A Lower Complexity Bound for Propositional Dynamic Logic with Intersection

Martin Lange

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

1 Introduction

Propositional Dynamic Logic, PDL, was defined in [5] to reason about program behaviour but is nowadays mainly interesting because of its connection to other logics like Description Logics [18] or Information Logics [15, 4] for example. 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 PTIME-complete and solvable in linear running time, its satisfiability problem is complete for EXPTIME [5, 16, 17]. PDL is embeddable into infinitary multi-modal logic and thus has the tree model property. It also has the finite model property [8]. It is finitely axiomatisable [19, 11]. 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 [10].

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 [20], an interleaving operator [13], 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 Description Logic which can express intersection of roles. Although intersection looks like yet another regular operation

Advances in Modal Logic, Volume 5. © 2005, the author.

it cannot be defined using the operators of a Kleene Algebra. This is because it does not retain the tree model property. In addition, it also does not have the finite model property 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 showed that it can be decided in double exponential time. 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 these models. The size of an automaton obtained from a formula is doubly exponential in the formula's size, and the emptiness problem for non-deterministic Büchi automata is solvable in polynomial time.

These are the only complexity 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 [12] 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 [6, 14, 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 instance, it loses both the tree model property and the finite model property.

In this paper we show that the satisfiability problem for IPDL is hard for exponential space. The proof consists of a reduction from the word problem for alternating exponential time bounded Turing Machines. This is inspired by [21] where satisfiability of the temporal logic CTL* is shown to be hard for double exponential time.

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 *points*, $\xrightarrow{a}$ for every $a \in \mathcal{A}$ is a binary relation on points, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ labels the points 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 follows:

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

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. Given a Kripke structure $\mathcal{T}$, the semantics of an IPDL formula is defined by simultaneous induction on the structure of programs and formulas:

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

where

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

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. W.l.o.g. we assume that $q_0 \neq q_{acc}$.

Q is partitioned into $Q_\exists \cup Q_\forall \cup Q_m \cup \{q_{acc}\}$, where we write Q_b for $Q_\exists \cup Q_\forall$, these are the *branching* states. We assume that $\mathcal{M}$, in order to accept an input, moves to the left end of the tape once its computation is finished and enters 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 non-deterministically 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'.

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

$\mathcal{M}$ accepts the input w if its initial configuration is accepting. This is denoted $w \in L(\mathcal{M})$. The configurations' acceptance behaviour 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 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 present in the tree.[1]

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. Equally, let ATIME($f(n)$) and ASPACE($f(n)$) be the classes of problems that can be decided by an alternating Turing Machine in time $f(n)$, resp. with space $f(n)$.

[1] 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.

The classes we refer to are those of exponential time, space and double 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. Furthermore, the classes of problemss decidable in alternating exponential time, resp. space, are

$$\begin{aligned}
\text{AEXPTIME} &:= \bigcup \text{ATIME}(2^{p(n)}) \\
\text{AEXPSPACE} &:= \bigcup \text{ASPACE}(2^{p(n)})
\end{aligned}$$

THEOREM 1. [2] *AEXPTIME = EXPSPACE, AEXPSPACE = 2-EXPTIME.*

3 The Reduction

Let $\mathcal{M} = (Q, \Sigma, q_0, q_{acc}, \delta)$ be an alternating and exponential time bounded Turing Machine. Assume that it has been tailored to obey the restrictions laid out in the previous section. Let $w = a_0 \ldots a_{n-1} \in \Sigma^*$ be an input for $\mathcal{M}$. We assume the space used by $\mathcal{M}$ on 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 used in the computation, and $P := p(n) - 1$ the maximal index of a bit needed for a counter that can store values between 0 and N.

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. Informally, a witness for an accepting computation of $\mathcal{M}$ on w will serve as a model for $\varphi_{\mathcal{M},w}$.

The encoding of $\mathcal{M}$'s configurations is very similar to the ones presented in [21] or [9]. A configuration of the form

$$\begin{array}{|c|c|c|c|c|}\hline a_0 & a_1 & a_2 & \qquad\ldots\qquad & a_N \\ \hline\end{array}$$

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

$$\begin{array}{ccccccccc}
 & & & & & & & & c_P \\
 & & & & & & & & \vdots \\
 & & & & q & & & & \\
 & & c_0 & & c_1 & & & & c_0 \\
a_0 & & a_1 & & a_2 & & & & a_N \\
\bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \ldots & \xrightarrow{x} & \bullet
\end{array}$$

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

The following propositions are used.

$$\mathcal{P} = Q \cup \Sigma \cup \{c_0, \ldots, c_P\} \cup \{d_0, \ldots, d_P\}$$

- $q \in Q$ 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 in the corresponding configuration.
- $c_P, \ldots, c_0$ 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.
- $d_P, \ldots, d_0$ 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).

For every fixed $m \in \{0, \ldots, N\}$ 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: for some $i \in \{0, \ldots, P\}$ the i-th bit 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.

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

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.

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

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 assume $\mathcal{M}$ to move its head to the very left, go into state q_{acc}, and leave the head there once its computation is finished.

$$\varphi_{acc} \ := \ [x^*](\chi_{C=0} \wedge \chi_{D=N} \rightarrow q_{acc})$$

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 relates a tape cell with 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. In all cases we need to require the existence of a successor configuration, though.

Finally, the label of any tape cell which is not under the tape head remains the same in the following configuration.

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

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

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

LEMMA 2. *Let $\mathcal{M}$ be an alternating, exponential time bounded Turing machine, and $w\in\Sigma^*$. If $w\in L(\mathcal{M})$ then $\varphi_{\mathcal{M},w}$ is satisfiable.*

Proof. Suppose there is a successful computation of $\mathcal{M}=(Q,\Sigma,q_0,q_{acc},\delta)$ on $w=a_0\dots a_{n-1}$, and assume that $\mathcal{M}$ decides w in time $2^{p(n)}$ for some polynomial p. The computation can be represented by a tree of configurations with the starting configuration C_0 as the root, and successor configurations as sons of a configuration in the tree. Let $\mathcal{C}$ be the set of all configurations occurring in this computation. We write $C'\in Succ(C)$ to denote that C' is a successor of C in this tree.

Since $\mathcal{M}$'s computation time is bounded by $2^{p(n)}$, so is its space. Hence, the tree of configurations can be refined into a tree of tape cells with each cell being the son of its left neighbour. The leftmost tape cell is the son of the rightmost tape cell of the previous configuration.

Formally, let $V:=\{(C,i)\mid C\in\mathcal{C}, i\in\{0,\dots,2^{p(n)}-1\}\}$. Define a binary relation $\xrightarrow{x}$ on V as

$$(C,i)\xrightarrow{x}(C',j) \text{ iff } \begin{cases} C=C', j=i+1, & \text{if } i<2^{p(n)}-1 \\ C'\in Succ(C), j=0, & \text{if } i=2^{p(n)}-1 \end{cases}$$

which can be extended to programs over x in the usual way.

Finally, define a labeling function $L : V \to 2^{\mathcal{P}}$ satisfying the following and not more.

$a \in L(C, i)$	if the symbol on cell i in configuration C is a
$q \in L(C, i)$	if the head is on cell i in configuration C and the machine is in state q
$c_j \in L(C, i)$	if the j-th bit of i in binary coding is 1
$d_j \in L(C, i)$	if there are $n \leq 2^{p(n)} - 1,$ and $C_1, \ldots, C_n \in \mathcal{C}$ s.t. $C = C_n$ and for all $k = 0, \ldots, n-1 : C_{k+1} \in \mathit{Succ}(C_k)$, and the j-th bit of n in binary coding is 1

Now let $T := (V, \{\xrightarrow{x}\}, L)$. We will show that $T, (C_0, 0) \models \varphi_{\mathcal{M},w}$. Remember that $\varphi_{\mathcal{M},w}$ consist of the following five conjuncts: (1) φ_{count} to identify tape cells, (2) φ_{start} to describe the starting configuration, (3) φ_{gen} to describe the general requirements on a Turing Machine, (4) φ_{acc} to describe the acceptance of the input, and (5) φ_δ for the correct evolving of the configurations.

(1) Note that φ_{count} formalises binary counting modulo $2^{p(n)}$ which is exactly what happens along each path of T by the definition of $\xrightarrow{x}$. Also, for every C and every i: $T, (C, i) \models \chi_{C=i}$. Now, if $(C, i) \xrightarrow{x} (C', j)$ then either $C = C'$, $i < 2^{p(n)} - 1$, and $j = i + 1$, or $C' \in \mathit{Succ}(C)$, $i = 2^{p(n)} - 1$, and $j = 0$. Furthermore, if $C' \in \mathit{Succ}(C)$ then $T, (C, i) \models \chi_{D=k}$ for some k, and $T, (C', j) \models \chi_{D=k+1}$. Otherwise $T, (C', j) \models \chi_{D=k}$.

Thus, for all (C, i) we have $T, (C, i) \models \chi_{C=N} \wedge [x]\chi_{C=0} \wedge \varphi^D_{inc}$ or $T, (C, i) \models \neg\chi_{C=N} \wedge \varphi^C_{inc} \wedge \varphi^D_{remain}$, therefore

$$T, (C_0, 0) \models \varphi_{count}$$

(2) We have $T, (C_0, 0) \models \varphi_{start}$ because along each path of T, the first $2^{p(n)}$ tape cells form the starting configuration.

(3) We have $T, (C_0, 0) \models \varphi_{gen}$ because by the construction of L and $\xrightarrow{x}$,

- every (C, j) is labeled with exactly one tape symbol and no more than one machine state;
- from any state $(C, 0)$ one cannot reach a state $(C, 2^{p(n)} - 1)$ whilst seeing less or more than one state carrying a label q. Hence, for all (C, i) we have

$$T, (C, i) \models \chi_{C=0} \to [\alpha_{rest} \cap (\alpha_{2h} \cup \alpha_{0h})]\mathtt{ff}$$

(4) Every path through T ends in a sequence $(C, 0), \ldots, (C, 2^{p(n)} - 1)$ of states s.t. $d \in L(C, i)$ for all $i = 0, \ldots, 2^{p(n)} - 1$. By assumption, $\mathcal{M}$ has moved its head to the left end of the tape and entered state q_{acc}. Hence, for all (C, i) we have

$$T, (C, i) \models \chi_{C=0} \wedge \chi_{D=N} \rightarrow q_{acc}$$

(5) What remains to be seen is that $T, (C_0, 0) \models \varphi_\delta$. Take any $C \in \mathcal{C}$ and any $i \in \{0, \ldots, 2^{p(n)} - 1\}$. Suppose that $q \in L(C, i)$ for some $q \in Q$. Note that

- $(C, i) \xrightarrow{\alpha_=} (C', j)$ iff $C' \in \mathit{Succ}(C)$ and $i = j$;
- $(C, i) \xrightarrow{\alpha_{+1}} (C', j)$ iff $C' \in \mathit{Succ}(C)$, $i < 2^{p(n)} - 1$ and $j = i + 1$;
- $(C, i) \xrightarrow{\alpha_{-1}} (C', j)$ iff $C' \in \mathit{Succ}(C)$, $i > 0$ and $j = i - 1$.

Now, there are three possibilities.

(a) $q \in Q_\exists$. Then C has exactly one successor configuration C' in the computation tree, i.e. $\mathit{Succ}(C) = \{C'\}$. Furthermore, the symbol a at cell i is replaced by b for some $(p, b) \in \delta(q, a)$ and the tape head is not moved. Hence, $T, (C', i) \models p \wedge b$.

(b) $q \in Q_\forall$. This is similar to case (a). However, C has several successor configurations which are all modeled in T. Here, for any (C', i) with $C' \in \mathit{Succ}(C)$ we have $T, (C', i) \models p \wedge b$ for some $(p, b) \in \delta(q, a)$. Since T arose from the computation tree of $\mathcal{M}$, every $(p, b) \in \delta(q, a)$ is covered by some branch in T.

(c) $q \in Q_m$. Here, only the tape head is moved. C has one successor C'. If $(p, L) \in \delta(q, a)$ then $p \in L(C', i - 1)$ and accordingly for $(p, R) \in \delta(q, a)$. Finally, the remaining conjuncts in φ_δ are also satisfied because for any C, C' with $C' \in \mathit{Succ}(C)$ and any i we have: if $Q \cap L(C, i) = \emptyset$ then $T, (C, i) \models a$ implies $T, (C', i) \models a$ for any $a \in \Sigma$. ■

LEMMA 3. *Let $\mathcal{M}$ be an alternating exponential time bounded Turing Machine and $w \in \Sigma^*$. Then $\varphi_{\mathcal{M},w}$ has the tree model property.*

Proof. Note that PDL without the intersection operator has the tree model property. Also, $\varphi_{\mathcal{M},w}$ is built from one single atomic program x only. Hence, a sufficient condition for the tree model property of $\varphi_{\mathcal{M},w}$ is the fact that within each subformula of the form $\langle\alpha\rangle\psi$ in a positive position – or $[\alpha]\psi$ in a negative position – there is no subprogram β that cannot be realised by a sequence $s_0, \ldots, s_n$ of points s.t. $s_{i-1} \xrightarrow{x} s_i$ for all $i = 1, \ldots, n$.

Note that the only programs occurring in such a position are $\alpha_=$, α_{-1} and α_{+1}. Take $\alpha_=$ for instance, and a point s_0. Now $\alpha_=$ can be realised by the sequence $s_0, \ldots, s_n$ for any $n > 0$ if only the following hold:

- for all $k = 0, \ldots, 2^{p(n)} - 1$: $s_0 \models \chi_{C=k}$ iff $s_n \models \chi_{C=k}$,
- there is exactly one $i \in \{1, \ldots, n\}$ s.t. $s_i \models \chi_{C=0}$.

Obviously, these requirements are not unsatisfiable. Given that $\alpha_{+1} = \alpha_{=}; x$, the same holds for α_{+1}, too. For α_{-1} it can be shown in a similar way.

Then a standard model construction procedure for PDL can be employed in order to build a tree model for $\varphi_{\mathcal{M},w}$. ■

LEMMA 4. *Let $\mathcal{M}$ be an alternating, exponential time bounded Turing machine, and $w \in \Sigma^*$. If $w \notin L(\mathcal{M})$ then $\varphi_{\mathcal{M},w}$ is unsatisfiable.*

Proof. Suppose that $\mathcal{M} = (Q, \Sigma, q_0, \delta, q_{acc})$ and $w = a_0 \ldots a_{n-1}$. Let $\varphi' := \varphi_{count} \wedge \varphi_{gen} \wedge \varphi_{start} \wedge \varphi_\delta$. First we need to see something similar to the converse of Lemma 2: a model for φ' bears a computation (not necessarily successful) for $\mathcal{M}$ on w. Suppose there is an $M = (V, \xrightarrow{x}, L)$ with a $v \in V$ s.t. $M, v \models \varphi_{\mathcal{M},w}$. According to Lemma 3 we can assume M to be a tree.

Let v be the root of M. Since $M, v \models \varphi_{start}$ we have $L(v) = \{q, a_1\}$ for some $q \in Q$. But then either $q \in Q_\exists$, $q \in Q_\forall$, $q \in Q_m$, or $q = q_{acc}$. The last case is excluded which will become clear later. In any other case, since we also have $M, v \models \varphi_\delta$, there must be at least one v' s.t. $v \xrightarrow{\alpha} v'$ with $\alpha \in \{\alpha_=, \alpha_{-1}, \alpha_{+1}\}$. $M, v \models \varphi_{count}$ ensures that between v and v' there are $2^{p(n)}$, resp. $2^{p(n)} + 1$ atomic x-transitions, hence, there are states $v_0, \ldots, v_{2^{p(n)}-1}$ with $v = v_0$. Now, $M, v \models \varphi_{start}$ ensures that these states are labeled successively with the input word w followed by blank symbols. $M, v \models \varphi_{gen}$ ensures that no other labelings, including machine states can occur along this sequence. Then $v_0, \ldots, v_{2^{p(n)}-1}$ can be regarded as the starting configuration of $\mathcal{M}$.

Note that there is a $q' \in Q$ s.t. $M, v' \models q'$. The above argument can now be iterated forming further configurations of $\mathcal{M}$. It is important to note that a state v is required to have an α-successor – $\alpha \in \{\alpha_=, \alpha_{-1}, \alpha_{+1}\}$ – iff there is a $q \in Q$ with $M, v \models q$. Therefore, the building of configurations can be continued until a tree of configurations is obtained. Depending on the nature of each q, several different successors may have to be found in M.

Hence, we have $M, v_0 \models \varphi'$ and M can be regarded as a computation tree of $\mathcal{M}$ on w. But $w \notin L(\mathcal{M})$ by assumption. Therefore, there must be at least one path through the computation tree that does not contain an accepting configuration. In other words, $M, v_0 \models \varphi'$ implies $M, v_0 \models \neg\varphi_{acc}$. Thus, $\varphi_{\mathcal{M},w}$ is unsatisfiable. ■

LEMMA 5. *Let $\mathcal{M}$ be an alternating, exponential time bounded Turing machine. For any $w \in \Sigma^*$: $|\varphi_{\mathcal{M},w}|$ is polynomial in $|w|$ and $|\mathcal{M}|$.*

Proof. Let $\mathcal{M} = (Q, \Sigma, q_0, \delta, q_{acc})$ and $w \in \Sigma^*$ s.t. $\mathcal{M}$ decides each such w in time $2^{p(|w|)}$ for some polynomial p. Consider the five conjuncts that make up $\varphi_{\mathcal{M},w}$. We have

- $|\varphi_{count}| = O(p(|w|)^2)$,
- $|\varphi_{gen}| = O(|\Sigma|^2 + |Q|^2 + p(|w|))$,
- $|\varphi_{start}| = O(|w|)$,
- $|\varphi_{acc}| = O(p(|w|))$,
- $|\varphi_\delta| = O(|Q|^2 \cdot |\Sigma|^2 \cdot p(|w|) + |Q| \cdot |\Sigma|)$.

■

THEOREM 6. *Satisfiability of IPDL is EXPSPACE-hard.*

Proof. According to Theorem 1 there is an alternating, exponential time bounded Turing Machine $\mathcal{M}$ whose word problem is EXPSPACE-hard. Let w be an input to $\mathcal{M}$, and assume that $\mathcal{M}$ decides every such w in time $2^{p(|w|)}$ for some polynomial p. The construction above yields an IPDL formula $\varphi_{\mathcal{M},w}$ whose size is polynomial in $|\mathcal{M}|$ and $|w|$ according to Lemma 5. Lemmas 2 and 4 show that that $\varphi_{\mathcal{M},w}$ is satisfiable iff $w \in L(\mathcal{M})$. ■

COROLLARY 7. *Satisfiability of IPDL over a singleton set of atomic programs and tests restricted to literals is already EXPSPACE-hard.*

Proof. Given $\mathcal{M}$ and w, it is possible to rewrite $\varphi_{\mathcal{M},w}$ s.t. the test operator is only applied to atomic propositions or their negations. Note that the only tests on non-literals occurring in $\varphi_{\mathcal{M},w}$ are $h?, \neg h?, \chi_{D=N}?, \chi_{C=k}?$, and $\neg\chi_{C=k}?$ for some $k \in \{0, 2^{p(n)} - 1\}$. They can be rewritten as programs over atomic tests in the following way:

- $h? := \bigcup_{q \in Q} q?$
- $\neg h? := \neg q_0?; \neg q_1?; \ldots; \neg q_m?$ if $Q = \{q_0, \ldots, q_m\}$
- $\chi_{C=0} := \neg c_0?; \ldots; \neg c_P?$
- $\neg\chi_{C=0} := \bigcup_{i=0}^{P} c_i?$

and similarly for $\chi_{C=N}?, \neg\chi_{C=N}?$ and $\chi_{D=N}?$. ■

COROLLARY 8. *Satisfiability of IPDL is 2-EXPTIME-hard under EXPTIME-reductions.*

Proof. According to Theorem 1 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 have. ■

COROLLARY 9. *Satisfiability for IPDL is already EXPSPACE-hard over the class of trees.*

Proof. Immediately from Lemma 3. ■

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 9. He also has found a simpler reduction using non-deterministic, exponential space bounded Turing Machines. However, there is reason to doubt that non-deterministic machines can be used to show 2-EXPTIME-hardness of IPDL.

I would also like to thank the two anonymous referees as well as Jan Johannsen and Stéphane Demri for their comments and suggestions.

BIBLIOGRAPHY

[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] S. Demri and E. Orłowska. *Incomplete Information: Structure, Inference, Complexity.* EATCS Monographs. Springer, 2002.
[5] 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.
[6] 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.
[7] D. Harel. Recurring dominoes: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.
[8] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, 2000.
[9] 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.
[10] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.
[11] D. Kozen and R. Parikh. An elementary proof of the completeness of PDL (note). *TCS*, 14:113 – 118, 1981.
[12] 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.
[13] A. J. Mayer and L. J. Stockmeyer. The complexity of PDL with interleaving. *TCS*, 161(1–2):109–122, 15 July 1996.
[14] S. Passy and T. Tinchev. An essay in combinatory dynamic logic. *Information and Computation*, 93(2):263–332, 1991.
[15] Z. Pawlak. Information systems theoretical foundations. *Information Systems*, 6(3):205–218, 1981.
[16] 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.
[17] V. R. Pratt. Models of program logics. In *Proc. 20th Symp. on Foundations of Computer Science, FOCS'79*, pages 115–122. IEEE, 1979.
[18] 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.
[19] K. Segerberg. A completeness theorem in the modal logic of programs. *Notices of the AMS*, 24(6):A–552, October 1977.
[20] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, July 1982.
[21] 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.

Martin Lange
University of Munich, Institut für Informatik
Oettingenstr. 67, D-80538 München, Germany
`Martin.Lange@ifi.lmu.de`