_1, \rho_2)\mathsf{Grant}(\rho_3, \rho_4)\varphi \Leftrightarrow$
$\mathsf{Grant}(\rho_3, \rho_4)\mathsf{Revoke}(\rho_1, \rho_2 \wedge \neg\rho_4)\mathsf{Revoke}(\rho_1 \wedge \neg\rho_3, \rho_2)\varphi$

Roughly speaking, axiom **I1** says that granting some permissions P_1 and then revoking other permissions P_2 is equivalent to first revoking the permissions P_2, and then granting the permissions in P_1 that would not have been revoked by P_2. A similar explanation applies to **I2**. Note that it follows from **I1** and **G6–G7** that if $\rho_1 \Rightarrow \rho_3$ and $\rho_2 \Rightarrow \rho_4$ are tautologies, then $\mathsf{Grant}(\rho_1, \rho_2)\mathsf{Revoke}(\rho_3, \rho_4)\varphi$ is equivalent to $\mathsf{Revoke}(\rho_3, \rho_4)\varphi$. In other words, granting permissions that are immediately revoked is equivalent to never granting the permissions at all. Again, a similar argument holds for axiom **I2**.

As discussed at the end of Section 2, a prerequisite for the soundness of these axioms is that a primitive action must be mapped to single transition. More specifically, the soundness of Axioms **G11** and **R11** depend on this restriction. To see why, consider the (violating) structure M that has three states s_1, s_2, s_3, with $\pi(p) = \{s_1\}$, $\pi(q) = \{s_3\}$, $\tau(a) = \{s_1 s_2\}$, $\tau(b) = \{s_2 s_3\}$, $\tau(c) = \{s_1 s_3\}$, and $\tau(d) = \{s_1 s_2 s_3\}$. Clearly, $(M, s_1, \varnothing) \models \mathsf{Grant}(p, q)(p \wedge \langle d \rangle q)$ holds. However, we do *not* have $(M, s_1, \varnothing) \models \mathsf{Grant}(p, q)\mathsf{Perm}(d)q$, since under the policy set $\varnothing^{p,q}$, the sequence $s_1 s_2 s_3$ is red. Therefore, axiom **G11** cannot be sound, unless every primitive action is mapped to a single transition. A similar argument holds for axiom **R11**.

Theorem 3.1. *The axiomatization* $\mathbf{AX}$ *is sound and complete for* $\mathrm{DLP}_{\mathrm{dyn}}$ *with respect to Kripke structures.*

To establish completeness, it is possible, although not at all immediate, to use an approach similar to that used by Kozen and Parikh [15] to prove completeness of the axiomatization for PDL. (This approach was also used by van der Meyden [21] to prove completeness of DLP.) We first note that completeness is equivalent to the statement that all consistent formulas are satisfiable. Recall that a formula φ is *consistent* if the formula $\neg\varphi$ is not provable and a formula φ is *satisfiable* if there exists a Kripke structure M, a state s of that structure, and a policy P such that $(M, s, P) \models \varphi$. So, we can prove completeness if for any consistent formula φ, we can construct a model that satisfies it. We construct this model for an arbitrary, consistent formula φ, by taking sets of subformulas of φ to be states. Details are given in the full paper.

4 Complexity

Having described a sound and complete axiomatization for our logic, we now turn to the complexity of the satisfiability problem. (Recall that the satisfiability problem asks if there is a a Kripke structure M, a state s in M, and a policy P such that $(M, s, P) \models \varphi$, for a given formula φ.) Because our logic extends PDL, our decision problem is at least as difficult as PDL's. Therefore, our decision problem has an EXPTIME lowerbound [5].

To find an upperbound, we first prove a small model theorem that intuitively says that if a formula φ is satisfiable, then it is satisfiable in a Kripke structure with a comparatively small number of states. Define the length $|\varphi|$ of a formula to be the number of symbols required to write φ.

Theorem 4.1. *If φ is satisfiable, then $(M, s, P) \models \varphi$ for a Kripke structure $M = (S, \pi, \tau)$ with $|S| \leq 2^{|\varphi|^2}$.*

The following theorem shows that checking that a formula is satisfied in a particular finite model can be done efficiently.

Theorem 4.2. *There is an algorithm that decides $(M, s, P) \models \varphi$ in time polynomial in $|M|$, $|P|$ and $|\varphi|$.*

Using Theorems 4.1 and 4.2, we can establish the following upperbound.

Theorem 4.3. *The decision problem for $\mathrm{DLP}_{\mathrm{dyn}}$ is in NEXPTIME.*

Theorem 4.3 establishes that $\mathrm{DLP}_{\mathrm{dyn}}$ is decidable. The theorem also implies a (previously unknown) bound on the decision problem of DLP. This result is not immediately apparent, because the DLP models are more general than ours; they allow primitive actions to be mapped to sequences of transitions. However, it is a consequence of van der Meyden's completeness proof that any satisfiable DLP formula is satisfiable in a model where primitive actions are mapped to single transitions. It follows from Theorem 4.3 that DLP is in NEXPTIME. We conjecture that the decision problem for $\mathrm{DLP}_{\mathrm{dyn}}$ is in fact EXPTIME-complete, just like PDL [9]. It should be possible to adapt the deterministic single exponential time algorithm given by Pratt [26], but this is left as future work.

5 Related Work

To the best of our knowledge, $\mathrm{DLP}_{\mathrm{dyn}}$ is the first language explicitly designed to answer the kind of questions we discussed in the introduction. There is, however, a significant body of work on reasoning about permissions. There are fundamentally two approaches, propositional modal logics and first-order logics.

Building on the work of von Wright [30], many people have based logics for reasoning about permissions on propositional modal logic [10]. These logics, which are typically called *deontic logics*, interpret permission via an operator $P\varphi$, which can be read 'φ is permitted', or 'it is permitted to make φ true'. Unfortunately, a naive treatment of permission as a modality leads to a number of counterintuitive results. Von Wright [31] recognized that many paradoxes arise because the logics do not distinguish between propositions and actions. More precisely, many paradoxes are a consequence of applying permissions to formulas, instead of just actions.

One of the first languages to restrict permissions to actions is due to Meyer [22]. Meyer's logic is PDL with additional modalities to reason about permissions. To interpret permissions, he essentially divides the states in the model of the system into good states and bad states; an action is permitted if it leads to a good state. Most of the paradoxes of deontic logic disappear in this setting.

As discussed by van der Meyden [21], however, some paradoxes remain. In particular, reasoning about the permission of sequential actions is problematic, because the logic assigns permissions only to states. For example, suppose that no one is allowed to murder the president and, if someone does, then that person goes to jail. If the state in which the murderer goes to jail is a good state, which intuitively it should be, then Meyer's logic says that anyone may murder the president, providing that he or she then goes to jail. But no one may murder the president, so this is a paradoxical situation. Another consequence of assigning permissions to states is that the logic cannot capture subtle distinctions in the use of the term 'permission'. In particular, the logic cannot distinguish between the two types of permissions captured in our logic by the DLP operators Perm and $\mathsf{FreePerm}$. To address these issues, van der Meyden designed the logic DLP.

Clearly, our work is an extension of DLP. One way to view the relationship between $\mathrm{DLP}_{\mathrm{dyn}}$ and DLP is that it is akin to the relationship between propositional logic and PDL. Propositional logic is used to reason about a single state, while PDL extends the logic to reason about multiple states and the transition between them. Similarly, DLP is used to reason about a single set of allowed transitions, while $\mathrm{DLP}_{\mathrm{dyn}}$ extends DLP to reason about multiple sets of allowed transitions, using the operators $\mathsf{Grant}(\rho_1, \rho_2)\varphi$ and $\mathsf{Revoke}(\rho_1, \rho_2)\varphi$ to move from set to set.

Although we base our logic on DLP, there is a difference between our models and the ones used by DLP. Specifically, DLP allows primitive actions to be assigned to sequences of transitions; we impose the restriction that each primitive action is mapped to a single transition. This restriction is necessary for the axiomatization that we give in Section 3.

The second class of languages for reasoning about permissions are first-order logics. In the Computer Science community, these languages are typically an extension of Datalog [6], which is a tractable fragment of first-order logic. Approaches based on Datalog include [4,17,14,16,18]. In these languages, the environment, which essentially corresponds to our application models, is a conjunction of formulas of the form $\forall x_1, \ldots, x_n.(l_1 \wedge \ldots \wedge l_k \Rightarrow l_{k+1})$, where each l_i is a literal, l_{k+1} is a positive (i.e., non-negated) literal, and depending on the particular language, other restrictions might apply. It is not clear whether or not our models can be encoded in their environments, because of the restrictions on negation. (This also holds for approaches that are not based on Datalog, such as [8].)

Although the first-order approaches might not be able to capture our models, they do support variables. This allows their specifications to be more concise. It is interesting to note, however, that XrML [2], which is a language that has recieved widespread support in industry, assumes the domain of interest is finite. [1] In other words, for any formula in the logic, there is an admittedly longer formula that is variable-free. Thus, in practice, it seems likely that variable-free languages are sufficiently expressive.

Finally, we should note that both the modal approaches and the first-order languages typically assume that any action that is not permitted is forbidden. However, there are exceptions [12,1,13,8]. By allowing actions to be neither permitted nor forbidden, we can sensibly merge policies that govern the same system. In future work, we would like to explore these possibilities within our framework.

[1] The XrML authorization algorithm, which determines if a permission follows from a set of XrML policies, terminates only for finite domains.

6 Conclusion

In this paper, we identify a class of problems that are of practical interest and that have not been addressed previously in the literature. Essentially, these problems arise when there is not a single, fixed set of policies. Examples include comparing different policy sets and understanding the consequences of an evolving policy set.

Not only have we found an interesting class of problems, our work shows that the approaches for reasoning about single sets can be adapted to handle the new issues. We were able to extend DLP to create a logic in which to compare policy sets and reason about changing policies. To the best of our knowledge, ours is the first logic designed explicitly for this purpose. By modifying existing proof techniques, we were able to obtain a sound and complete axiomatization for the logic. Moreover, despite the added expressiveness, the decision problem remains decidable.

As illustrated by Examples 2.1, 2.2, and 2.3, a key problem is verifying that a model satisfies a given formula. Theorem 4.2 provides a general bound on the complexity of the model checking problem. It would be interesting to investigate efficient techniques to perform this verification.

Acknowledgments. Thanks to Joe Halpern for comments on an early draft of this paper. This work was partially supported by NSF under grant CTC-0208535, by ONR under grant N00014-02-1-0455, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

References

1. J. Chomicki, J. Lobo, and S. Naqvi. A logic programming approach to conflict resolution in policy management. In *Principles of Knowledge Representation and Reasoning: Proc. Ninth International Conference (KR '00)*, pages 121–132, 2000.
2. ContentGuard. XrML: Extensible rights Markup Language. Available from `http://www.xrml.org`, 2001.
3. L. Csirmaz. Multi-level permission. Technical Report 90-25, DIMACS, 1990.
4. J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Research in Security and Privacy*, pages 95–103. IEEE Computer Society Press, 2002.
5. M. J. Fisher and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
6. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.
7. J. Glasgow, G. MacEwen, and P. Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, 1992.
8. J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 187–201. IEEE Computer Society Press, 2003.
9. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, 2000.
10. G. Hughes and M. Cresswell. *An Introduction to Modal Logic*. Methuen, 1972.

11. R. Iannella. Open Digital Rights Language (ODRL) version 1.1. Available from `http://www.w3.org/TR/odrl`, 2002.
12. Y. Ioannidis and T. Sellis. Supporting inconsistent rules in database systems. *Journal of Intelligent Information Systems*, 1(3/4):243–270, 1992.
13. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.
14. T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Research in Security and Privacy*, pages 106–115. IEEE Computer Society Press, 2001.
15. D. Kozen and R. Parikh. An elementary proof of the completeness of PDL. *Theoretical Computer Science*, 14:113–118, 1981.
16. N. Li, B. N. Grosof, and J. Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security (TISSEC)*, 6(1):128–171, 2003.
17. N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, volume 2562 of *Lecture Notes in Computer Science*, pages 58–73, 2003.
18. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Research on Security and Privacy*, pages 114–130, 2002.
19. L. T. McCarty. Permissions and obligations. In *Proceedings of IJCAI-83*, pages 287–294, 1983.
20. E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand, New York, 1964.
21. R. van der Meyden. The dynamic logic of permission. *Journal of Logic and Computation*, 6(3):465–479, 1996.
22. J.-J. C. Meyer. 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, 1988.
23. J.-J. C. Meyer, H. Weigand, and R. Wieringa. A specification language for static, dynamic and deontic integrity constraints. In J. Demetrovics and B. Thalheim, editors, *Mathematical Fundamentals of Database Systems*, volume 346 of *Lecture Notes in Computer Science*, 1989.
24. C. Pfleeger. *Security in Computing*. Prentice-Hall, second edition, 1997.
25. V. Pratt. Process logic. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 93–100. ACM Press, 1979.
26. V. R. Pratt. A practical decision method for propositional dynamic logic. In *Proceedings of the 10th Symposium on Theory of Computing*, pages 326–337. ACM Press, 1978.
27. R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 282–294. IEEE Computer Society Press, 2002.
28. K. Segerberg. A completeness theorem in the modal logic of programs. *Notices AMS*, 24(6):A–552, 1977.
29. R. J. Wieringa and J.-J. C. Meyer. Applications of deontic logic in computer science: A concise overview. In J.-J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, chapter 2, pages 17–40. John Wiley & Sons, 1993.
30. G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.
31. G. H. von Wright. An essay in deontic logic and the general theory of action. In *Acta Phiiosophica Fennica*, volume 21. North Holland, 1968.