A Lower Complexity Bound for Propositional Dynamic Logic with Intersection

Martin Lange

Institut für Informatik, University of Munich

July 13, 2004

Abstract

This paper shows that satisfiability for Propositional Dynamic Logic with Intersection is EXPSPACE-hard. The proof uses a reduction from the word problem for alternating, exponential time bounded Turing Machines.

KEYWORDS: PDL, intersection, satisfiability, complexity

1 Introduction

Propositional Dynamic Logic, PDL, was defined in [4] to reason about program behaviour but is nowadays mainly interesting because of its connection to Description Logics (DL) [16]. It is an extension of multi-modal logic where modalities take as arguments elements of a Kleene Algebra with a test operator.

PDL enjoys nice algorithmic properties: its model checking problem is P-complete and solvable in linear running time [4]; its satisfiability problem is complete for EXPTIME [14, 15]. PDL is embeddable into infinitary multi-modal logic and thus has the tree model property. It also has the finite model property [6]. It is finitely axiomatisable [17, 10]. However, it is rather weak in expressive power since it is strictly less expressive than the alternation-free fragment of Kozen's modal μ-calculus [9].

Several variants of PDL have been studied since, for example restrictions to deterministic atomic programs, etc. Most variants aim at extending the set of operators in the underlying Kleene Algebra in order to allow properties of more programs to be expressed. Examples of these are loop constructs, the converse operator [18], an interleaving operator [12], etc. In most cases axiomatisations and decision procedures for these extension can be obtained by extending PDL's axiom system and its decision procedures.

One variant for which this approach fails entirely is PDL with Intersection, IPDL [7]. Using the connection to Description Logics mentioned above, IPDL can be seen as a DL which can express intersection of roles. Although intersection looks like yet another regular operation it cannot be defined using the operators of a Kleene Algebra. This is simply because of the lack of both tree and finite model property for IPDL which makes it a good candidate for an undecidable logic.

Nevertheless, IPDL is only undecidable if atomic programs are required to be deterministic [7]. If they are allowed to be non-deterministic then its satisfiability problem is decidable [3]. Danecki even showed that it can be decided in double exponential running time.[1] This is proved by constructing Büchi tree automata for IPDL formulas. They do not work on the formula's models directly but on trees describing their models.

These are the only results about IPDL so far. It is not known whether there is a better decision procedure for IPDL or whether it is complete for double exponential time. Complexity issues for modal logics containing the intersection operator have been addressed in [11] for instance. However, there modalities take elements of a boolean algebra rather than a Kleene algebra like PDL does.

Fragments of IPDL have been studied in [5, 13, 1] for instance, mainly regarding the issue of axiomatisability. They also show that the presence of the intersection operator makes IPDL a "strange" logic compared to PDL for example.

In this paper we show that the satisfiability problem for IPDL is hard for exponential space. The proof presents a reduction from the word problem for alternating exponential time bounded Turing Machines. This is inspired by [19] where satisfiability of the temporal logic CTL* is shown to be hard for double exponential time. It remains to be seen whether the lower bound can be pushed up match IPDL's 2-EXPTIME upper bound.

2 Preliminaries

2.1 Propositional Dynamic Logic with Intersection

Let $\mathcal{P} = \{p, q, \ldots\}$ be a finite set of *propositional constants* which includes $\mathtt{tt}$ and $\mathtt{ff}$. Let $\mathcal{A} = \{a, b, \ldots\}$ be a finite set of *atomic program names.* A *Kripke structure* is a triple $(\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ with $\mathcal{S}$ being a set of *states*, $\xrightarrow{a}$ for every $a \in \mathcal{A}$ is a binary relation on states, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ labels the states with propositions, s.t. for all $s \in \mathcal{S}$: $\mathtt{tt} \in L(s)$ and $\mathtt{ff} \notin L(s)$.

Formulas φ and *programs* α of IPDL are defined as:

$$\begin{array}{lcl}
\varphi & ::= & q \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\alpha\rangle\varphi \\
\alpha & ::= & a \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha;\alpha \mid \alpha^* \mid \varphi?
\end{array}$$

[1]However, this piece of information seems to have never made it into common knowledge. Many seem to believe that only decidability but no complexity results are known.

where q ranges over $\mathcal{P}$, and a ranges over $\mathcal{A}$.

We will use the standard abbreviations $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $[\alpha]\varphi := \neg\langle\alpha\rangle\neg\varphi$ and $\alpha^+ := \alpha;\alpha^*$.

IPDL formulas are interpreted over Kripke structures. The semantics of an IPDL formula is explained using an extension of the accessibility relations $\xrightarrow{a}$ to full programs α.

$$\begin{array}{lll}
s \xrightarrow{\alpha;\beta} t & \text{iff} & \exists u \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} u \text{ and } u \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cup\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ or } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cap\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ and } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha^*} t & \text{iff} & \exists n \in \mathbb{N}, s \xrightarrow{\alpha^n} t \text{ where} \\
 & & \forall s,t \in \mathcal{S} : s \xrightarrow{\alpha^0} s, \text{ and } s \xrightarrow{\alpha^{n+1}} t \text{ iff } s \xrightarrow{\alpha;\alpha^n} t \\
s \xrightarrow{\varphi?} s & \text{iff} & s \models \varphi
\end{array}$$

where the meaning of $s \models \varphi$ is explained below.

Assuming a Kripke structure $\mathcal{T}$ to be fixed we define the semantics of a formula φ just as $s \models \varphi$ instead of $\mathcal{T}, s \models \varphi$.

$$\begin{array}{lll}
s \models q & \text{iff} & q \in L(s) \\
s \models \varphi \vee \psi & \text{iff} & s \models \varphi \text{ or } s \models \psi \\
s \models \neg\varphi & \text{iff} & s \not\models \varphi \\
s \models \langle\alpha\rangle\varphi & \text{iff} & \exists t \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} t \text{ and } t \models \varphi
\end{array}$$

2.2 Alternating Turing Machines.

We use the following model of an alternating Turing Machine, which differs slightly from the standard model [2] in the way that it either moves its head or it writes a symbol and branches existentially or universally. It is not hard to see that this model is equivalent to the standard one w.r.t. the standard time and space complexity classes.

An alternating Turing Machine $\mathcal{M}$ is of the form $\mathcal{M} = (Q, \Sigma, q_0, q_{acc}, \delta)$, where Q is the set of states, Σ is the alphabet which contains a blank symbol $\square$, and $q_0, q_{acc} \in Q$ – the starting and the accepting state.

The set Q of states is partitioned into $Q = Q_\exists \cup Q_\forall \cup Q_m$, where we write Q_b for $Q_\exists \cup Q_\forall$, these are the *branching* states. We assume Q_m to contain the only accepting state q_{acc} in which $\mathcal{M}$ simply moves to the left end of the tape whilst staying in state q_{acc}. Having arrived there, it stays on this cell in state q_{acc}.

The transition relation δ is of the form

$$\delta \subseteq \big(Q_b \times \Sigma \times Q \times \Sigma\big) \cup \big(Q_m \times \Sigma \times Q \times \{L, R\}\big) .$$

We also write $(q', b) \in \delta(q, a)$ to denote $(q, a, q', b) \in \delta$ for given q and a.

In a branching state $q \in Q_b$, the machine can overwrite the symbol under the head but not move the head. Furthermore, it can branch nondeterministically or universally. In a state $q \in Q_m$, the machine acts deterministically and moves its head, i.e., for each $a \in \Sigma$, there is exactly one transition $(q, a, q', D) \in \delta$, for $q' \in Q$ and $D \in \{L, R\}$, meaning that the head moves to the left (L) or right (R), and the machine enters state q'.

Every alternating, exponential time bounded Turing Machine can be transformed into such one that still decides its language in alternating exponential time.

A configuration of a Turing Machine is a snapshot in time consisting of the actual state that the machine is in plus the current content of the tape. The initial configuration consists of q_0 and the input word written on the tape followed by blank symbols.

The Turing Machine accepts the input if its initial configuration is accepting. The configurations' acceptance depends on the kind of state:

- If the state is q_{acc} then the configuration is accepting.
- If the state is in $Q_m \setminus \{q_{acc}\}$, then the configuration is accepting iff its unique successor is accepting.
- If the state is in $Q_\exists$, then the configuration is accepting iff at least one of its successors is accepting.
- If the state is in $Q_\forall$, then the configuration is accepting iff all of its successors are accepting.

The entire computation is accepting if the initial configuration is. Note that a witness for an accepting computation can be represented as a tree of configurations with the starting configuration as its root, and where every existential and deterministic configuration has exactly one successor whereas all possible successors of a universal configuration are preserved in the tree.[2]

2.3 Complexity classes

We will quickly recall the definitions of the complexity classes used in this paper. Let DTIME($f(n)$), resp. DSPACE($f(n)$), be the classes of problems that can be decided by a deterministic Turing Machine in time $f(n)$, resp. with space $f(n)$, where n is the size of the input to the machine. The classes we refer to in this paper are those of exponential time, space and double

[2]If computations of alternating Turing Machines are regarded as a game then such a witness is nothing more than a winning strategy for the existential player.

exponential time. They are defined as

$$\begin{aligned}\text{EXPTIME} &:= \bigcup \text{DTIME}(2^{p(n)})\\ \text{EXPSPACE} &:= \bigcup \text{DSPACE}(2^{p(n)})\\ 2-\text{EXPTIME} &:= \bigcup \text{DTIME}(2^{2^{p(n)}})\end{aligned}$$

where $p(n)$ ranges over all polynomials in n. These classes also have characterisations via alternating Turing Machines [2]. The following hold: EXPSPACE = AEXPTIME and 2-EXPTIME = AEXPSPACE.

3 The Reduction

Theorem 1 *Satisfiability of IPDL is EXPSPACE-hard.*

Proof According to [2], there is an alternating, exponential time bounded Turing Machine whose word problem is EXPSPACE-hard. Suppose $\mathcal{M} = (Q, \Sigma, q_0, q_a, \delta)$ is such a Turing Machine that has been tailored so that it obeys the restrictions laid out in the previous section. Let $w = a_0 \dots a_{n-1} \in \Sigma^*$ be an input for $\mathcal{M}$. W.l.o.g. we assume the space needed by $\mathcal{M}$ on input w to be bounded by $2^{p(n)}$ for some polynomial p. Let $N := 2^{p(n)} - 1$ be the maximal index of a tape cell.

In the following we will construct an IPDL formula $\varphi_{\mathcal{M},w}$ over a singleton set $\mathcal{A} = \{x\}$ of atomic programs s.t. $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable. The letter x indicates "moving to the next state". Informally, a witness for an accepting computation of $\mathcal{M}$ on w will serve as a model for $\varphi_{\mathcal{M},w}$.

The following propositions are needed.

$$\mathcal{P} = Q \cup \Sigma \cup \{c_0, \dots, c_{p(n)-1}\} \cup \{d_0, \dots, d_{p(n)-1}\}$$

- $q \in Q$ is true in a state of the model iff the head of $\mathcal{M}$ is on the corresponding tape cell in the corresponding configuration while the machine is in state q. The formula $h := \bigvee_{q \in Q} q$ says that the tape head is on the current cell.

- $a \in \Sigma$ is true iff a is the symbol on the corresponding tape cell.

- $c_{p(n)-1}, \dots, c_0$ represent a counter C in binary coding. The counter value will be 0 in the leftmost and N in the rightmost tape cell for instance. Let $P := p(n)-1$ be the index of the most significant counter bit.

- $d_{p(n)-1}, \dots, d_0$ represent a counter D. The value of this counter will be the index of the actual configuration (i.e. the distance to the starting configuration in $\mathcal{M}$'s computation tree).

The encoding of $\mathcal{M}$'s configurations is very similar to the ones presented in [19] or [8]. A configuration of the form

a_0	a_1	a_2	$\dots$	a_N

with the tape head on, say, the third cell from the left and the machine in state q is modelled by a sequence of states of the form

$$\begin{tikzcd}
 & & & & c_P \\
 & & & & \vdots \\
 & & q & & \\
 & c_0 & c_1 & & c_0 \\
a_0 & a_1 & a_2 & & a_N \\
\bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] & \dots \arrow[r, "x"] & \bullet
\end{tikzcd}$$

Here, the counter bits for D are left out to avoid clutter. Successive configurations are modelled by concatenating these sequences.

For every fixed $m \in \{0, \dots, N\}$ we can write a formula $\chi_{C=m}$, resp. $\chi_{D=m}$, which says that the counter value of C or D is m in the current state, e.g.

$$\chi_{C=0} \ := \ \bigwedge_{i=0}^{P} \neg c_i \ , \ \chi_{C=1} \ := \ c_0 \wedge \bigwedge_{i=1}^{P} \neg c_i \ \text{ and } \ \chi_{C=N} \ := \ \bigwedge_{i=0}^{P} c_i$$

for the leftmost ($m = 0$), second ($m = 1$) and rightmost ($m = N$) position in a configuration.

For the next part we need auxiliary formulas $\varphi^C_{inc}, \varphi^D_{inc}$ that increase the value of C, resp. D, as long as they do not equal N.

$$\begin{aligned}
\varphi^C_{inc} \ := \ \bigvee_{i=0}^{P} (\ \neg c_i \ \wedge \ [x]c_i \ \wedge \ \bigwedge_{j<i} (c_j \wedge [x]\neg c_j) \ \wedge \\
\bigwedge_{j>i} (c_j \rightarrow [x]c_j) \wedge (\neg c_j \rightarrow [x]\neg c_j) \ )
\end{aligned}$$

This is just the standard formula for incrementing a binary value: there is a bit which changes from 0 to 1, all higher bits remain the same and all lower bits change from 1 to 0. We also need to be able to say that the value of D remains unchanged.

$$\varphi^D_{remain} \ := \ \bigwedge_{i=0}^{P} (\ d_i \rightarrow [x]d_i \ ) \ \wedge \ (\ \neg d_i \rightarrow [x]\neg d_i \ )$$

Then we can write down a formula which requires the counter C to be increased by one modulo $N + 1$ in every move from a state to a successor.

At the same time, counter D is incremented iff the value of C changes from N back to 0.

$$\begin{aligned}\varphi_{count} \;:=\; [x^*](&\; (\; \chi_{C=N} \;\wedge\; [x]\chi_{C=0} \;\wedge\; \varphi^D_{inc} \;) \;\vee \\ &\; (\; \neg\chi_{C=N} \;\wedge\; \varphi^C_{inc} \;\wedge\; \varphi^D_{remain} \;) \;)\end{aligned}$$

We form programs

- α_{rest} which goes from any state to the beginning of the next configuration, i.e. it traverses the rest of the current configuration,

$$\alpha_{rest} \;:=\; (\neg\chi_{C=N}?; x)^*; \chi_{C=N}?$$

- α_{2h}, α_{0h} which do an arbitrary amount of x-actions whilst seeing at least two, resp. no tape heads.

$$\begin{aligned}\alpha_{2h} \;&:=\; x^*; h?; x^+; h?; x^* \\ \alpha_{0h} \;&:=\; (\neg h?; x)^*; \neg h?\end{aligned}$$

Then we can formalise the general requirements on a Turing Machine: every tape cell is marked with exactly one symbol from Σ and never with two different states; no configuration has more or less than one cell marked with the tape head. Finally, as long as the counters both do not have the value N there is still a successor.

$$\begin{aligned}\varphi_{gen} \;:=\; [x^*](&\; (\bigvee_{a\in\Sigma} a \;) \;\wedge\; \bigwedge_{a,b\in\Sigma, b\neq a} \neg(a \wedge b) \;\wedge\; \bigwedge_{q,q'\in Q, q\neq q'} \neg(q \wedge q') \\ &\wedge\; (\; \chi_{C=0} \;\rightarrow\; [\alpha_{rest} \cap (\alpha_{2h} \cup \alpha_{0h})]\mathtt{ff} \;) \\ &\wedge\; (\; \neg\chi_{C=N} \wedge \neg\chi_{D=N} \;\rightarrow\; \langle x \rangle \mathtt{tt} \;) \;)\end{aligned}$$

At the beginning, the input word $w = a_0 \ldots a_{n-1}$ is written on the tape, followed by blank symbols $\Box$ until a state with counter value 0 is reached again.

$$\begin{aligned}\varphi_{start} \;:=\; &\chi_{C=0} \;\wedge\; q_0 \;\wedge\; a_0 \;\wedge \\ &[x](\; a_1 \;\wedge \\ &\quad [x](\; a_2 \;\wedge \\ &\qquad \ldots \;\wedge\; \ldots \\ &\qquad\quad [x](\; a_{n-1} \;\wedge \\ &\qquad\qquad [(x; \neg\chi_{C=0}?)^+]\Box \;) \ldots))\end{aligned}$$

Next we give a formula which expresses the fact that $\mathcal{M}$'s computation is accepting. Note that we assumed $\mathcal{M}$ to move its head to the very left, go into state q_a, and leave the head there once its computation is finished.

$$\varphi_{acc} \;:=\; [x^*; \chi_{C=0}?; \chi_{D=N}?]q_{acc}$$

Before we can encode $\mathcal{M}$'s transition function δ, we need to write programs that relate a tape cell in one configuration to itself or its neighbours in the next configuration. Program $\alpha_{0!}$ runs the atomic program x arbitrarily often whilst seeing the counter value $C = 0$ only once.

$$\alpha_{0!} \;:=\; (x; \neg\chi_{C=0}?)^*; x; \chi_{C=0}?; (x; \neg\chi_{C=0}?)^*$$

Using this program and the usual trick of incrementing, resp. decrementing a binary counter we can write a program α_{-1} that turns a tape cell into its left neighbour in the following configuration.

$$\begin{aligned}\alpha_{-1} \;:=\; \neg\chi_{C=0}?; \alpha_{0!} \cap \bigcup_{i=0}^{P}\Big(& c_i?; x^+; \neg c_i? \;\cap\; \bigcap_{j<i} \neg c_j?; x^+; c_j? \\ & \cap \bigcap_{j>i} (\, c_j?; x^+; c_j? \cup \neg c_j?; x^+; \neg c_j?\,)\Big)\end{aligned}$$

Equally, $\alpha_=$ and α_{+1} turn it into itself, resp. its right neighbour.

$$\begin{aligned}\alpha_= \;&:=\; \alpha_{0!} \cap \bigcap_{i=0}^{P} (\, c_i?; x^+; c_i? \;\cup\; \neg c_i?; x^+; \neg c_i?\,) \\ \alpha_{+1} \;&:=\; \neg\chi_{C=N}?; \alpha_=; x\end{aligned}$$

Now we are able to formalise δ's transitions in IPDL. Again, there are different requirements on the model depending on the nature of a machine's state.

In an existential state, there is one transition that determines all successors. In a universal state, all successors behave according to one of the possible transitions and every possible transition is present in the computation tree. In a moving state the machine acts deterministically, hence, all successor configurations must be the same. Finally, the label of any tape cell which is not under the tape head remains the same in the following configuration.

$$\begin{aligned}\varphi_\delta \;:=\; [x^*]\Big(& \bigwedge_{q\in Q_\exists, a\in\Sigma} (\, q \wedge a \;\rightarrow \bigvee_{(p,b)\in\delta(q,a)} [\alpha_=](p \wedge b)\,) \\ & \wedge \bigwedge_{q\in Q_\forall, a\in\Sigma} (\, q \wedge a \;\rightarrow\; (\, [\alpha_=] \bigvee_{(p,b)\in\delta(q,a)} (p \wedge b)\,) \;\wedge \bigwedge_{(p,b)\in\delta(q,a)} \langle\alpha_=\rangle(p \wedge b)\,)\,) \\ & \wedge \bigwedge_{(q,a,q',L)\in\delta} (\, \neg\chi_{C=0} \wedge q \wedge a \;\rightarrow\; [\alpha_{-1}]q'\,)\end{aligned}$$

$$\wedge \bigwedge_{(q,a,q',R)\in\delta} (\ \neg\chi_{C=N} \wedge q \wedge a \ \rightarrow\ [\alpha_{+1}]q'\ )$$

$$\wedge \bigwedge_{a\in\Sigma} (\ \neg h \wedge a \ \rightarrow\ [\alpha_{=}]a\ )\ )$$

Altogether, the machine's behaviour is described by the formula

$$\varphi_{\mathcal{M},w} \ := \ \varphi_{count} \wedge \varphi_{gen} \wedge \varphi_{start} \wedge \varphi_{acc} \wedge \varphi_{\delta}$$

Then, a model for $\varphi_{\mathcal{M},w}$ bears a witness for a successful computation of $\mathcal{M}$ on w. Conversely, each successful computation can be transformed into a model for $\varphi_{\mathcal{M},w}$ by removing the nondeterminism from the model and keeping the universal branches. Finally, $|\varphi_{\mathcal{M},w}|$ is polynomial in $|\mathcal{M}|$ and $|w|$. ■

Corollary 2 *Satisfiability of IPDL over a singleton set of atomic programs and tests restricted to atomic propositions is already EXPSPACE-hard.*

This is not surprising since every IPDL formula φ with complex tests can be transformed into a φ' over additional propositions, s.t. φ' only features atomic tests. Moreover, φ' is satisfiable iff φ is satisfiable. On the other hand, the reduction in the proof of Theorem 1 can easily be rewritten s.t. all tests are atomic using for instance the equivalence

$$(\bigvee_{q\in Q} q)? \ \equiv \ \bigcup_{q\in Q} q?$$

Corollary 3 *Satisfiability of IPDL is 2-EXPTIME-hard under EXPTIME-reductions.*

Proof According to [2], there is an alternating, exponential space bounded Turing Machine whose word problem is hard for double exponential time and polynomial time reductions. If the reduction is allowed to take exponential time then one can use exponentially many counter bits for D and make the reduction go through for an AEXPSPACE machine. Note that with exponentially many counter bits one can count up to $2^{2^{p(n)}}$ which – with the right choice of p – is the maximal number of different configurations an AEXPSPACE machine can be in. ■

Corollary 4 *Satisfiability for IPDL is already EXPSPACE-hard over the class of trees.*

Proof Take $\mathcal{M}$, w and $\varphi_{\mathcal{M},w}$ from the proof of Theorem 1. Note that the representation of a witness for an accepting run of $\mathcal{M}$ on w is a tree since only one atomic program is used. Hence, formula $\varphi_{\mathcal{M},w}$ has the tree model property which is not true for arbitrary IPDL formulas. ■

4 Conclusions

This is the first step towards closing the complexity gap for satisfiability of IPDL. It remains to be seen whether this lower bound can be improved in order to achieve 2-EXPTIME-completeness or Danecki's upper bound can be improved in order to obtain EXPSPACE-completeness.

The other main question that remains open is the exact complexity (i.e. both upper and lower bound) for test-free IPDL. The proof of the lower bound presented here relies heavily on the presence of the test operator. A similar reduction might work for test-free IPDL, but then the encoding of a model would have to be altered.

Acknowledgments I would like to thank Carsten Lutz for valuable suggestions concerning the improvement of this paper. For example, he pointed out Corollary 4. He also has found a simpler reduction using nondeterministic, exponential space bounded Turing Machines. I would also like to thank the two anonymous referees as well as Jan Johannsen and Stéphane Remri for their comments and suggestions.

References

[1] P. Balbiani and D. Vakarelov. Iteration-free PDL with intersection: a complete axiomatization. *Fundamenta Informaticae*, 45(3):173–194, February 2001.

[2] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.

[3] S. Danecki. Nondeterministic propositional dynamic logic with intersection is decidable. In A. Skowron, editor, *Proc. 5th Symp. on Computation Theory*, volume 208 of *LNCS*, pages 34–53, Zaborów, Poland, December 1984. Springer.

[4] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, April 1979.

[5] R. I. Goldblatt and S. K. Thomason. Axiomatic classes in propositional modal logic. In J. N. Crossley, editor, *Algebra and Logic: Papers 14th Summer Research Inst. of the Australian Math. Soc.*, volume 450 of *Lecture Notes in Mathematics*, pages 163–173. Springer, 1975.

[6] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[7] David Harel. Recurring dominoes: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

[8] J. Johannsen and M. Lange. CTL$^+$ is complete for double exponential time. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. 30th Int. Coll. on Automata, Logics and Programming, ICALP'03*, volume 2719 of *LNCS*, pages 767 – 775, Eindhoven, NL, June 2003. Springer.

[9] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.

[10] D. Kozen and R. Parikh. An elementary proof of the completeness of PDL (note). *TCS*, 14:113 – 118, 1981.

[11] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logic. In *Advances in Modal Logic 2000 (AiML 2000)*, Leipzig, Germany, 2000. Final version appeared in Advances in Modal Logic Volume 3, 2001.

[12] A. J. Mayer and L. J. Stockmeyer. The complexity of PDL with interleaving. *TCS*, 161(1–2):109–122, 15 July 1996.

[13] S. Passy and T. Tinchev. An essay in combinatory dynamic logic. *Information and Computation*, 93(2):263–332, 1991.

[14] V. R. Pratt. A practical decision method for propositional dynamic logic. In *Proc. 10th Symp. on Theory of Computing, STOC'78*, pages 326–337, San Diego, California, May 1978.

[15] V. R. Pratt. Models of program logics. In *Proc. 20th Symp. on Foundations of Computer Science, FOCS'79*, pages 115–122. IEEE, 1979.

[16] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. 12th Int. Joint Conf. on Artificial Intelligence, IJCAI'91*, pages 466–471, Sydney, Australia, August 1991. Morgan Kaufmann.

[17] K. Segerberg. A completeness theorem in the modal logic of programs. *Notices of the AMS*, 24(6):A–552, October 1977.

[18] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, July 1982.

[19] M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, May 1985. ACM.