THE JOURNAL OF SYMBOLIC LOGIC
Volume 00, Number 0, XXX 0000

2-EXPTIME LOWER BOUNDS FOR PROPOSITIONAL DYNAMIC LOGICS WITH INTERSECTION

MARTIN LANGE AND CARSTEN LUTZ

Abstract. In 1984, Danecki proved that satisfiability in IPDL, i.e., Propositional Dynamic Logic (PDL) extended with an intersection operator on programs, is decidable in deterministic double exponential time. Since then, the exact complexity of IPDL has remained an open problem: the best known lower bound was the EXPTIME one stemming from plain PDL until, in 2004, the first author established EXPSPACE-hardness. In this paper, we finally close the gap and prove that IPDL is hard for 2-EXPTIME, thus 2-EXPTIME-complete. We then sharpen our lower bound, showing that it even applies to IPDL without the test operator interpreted on tree structures.

§1. Introduction. Building on a proposal by Pratt [25], Fischer and Ladner introduced propositional dynamic logic (PDL) as a logical system for reasoning about programs [9]. Since its invention in 1979, PDL has undergone countless modifications and extensions, mainly for two purposes: first, PDL was enriched with additional expressive means to capture properties of programs that cannot be expressed in basic PDL [28, 15, 29]; and second, several variants of PDL have been proposed with the goal of adapting the original formalism to completely new applications such as knowledge representation [10, 27, 21] and querying of semi-structured data [1, 2]. In the last years, these new applications have been the main driving force behind the persisting interest in the PDL family of modal logics. Many of PDL's variants are discussed in the surveys and monograph by Harel, Kozen, and Tiuryn [12, 18, 13].

In this paper, we study IPDL, the extension of PDL with an intersection operator on programs. Apart from being a natural extension from a theoretical viewpoint, there are two application-driven motivations for considering IPDL: first, IPDL is capable of capturing certain aspects of concurrency, thus belonging to a group of several PDL variants whose purpose is to allow reasoning about concurrent programs [17, 24, 14, 11, 23]. Second, IPDL may be viewed as a natural and powerful description logic (DL) [4], and thus has interesting applications as a knowledge representation tool in artificial intelligence. In particular, IPDL is closely related to the description logic $\mathcal{ALC}^{\cap}_{reg}$, i.e., the extension of the well-known DL $\mathcal{ALC}_{reg}$ [26, 3, 10] with an intersection operator on roles as considered e.g. in [8, 5, 20, 22]. The main difference between IPDL and $\mathcal{ALC}^{\cap}_{reg}$ is that the

0022-4812/00/0000-0000/$00.00

latter is usually defined without the test operator, as this operator is not very natural from a knowledge representation perspective.

The most important reasoning problems in description logic are satisfiability and subsumption, where the latter corresponds to the validity of implications $\varphi \rightarrow \psi$ and can easily be reduced to (the complement of) the former. Still, the computational complexity of deciding satisfiability in IPDL has never been exactly determined: a 2-ExpTime upper bound was established by Danecki in 1984 [7], but is not matched by the ExpTime lower bound inherited from PDL [9]. In a recent attempt at improving the lower bound, the first author showed that IPDL is at least ExpSpace-hard [19]. The purpose of the current paper is to close this gap, i.e., *to determine the exact computational complexity of deciding satisfiability in IPDL.*

We start with the proof of a 2-ExpTime lower bound for satisfiability in IPDL by reducing the word problem of exponentially space bounded alternating Turing machines (ATMs). Thus, the complexity of IPDL is determined as 2-ExpTime-complete. Interestingly, this is also the complexity of satisfiability in several other variants of PDL for reasoning about concurrent programs. Examples include PDL with programs specified by concurrent automata [14] and PDL with interleaving regular expressions [23].

Our first proof of 2-ExpTime hardness has two notable properties: first, it even applies if we admit only a single program and restrict models to tree structures. This is surprising since the main difficulty in obtaining *upper* bounds for IPDL is its lack of the tree model property [7]. Second, our reduction relies on the presence of the test operator which is, as noted above, usually omitted in description logics. Therefore, as a next step we reduce satisfiability in IPDL to satisfiability in test-free IPDL. Thus, the latter is also 2-ExpTime complete. This additional result, however, does not anymore capture the case of tree models as it relies on the presence of models in which states have reflexive loops. For this reason, we finally exhibit another, more intricate reduction of the word problem for exponentially space bounded ATMs showing that even test-free IPDL on tree-structures is 2-ExpTime hard.

§2. Preliminaries. We first give the syntax and semantics of propositional dynamic logic with intersection, and then discuss some basics about alternating Turing machines that are needed for the subsequent reductions.

2.1. Propositional Dynamic Logic with intersection. Let $\mathcal{P} = \{p, q, \dots\}$ be a countably infinite set of *atomic propositions* which includes $\mathtt{tt}$ and $\mathtt{ff}$. Let $\mathcal{A} = \{a, b, \dots\}$ be a countably infinite set of *atomic program names. Formulas* φ and *programs* α of IPDL are defined by the following syntax rules:

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

where q ranges over $\mathcal{P}$, and a ranges over $\mathcal{A}$. We will use the standard abbreviations $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$, $[\alpha]\varphi := \neg\langle\alpha\rangle\neg\varphi$, and $\alpha^+ := \alpha; \alpha^*$. For $n \in \mathbb{N}$, we write α^n to denote the n-fold composition of the program α.

Since we will sometimes have to write quite complex formulas and programs, it is convenient to explicate operator precedence. Concerning formula operators, we use the usual conventions that $\neg$, $\langle\alpha\rangle$, and $[\alpha]$ have higher precedence than $\wedge$ and $\vee$, which in turn have higher precendence than $\to$ and $\leftrightarrow$. For programs, $\cdot^*$ has higher precedence than ;.

IPDL formulas are interpreted in Kripke structures. A *Kripke structure* is a triple $(\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ with $\mathcal{S}$ a set of *states*, $\xrightarrow{a}$ a binary relation on states for every $a \in \mathcal{A}$, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ a labelling of states with sets of atomic propositions such that $\mathtt{tt} \in L(s)$ and $\mathtt{ff} \notin L(s)$ for all $s \in \mathcal{S}$. Let $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ be such a structure. The extension of the accessibility relations "$\xrightarrow{\alpha}$" to non-atomic programs and the consequence relation "$\models$" of IPDL are defined by simultaneous induction:

$$
\begin{array}{lll}
s \xrightarrow{\alpha;\beta} t & \text{iff} & \exists u \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} u \text{ and } u \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cup\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ or } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cap\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ and } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha^*} t & \text{iff} & \exists n \in \mathbb{N}, s \xrightarrow{\alpha^n} t \text{ where} \\
 & & \forall s,t \in \mathcal{S} : s \xrightarrow{\alpha^0} s, \text{ and } s \xrightarrow{\alpha^{n+1}} t \text{ iff } s \xrightarrow{\alpha;\alpha^n} t \\
s \xrightarrow{\varphi?} t & \text{iff} & s = t \text{ and } \mathcal{K}, s \models \varphi \\
 & & \\
\mathcal{K}, s \models q & \text{iff} & q \in L(s) \\
\mathcal{K}, s \models \varphi \vee \psi & \text{iff} & \mathcal{K}, s \models \varphi \text{ or } \mathcal{K}, s \models \psi \\
\mathcal{K}, s \models \neg\varphi & \text{iff} & \mathcal{K}, s \not\models \varphi \\
\mathcal{K}, s \models \langle\alpha\rangle\varphi & \text{iff} & \exists t \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} t \text{ and } \mathcal{K}, t \models \varphi
\end{array}
$$

A formula φ is *satisfiable* if there exists a Kripke structure $\mathcal{K}$ and a state s of $\mathcal{K}$ such that $\mathcal{K}, s \models \varphi$. Such a structure is called a *model* of φ.

As examples, consider the following two IPDL formulas:

$$\langle x \cap y^* \rangle \mathtt{tt} \wedge [y]\mathtt{ff} \qquad [x^*]\big(\langle x\rangle\mathtt{tt} \wedge [(x;x^*) \cap \mathtt{tt}?]\mathtt{ff}\big)$$

It is not hard to verify that the left-hand formula enforces a reflexive x-loop and that the right-hand formula enforces an infinite x-path that does not loop back to some state.

2.2. Alternating Turing machines. An *Alternating Turing Machine (ATM)* is of the form $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \Delta)$. The set of *states* $Q = Q_\exists \uplus Q_\forall \uplus \{q_a\} \uplus \{q_r\}$ consists of *existential states* from $Q_\exists$, *universal states* from $Q_\forall$, an *accepting state* q_a, and a *rejecting state* q_r; Σ is the *input alphabet* and Γ the *work alphabet* containing a *blank symbol* $\Box$ and satisfying $\Sigma \subseteq \Gamma$; $q_0 \in Q_\exists \cup Q_\forall$ is the *starting* state; and the *transition relation* δ is of the form

$$\delta \;\subseteq\; Q \times \Gamma \times Q \times \Gamma \times \{L, R, N\}.$$

We will write $\delta(q,a)$ for $\{(q', b, M) \mid (q, a, q', b, M) \in \delta\}$. As usual, we assume that $q \in Q_\exists \cup Q_\forall$ implies $\delta(q,b) \neq \emptyset$ for all $b \in \Gamma$ and $q \in \{q_a, q_r\}$ implies $\delta(q,b) = \emptyset$ for all $b \in \Gamma$.

A *configuration* of an ATM is a word wqw' with $w, w' \in \Gamma^*$ and $q \in Q$. The intended meaning is that the tape contains the word ww' (with only blanks before and behind it), the machine is in state q, and the head is on the leftmost symbol of w'. The *successor configurations* of a configuration wqw' are defined in the usual way in terms of the transition relation δ. A *halting configuration* is of the form wqw' with $q \in \{q_a, q_r\}$.

A *computation path* of an ATM $\mathcal{M}$ on a word w is a (finite or infinite) sequence of configurations $c_1, c_2, \ldots$ such that $c_1 = q_0 w$ and c_{i+1} is a successor configuration of c_i for $i \geq 0$. All ATMs considered in this paper have only *finite* computation paths on any input. Since this case is simpler than the general one, we define acceptance for ATMs with finite computation paths, only, and refer to [6] for the full definition. Let $\mathcal{M}$ be such an ATM. A halting configuration is *accepting* iff it is of the form $wq_a w'$. For other configurations $c = wqw'$, the acceptance behaviour depends on q: if $q \in Q_\exists$, then c is accepting iff at least one successor configuration is accepting; if $q \in Q_\forall$, then c is accepting iff all successor configurations are accepting. Finally, the ATM $\mathcal{M}$ with starting state q_0 *accepts* the input w iff the *initial configuration* $q_0 w$ is accepting. We use $L(\mathcal{M})$ to denote the language accepted by $\mathcal{M}$, i.e., $L(\mathcal{M}) = \{w \in \Sigma^* \mid \mathcal{M} \text{ accepts } w\}$.

To obtain a witness for the acceptance of an input by an ATM, it is common to arrange configurations in a tree. Such an *acceptance tree of an ATM $\mathcal{M}$* with starting state q_0 *on a word w* is a finite tree whose nodes are labelled with configurations such that

- the root node is labelled with the initial configuration $q_0 w$;
- if a node s in the tree is labelled with wqw', $q \in Q_\exists$, then s has exactly one successor, and this successor is labelled with a successor configuration of wqw';
- if a node s in the tree is labelled with wqw', $q \in Q_\forall$, then there is exactly one successor of s for each successor configuration of wqw';
- leaves are labelled with accepting halting configurations.

The following is immediate.

LEMMA 1. *Let $\mathcal{M}$ be an ATM with only finite computation paths. Then there exists an acceptance tree of $\mathcal{M}$ on w iff $\mathcal{M}$ accepts w.*

Note that, if computations of alternating Turing machines are regarded as a game, then an acceptance tree corresponds to a winning strategy for the existential player.

§3. The basic result. The aim of this section is to prove our basic result: 2-EXPTIME-hardness of satisfiability in IPDL. The proof is by reduction of the word problem of exponentially-space bounded ATMs, and resembles the techniques used in [30, 23, 16]. Together with the 2-EXPTIME upper bound proved by Danecki [7], we thus obtain the following:

THEOREM 1. *Satisfiability in IPDL is 2-*EXPTIME*-complete.*

According to Theorem 3.4 of [6], there is an exponentially space bounded ATM $\mathcal{M}$ whose word problem is 2-EXPTIME-hard. According to Theorem 2.6 of the same paper, we may w.l.o.g. assume that there exists a polynomial p such that the length of every computation path of $\mathcal{M}$ on $w \in \Sigma^n$ is bounded by $2^{2^{p(n)}}$, and all the configurations wqw' in such computation paths satisfy $|ww'| \leq 2^{p(n)}$.

Let $w = a_0 \cdots a_{n-1} \in \Sigma^*$ be an input to $\mathcal{M}$. In the following, we construct an IPDL formula $\varphi_{\mathcal{M},w}$ over a singleton set $\mathcal{A} = \{x\}$ of atomic programs such that $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable. Intuitively, $\varphi_{\mathcal{M},w}$ will be constructed such that its models correspond to an acceptance tree of $\mathcal{M}$ on w. In such models, each state represents a tape cell of a configuration of $\mathcal{M}$, and the program x indicates both "moving to the next tape cell in the same configuration" and "moving to the first tape cell of a successor configuration". We use the set of atomic propositions $Q \cup \Gamma \cup \{c_0, \dots, c_{p(n)-1}\}$, whose intuitive meaning is as follows:

- $q \in Q$ is true in a state of the model if the head of $\mathcal{M}$ is on the corresponding tape cell in the corresponding configuration while the machine is in state q;
- $a \in \Gamma$ is true if a is the symbol on the corresponding tape cell;
- $c_{p(n)-1}, \dots, c_0$ represent a counter C in binary coding for counting the $2^{p(n)}$ tape cells of configurations with the leftmost cell having counter value 0.

Take for example a configuration such that the tape head is on the third cell from the left, the machine is in state q, and the tape is labelled with $a_1 a_2 \cdots a_{2^{p(n)}-1}$. This configuration is modelled by a sequence of states of the form

$$
\begin{array}{ccccccccccccc}
 & & & & & & & & & & & & c_{p(n)-1}\\
 & & & & q & & c_0 & & & & & & \vdots\\
 & & c_0 & & c_1 & & c_1 & & c_2 a & & & & c_0\\
a_1 & & a_2 & & a_3 & & a_4 & & a_5 & & & & a_{2^{p(n)}-1}\\
\bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \bullet & \xrightarrow{x} & \cdots & \xrightarrow{x} & \bullet
\end{array}
$$

Successive configurations are modelled by connecting such sequences with the program x.

We now start to assemble the reduction formula $\varphi_{\mathcal{M},w}$. Let N abbreviate $2^{p(n)}$ in what follows. We will often need the following auxiliary formulas stating that the value of the counter C is zero and $N-1$, respectively:

$$\chi_{C=0} := \bigwedge_{i=0}^{p(n)-1} \neg c_i \quad \text{and} \quad \chi_{C=N-1} := \bigwedge_{i=0}^{p(n)-1} c_i$$

Next, we need an auxiliary formula φ_{inc} which ensures that the value of C is incremented modulo N when moving from a state to one of its x-successors. A regular incrementation corresponds to moving to the successor cell within the same configuration, while the "modulo step" corresponds to moving to a successor configuration.

$$
\begin{aligned}
\varphi_{inc} \quad := [x^*]\Big(& \bigwedge_{k=0}^{p(n)-1} \Big(\bigwedge_{j=0}^{k-1} c_j \to (c_k \to [x]\neg c_k) \wedge (\neg c_k \to [x]c_k)\Big) \wedge \\
& \bigwedge_{k=0}^{p(n)-1} \Big(\bigvee_{j=0}^{k-1} \neg c_j \to (c_k \to [x]c_k) \wedge (\neg c_k \to [x]\neg c_k)\Big) \Big)
\end{aligned}
$$

This is essentially just the standard propositional formula for incrementing a binary counter modulo N: a bit is toggled if all bits strictly lower have value one, and kept otherwise.

We introduce two auxiliary programs: a program α_{last} that relates any state to the last cell of the same configuration, and a program α_{2h} that does an arbitrary amount of x-steps whilst seeing two tape heads. For simplicity, we use the formula $h := \bigvee_{q \in Q} q$ saying that the tape head is on the current cell.

$$\begin{array}{rcl} \alpha_{last} & := & (\neg\chi_{C=N-1}?;x)^*;\chi_{C=N-1}? \\ \alpha_{2h} & := & x^*;h?;x^+;h?;x^* \end{array}$$

Now we can formalise the general requirements on an ATM: every tape cell is marked with exactly one symbol from Γ and never with two different states, and no configuration has more than one cell marked with the tape head. At this point, it is not necessary to explicitly state that each configuration has *at least* one cell marked with the tape head: this will be a consequence of other formulas added below.

$$\begin{array}{rcl} \varphi_{gen} & := & [x^*]\Big(\big(\bigvee\limits_{a \in \Gamma} a \big) \ \wedge \ \bigwedge\limits_{a,b \in \Gamma, b \neq a} \neg(a \wedge b) \ \wedge \ \bigwedge\limits_{q,q' \in Q, q \neq q'} \neg(q \wedge q') \ \wedge \\ & & \qquad (\ \chi_{C=0} \ \rightarrow \ [\alpha_{last} \cap \alpha_{2h}]\mathtt{ff} \ ) \Big) \end{array}$$

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

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

We now encode $\mathcal{M}$'s transition function δ. To this end, we need some more auxiliary programs. First, we devise a program $\alpha_=$ that relates a tape cell to the corresponding cell in successor configurations. This corresponding cell is identified by having the same C-value. To ensure that we reach the direct successor configuration, we additionally require the counter C to become 0 only once on the way.

$$\begin{array}{rcl} \alpha_{0!} & := & (x;\neg\chi_{C=0}?)^*;x;\chi_{C=0}?;(x;\neg\chi_{C=0}?)^* \\ \alpha_= & := & \alpha_{0!} \ \cap \ \bigcap\limits_{i=0}^{p(n)-1} \ (\ (c_i?;x^+;c_i?) \ \cup \ (\neg c_i?;x^+;\neg c_i?) \ ) \end{array}$$

Second, we define a program $\alpha_{q,a,M}$ for each $(q,a,M) \in Q \times \Gamma \times \{L,R,N\}$. Intuitively, the purpose of these programs is as follows: if a state s represents the head position of a configuration c, then enforcing the existence of an $\alpha_{q,a,M}$-successor of s ensures that c has a successor configuration that is produced by writing a, moving according to M, and switching to state q.

$$\begin{array}{rcl} \alpha_{q,a,N} & := & \alpha_=;(q \wedge a)? \\ \alpha_{q,a,R} & := & \neg\chi_{C=N-1}?;\alpha_=;a?;x;q? \\ \alpha_{q,a,L} & := & \neg\chi_{C=0}?;(\alpha_= \cap (x^*;(q \wedge [x]a)?);x) \end{array}$$

The left-most components of $\alpha_{q,a,R}$ and $\alpha_{q,a,L}$ ensure that we never make a right move on the right end of the tape, and never a left move on the left end of the tape.

Now we are able to formalise δ. In an existential state, we find one successor configuration. In a universal state, all successor configurations are present in the model. Finally, the label of any tape cell which is not under the tape head remains the same in the following configuration.

$$\begin{aligned}\varphi_\delta \ := \ [x^*]\Big(& \bigwedge_{q\in Q_\exists, a\in\Gamma} \Big(q \wedge a \ \rightarrow \bigvee_{(p,b,M)\in\delta(q,a)} \langle\alpha_{p,b,M}\rangle\texttt{tt} \Big) \ \wedge \\ & \bigwedge_{q\in Q_\forall, a\in\Gamma} \Big(q \wedge a \ \rightarrow \bigwedge_{(p,b,M)\in\delta(q,a)} \langle\alpha_{p,b,M}\rangle\texttt{tt} \Big) \ \wedge \\ & \bigwedge_{a\in\Gamma} (\neg h \wedge a \ \rightarrow \ [\alpha_=]a \ ) \Big)\end{aligned}$$

It remains to describe acceptance of the machine. Since all computation paths of $\mathcal{M}$ are finite and configurations wqw' with $q \in Q_\forall \cup Q_\exists$ have at least one successor configuration, it suffices to require that the state q_r never appears:

$$\varphi_{acc} := [x^*]\neg q_r$$

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

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

Since $|\varphi_{\mathcal{M},w}|$ is polynomial in $|\mathcal{M}|$ and $|w|$, the following lemma together with Danecki's 2-ExpTime upper bound yields Theorem 1.

LEMMA 2. *$w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable.*

PROOF. "if". Let $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ be a model of $\varphi_{\mathcal{M},w}$, and let $s_{ini} \in \mathcal{S}$ such that $\mathcal{K}, s_{ini} \models \varphi_{\mathcal{M},w}$. We inductively define an acceptance tree T of $\mathcal{M}$ on w. By Lemma 1, the existence of such a tree implies that $w \in L(\mathcal{M})$ as required.

For a node τ of T, we use $w_\tau q_\tau w'_\tau$ to denote the (components of) the configuration that labels τ. The inductive definition of T proceeds in steps $1, 2, 3, \ldots$ as follows: in the i-th step, we define the components w_τ and q_τ for all nodes τ on level $i-1$ and the component w'_τ for all nodes τ on level $i-2$ (where the root is on level 0). Finally, in a "finishing off" step we define the components w'_τ for all nodes τ that have no successors (i.e., where $q_\tau \in \{q_a, q_r\}$). Along with T, we define a function f assigning each node τ of T to a corresponding state in $\mathcal{K}$ such that the following properties hold:

1. $\mathcal{K}, f(\tau) \models q_\tau$ for all nodes τ of T;
2. if π is a successor of τ in T, then there are states $s_0, \ldots, s_k$, with $k = |w'_\tau| + |w_\pi|$, such that
   (a) $f(\tau) = s_0$, $f(\pi) = s_k$;
   (b) $s_i \xrightarrow{x} s_{i+1}$, and for $i < k$;
   (c) $\mathcal{K}, s_i \models a$ if a is the $i+1$-st symbol of w'_τ for $i < |w'_\tau|$;
   (d) $\mathcal{K}, s_{|w'_\tau|+i} \models a$ if a is the $i+1$-st symbol of w_π for $i < |w_\pi|$.

For the induction start, take a tree T constisting of a single node r and set $w_r := \varepsilon$, $q_r := q_0$, and $f(r) := s_{ini}$. Then Property 1 of f is satisfied by φ_{start} and Property 2 is trivially satisfied.

For the induction step, take a node τ such that w_τ and q_τ have already been defined. Assume that q_τ is an existential state. By φ_{gen} (and by Property 2 of f), there is a unique $a \in \Sigma$ such that $\mathcal{K}, f(\tau) \models a$. By Property 1 of f and φ_δ, there is a $(p, b, M) \in \delta(q_\tau, a)$ such that $\mathcal{K}, f(\tau) \models \langle \alpha_{p,b,M} \rangle \mathtt{tt}$. Assume that $M = N$, i.e., $\mathcal{M}$ does not move the head. By definition of $\alpha_{p,b,N}$ and due to φ_{inc}, there thus exists a sequence of states $s_0, \ldots, s_N$ such that $s_0 = f(\tau)$, $s_i \xrightarrow{x} s_{i+1}$ for $i < N$, and $\mathcal{K}, s_N \models p \wedge b$. For $i < N$, let $b_i \in \Sigma$ denote the unique (due to φ_{gen}) symbol such that $\mathcal{K}, s_i \models b_i$. Modify the tree as follows:

- Set $w'_\tau := b_0 \cdots b_{N-(|w_\tau|+1)}$.
- Introduce a successor π of τ in T. Set $w_\pi := b_{N-|w_\tau|} \cdots b_{N-1}$, $q_\pi := p$, and $f(\pi) := s_N$.

The cases of left and right moves are handled analogously. The case of universal states is also handled analogously with the difference that we introduce a successor node in T for *every* $(p, b, M) \in \delta(q_\tau, a)$.

Finally, the "finishing off" phase is done as follows. Let π be a node in T such that $q_\pi \in \{q_a, q_r\}$. Since, by definition of ATMs, q_0 is either existential or universal, there exists a predecessor τ of π in T. Assume that π was introduced for some $(p, b, M) \in \delta(q_\tau, a)$ with $M = L$. Then set $w'_\pi := c \cdot b \cdot {w'}_\tau^-$, where c is the right-most symbol of w_τ and ${w'}_\tau^-$ is obtained from w'_τ by deleting the left-most symbol. The cases of right and no moves are handled analogously.

We leave it as an exercise to show that the root of T is labelled with the initial configuration, and that if π is a successor of τ in T, then $w_\pi q_\pi w'_\pi$ is a successor configuration of $w_\tau q_\tau w'_\tau$.

"only if". Assume that T is an acceptance tree for $\mathcal{M}$ on w with set of nodes $\mathcal{N}$. If $\tau \in \mathcal{N}$ is a node and τ is labelled with wqw', we use q_τ to denote q, w^i_τ to denote the i-th symbol of ww', for $i < N$, and h_τ to denote the length of w (i.e., the head position). Construct a Kripke structure $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ as follows:

- $\mathcal{S} = \mathcal{N} \times \{0, \ldots, N-1\}$.
- $(\tau, i) \xrightarrow{x} (\pi, j)$ if $\tau = \pi$ and $j = i + 1$
  or if $i = N - 1$, $j = 0$, and π is a successor of τ in T;
- $q \in L((\tau, i))$ if $q = q_\tau$ and $i = h_\tau$, for all $q \in Q$;
- $a \in L((\tau, i))$ if $w^i_\tau = a$, for $a \in \Sigma$;
- $c_j \in L((\tau, i))$ if the j-th bit of i is one, for $j < N$.

It is tedious but straightforward to verify that $\mathcal{K}, (r, 0) \models \varphi_{\mathcal{M},w}$, where $r \in \mathcal{N}$ denotes the root of T. ⊣

There are two interesting things to be noted about the reduction: first, it only uses a single atomic program x. And second, the proof of Lemma 2 shows that, if the formula $\varphi_{\mathcal{M},w}$ is satisfiable, then it is satisfiable in a tree-shaped model (as constructed in the "only if" direction). Thus, even satisfiability of IPDL formulas in tree models is 2-ExpTime-hard. This is surprising since the main problem in proving *upper* bounds for IPDL is its lack of the tree model propery.

Note that IPDL models need not even be directed acyclic graphs: the formula $\langle x \cap y^* \rangle \mathtt{tt} \wedge [y]\mathtt{ff}$ introduced in Section 2.1 enforces a reflexive x-loop. If we modify this formula to $\langle x^+ \cap y^* \rangle \mathtt{tt} \wedge [y]\mathtt{ff} \wedge \varphi_{inc}$, with φ_{inc} as defined in the reduction, we even obtain a cycle of at least exponential length.

To define tree-shaped models in a precise way, associate with each Kripke structure $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{A}\}, L)$ an edge-labelled directed graph $G_\mathcal{K} := (V_\mathcal{K}, E_\mathcal{K})$ with

$$\begin{array}{rcl} V_\mathcal{K} & := & \mathcal{S} \\ E_\mathcal{K} & := & \{(s,a,t) \mid s \xrightarrow{a} t\}. \end{array}$$

Now, $\mathcal{K}$ is called a *tree structure* if $G_\mathcal{K}$ is connected and acyclic and each state $s \in \mathcal{S}$ has at most a single incoming edge. An IPDL formula is called *tree satisfiable* if it is satisfiable in a tree structure. By the above observations, we obtain the following corollary.

COROLLARY 1. *Tree satisfiability in IPDL over a singleton set of atomic programs is 2-*EXPTIME*-complete.*

§4. Strengthening the result. As noted in the introduction, the test operator of IPDL is not very natural when IPDL is viewed as a description logic. Therefore, our next aim is to eliminate tests from the above proof. To start, note that we can easily modify the reduction to contain tests of atomic propositions, only: each complex test $\varphi?$ can be replaced with the atomic test $p_\varphi?$, p_φ a fresh atomic proposition, if we add the conjunct $[x^*](\varphi \leftrightarrow p_\varphi)$ to $\varphi_{\mathcal{M},w}$. However, the next theorem shows that tests can even be completely omitted. We call the variant of IPDL that is obtained by disallowing the test operator *test-free IPDL*.

THEOREM 2. *Satisfiability of test-free IPDL is 2-*EXPTIME*-complete.*

PROOF. As the upper bound follows from Danecki's results, we concentrate on the lower bound and present a satisfiability preserving and polynomial reduction from IPDL to test-free IPDL. Together with Theorem 1, we obtain the desired result.

Let φ be an IPDL formula with $\psi_1?, \ldots, \psi_n?$ all test programs occurring in φ. Assume that $a_1, \ldots, a_m$ are all atomic programs occurring in φ. Take new atomic programs $b_1, \ldots, b_n$ and d, and let

$$\varphi' \;:=\; \widehat{\varphi} \wedge [(a_1 \cup \ldots \cup a_m)^*]\Big([d]\mathtt{ff} \wedge \bigwedge_{i=1}^{n} (\psi_i \leftrightarrow \langle b_i \cap d^* \rangle \mathtt{tt})\Big)$$

where $\widehat{\varphi}$ denotes the result of replacing each program $\psi_i?$ in φ with $(b_i \cap d^*)$, for $i = 1, \ldots, n$. Clearly, φ' is test-free.

Now suppose that φ' is satisfiable, i.e., it has a model $\mathcal{K}$. Let s_0 be a state with $\mathcal{K}, s_0 \models \varphi'$. Take the restriction $\mathcal{K}'$ of $\mathcal{K}$ to the set of states $\{s \mid s_0 \xrightarrow{(a_1 \cup \ldots \cup a_m)^*} s\}$. By the second conjunct of φ', for all states s of $\mathcal{K}'$ we have $s \xrightarrow{\psi_i?} s$ iff $s \xrightarrow{b_i} s$. Thus, $\mathcal{K}$ being a model of the first line of φ' implies that $\mathcal{K}$ is also a model of φ.

Conversely, let $\mathcal{K}$ be a model of φ. Construct a Kripke structure $\mathcal{K}'$ from $\mathcal{K}$ by interpreting d as the empty program and the programs $b_1, \ldots, b_n$ by setting

$s \xrightarrow{b_i} s'$ iff $s \xrightarrow{\psi_i?} s'$. Thus, b_i consists of reflexive loops at precisely those points where ψ_i is true. Clearly, $\mathcal{K}'$ is a model of φ'. ⊣

The proof of Theorem 2 relies on the ability to add reflexive loops to points in a Kripke structure. Since, in contrast, Theorem 1 captures the case of tree structures, it is a natural question whether tree satisfiability of test-free IPDL is still 2-ExpTime-hard. By refining the proof of Theorem 1, we answer this question to the affirmative. A corresponding upper bound is again obtained from Danecki's result.

THEOREM 3. *Tree satisfiability of test-free IPDL is 2-*ExpTime*-complete.*

Again, we reduce the word problem for exponentially space bounded ATMs. Let $\mathcal{M}$ be such an ATM and let p be a polynomial such that the length of every computation path of $\mathcal{M}$ on $w \in \Sigma^n$ is bounded by $2^{2^{p(n)}}$, and all the configurations wqw' in such computation paths satisfy $|ww'| \leq 2^{p(n)}$. Let $w = a_0, \ldots, a_{n-1}$ be an input to $\mathcal{M}$.

The essential idea to compensate for the lack of test operators is to encode the counter C using atomic *programs* instead of atomic propositions. To do this, we have to give up the idea that a single tape cell is represented by a single state. Instead, a tape cell is represented by a sequence $s_0, \ldots, s_{p(n)}$ of states. In this sequence, each state s_i is connected to s_{i+1} by one of the atomic programs $c_{i/0}$ and $c_{i/1}$. Thus, we obtain a sequence $c_{0/t_0}, \ldots, c_{p(n)-1/t_{p(n)-1}}$ of atomic programs that encodes the counter C via the bit sequence $t_0, \ldots, t_{p(n)-1}$. As in the original reduction, the alphabet symbol found in a tape cell and the ATM state are encoded using atomic propositions. We use the following signature:

- atomic propositions $q \in Q$ and $a \in \Gamma$ as in the original reduction;
- programs $c_{i/t}$ with $i < p(n)$ and $t \in \{0, 1\}$ for representing the counter C as described above;
- a program x for going to the next tape cell of the same configuration;
- a program s for going to the first tape cell of successor configurations;
- additional atomic propositions $m_{q,a}$, $q \in Q$ and $a \in \Sigma$, that will be used as "markers" for dealing with left moves.

To illustrate the representation of tape cells via multiple states, assume that $p(n)$ is 3 and consider the tape cell number four (i.e., 100 in binary). Assume that the symbol in this cell is a, that the head is on the cell, and that the current state is q. This tape cell is modelled by a sequence of states of the form

$$\begin{array}{ccccccc} a & & a & & a & & a \\ q & & q & & q & & q \\ \bullet & \xrightarrow{c_{0/1}} & \bullet & \xrightarrow{c_{1/0}} & \bullet & \xrightarrow{c_{2/0}} & \bullet \end{array}$$

Configurations are then modelled by concatenating such sequences using the atomic program x.

Let $\mathcal{P}$ be the set of all programs listed above. We again use N to abbreviate $2^{p(n)}$ and define further abbreviations as follows:

- $c_i := c_{i/0} \cup c_{i/1}$ for $i < p(n)$, i.e., counter programs encoding bit number i;
- $c := c_0 \cup \cdots \cup c_{p(n)-1}$, i.e., the union of all counter programs;

- $c_{true} := c_{0/1} \cup \cdots \cup c_{p(n)-1/1}$, i.e., counter programs encoding a bit that is set;
- $c_{false} := c_{0/0} \cup \cdots \cup c_{p(n)-1/0}$, i.e., counter programs encoding a bit that is not set;
- $-(p_1 \cup \cdots \cup p_k) := \bigcup_{p \in \mathcal{P} \setminus \{p_1,\ldots,p_k\}} p$ for $p_1, \ldots, p_k \in \mathcal{P}$, i.e., the union of all relevant programs except $p_1, \ldots, p_k$;
- $u := \bigcup_{p \in \mathcal{P}} p$, i.e., the union of all relevant programs.

The following formula φ_{sane} ensures that all paths in models of the reduction formula exhibit the expected pattern of programs, which can be described by the regular expression $(c_0; \cdots; c_{p(n)}; (x \cup s))^*$ (Lines 1 to 4). It also enforces that the program s appears exactly after those tape cells with number $N-1$ (Lines 5 to 7).

$$\begin{aligned} \varphi_{sane} \;:=\; & [-c_0]\mathtt{ff} \;\wedge \\ & [u^*]\Big(\bigwedge_{i=0}^{p(n)-2} [c_i; -c_{i+1}]\mathtt{ff} \;\wedge \\ & \quad [c_{p(n)}; -(x \cup s)]\mathtt{ff} \;\wedge \\ & \quad [(x \cup s); -c_0]\mathtt{ff} \;\wedge \\ & \quad [c_{0/1}; \cdots; c_{p(n)-1/1}; x]\mathtt{ff} \;\wedge \\ & \quad [c_0; c^*; c_{false}; c^*; s]\mathtt{ff} \;\wedge \\ & \quad [c_{0/0}; c^*; s]\mathtt{ff} \Big) \end{aligned}$$

We also want that any two states are connected by at most a single atomic program. Since this already follows from the definition of tree structures, there is no need to explicitly enforce it (this would be simple).

Apart from the additional φ_{sane}, the reduction formula $\varphi_{\mathcal{M},w}$ will consist of the same conjuncts as in the proof of Theorem 1, only in a different formulation. We start with encoding incrementation of the counter C modulo N:

$$\begin{aligned} \alpha_{k,t} \;&:=\; c_{0/1}; \cdots; c_{k-1/1}; c_{k/t}; c^*; (x \cup s); c_0; \cdots; c_{k-1}; c_{k/t} \\ \alpha'_{k,t} \;&:=\; ((c_0; \cdots; c_{k-1}) \cap (c^*; c_{false}; c^*)); c_{k/t}; c^*; (x \cup s); c_0; \cdots; c_{k-1}; c_{k/1-t} \\ \varphi_{inc} \;&:=\; [u^*]\Big(\bigwedge_{k=0}^{p(n)-1} \bigwedge_{t \in \{0,1\}} [\alpha_{k,t}]\mathtt{ff} \;\wedge\; \bigwedge_{k=0}^{p(n)-1} \bigwedge_{t \in \{0,1\}} [\alpha'_{k,t}]\mathtt{ff} \Big) \end{aligned}$$

The idea is the same as in the corresponding formula in the proof of the original reduction: a bit is toggled if all bits strictly lower have value one, and kept otherwise. Note that $\alpha_{k,t}$ relates the first element of a sequence encoding a tape cell whose bits 0 to $k-1$ have value one and whose k-th bit has value t to the $k+1$-st state of a successor tape cell whose k-th bit has also value t. By using $[\alpha_{k,t}]\mathtt{ff}$, such unwanted situations are forbidden implying that successor cells have the correct value of the k-th bit. The program $\alpha'_{k,t}$ can be understood similarly.

We now encode some basic facts about ATMs: that all the states in a sequence representing a single tape cell are associated with a unique alphabet symbol and with a unique state (or no state at all). We also ensure that there is not more than one head per configuration. As in the original reduction, it is not necessary to state that there is at least one head. We again use the formula $h := \bigvee_{q\in Q} q$.

$$\begin{aligned}\varphi_{gen} \;:=\; [u^*]\Big(& \bigvee_{a\in\Gamma} a \;\wedge \bigwedge_{a,b\in\Gamma,a\neq b} \neg(a\wedge b) \;\wedge \bigwedge_{q,p\in Q,q\neq p} \neg(q\wedge p) \;\wedge \\ & \bigwedge_{q\in Q} (q\to[c]q)\wedge(\langle c\rangle q\to q) \;\wedge \\ & \bigwedge_{a\in\Gamma} (a\to[c]a)\wedge(\langle c\rangle a\to a) \;\wedge \\ & h\to[c^*;x;(-s)^*]\neg h \;\Big)\end{aligned}$$

Next, we describe the initial configuration.

$$\begin{aligned}\varphi_{start} \;:=\; & q_0 \wedge a_0 \;\wedge\; \langle c_{0/0};\cdots;c_{p(n)/0}\rangle\mathtt{tt} \;\wedge \\ & [c^*;x](\; a_1 \;\wedge \\ & \quad [c^*;x](\; a_2 \;\wedge \\ & \quad\quad \cdots \\ & \quad\quad\quad [c^*;x](\; a_{n-1}\wedge[c^*;x;(-s)^*]\Box\;)\cdots))\end{aligned}$$

The next step is to encode $\mathcal{M}$'s transition function δ. We first construct a program that relates the first state representing a tape cell to the last state representing the same cell in successor configurations.

$$\begin{aligned}\alpha_{=} \;:=\; & ((-s)^*;s;(-s)^*) \;\cap \\ & \bigcap_{j=0}^{p(n)-1} (c^*;c_{j/0};u^*;c_{j/0};c^*)\cup(c^*;c_{j/1};u^*;c_{j/1};c^*)\end{aligned}$$

Two notes are in order. First, using $\alpha_=$ inside a diamond, as we shall do below, will produce a sequence of tape cells and configurations as expected due to the formulas φ_{sane} and φ_{inc}. Second, $\alpha_=$ works as expected only since we are considering tree structures. On arbitrary structures, $\alpha_=$ will relate the tape cells with the same number in successive configurations as desired, but possibly also additional ones: if there is more than a single path between two states, the program may relate tape cells that do *not* have the same number. Clearly, this case is irrelevant on tree structures.

Next, we define a formula $\varphi_{q,a,M}$ for each $(q,a,M)\in Q\times\Gamma\times\{L,R,N\}$. The purpose of these formulas is similar to that of the *programs* $\alpha_{q,a,M}$ in the original reduction: if the state s is the first state of a sequence representing a tape cell carrying the head position in the current configuration c, then enforcing $\varphi_{q,a,M}$ to be true at s ensures that c has a successor configuration that is produced by writing a, moving according to M, and switching to state q. For dealing with left moves, we use the marker propositions $m_{q,a}$: if $m_{q,a}$ is true in the last state of a tape cell, then the head has been on this cell in the previous configuration, and the machine moved left, wrote a, and switched to state q in its last transition.

$$\begin{array}{lcl} \varphi_{q,a,N} & := & \langle \alpha_= \rangle (a \wedge q) \\ \varphi_{q,a,R} & := & \langle \alpha_= \rangle (a \wedge \langle x ; c^{p(n)} \rangle q) \ \wedge [c_{0/1} ; \cdots ; c_{p(n)/1}]\mathtt{ff} \\ \varphi_{q,a,L} & := & \langle \alpha_= \rangle m_{q,a} \ \wedge [c_{0/0} ; \cdots ; c_{p(n)/0}]\mathtt{ff} \end{array}$$

The last conjuncts of $\varphi_{q,a,R}$ and $\varphi_{q,a,L}$ ensure that we never make a right move on the right end of the tape, and never a left move on the left end of the tape. There is no need to enforce that $m_{q,a}$ holds either in all or in no state of the state sequence representing a cell. Only occurrences of $m_{q,a}$ in the last state of such a sequence are relevant.

Now we are able to formalize δ. The idea is the same as in the original reduction with the difference that we use the marker $m_{q,a}$ for transitions moving to the left. Also, we have to be careful to use the program $\alpha_=$ only in states that are the first state in the sequence representing a tape cell. To this end, we employ the program $\varphi_{\mathit{first}} = \langle c^{p(n)} \rangle \mathtt{tt}$.

$$\begin{array}{lcl} \varphi_\delta & := & [u^*]\Big(\bigwedge_{q \in Q_\exists, a \in \Gamma} \Big((a \wedge q \wedge \varphi_{\mathit{first}}) \ \rightarrow \bigvee_{(p,b,M) \in \delta(q,a)} \varphi_{p,b,M} \Big) \ \wedge \\ & & \quad \bigwedge_{q \in Q_\forall, a \in \Gamma} \Big((a \wedge q \wedge \varphi_{\mathit{first}}) \ \rightarrow \bigwedge_{(p,b,M) \in \delta(q,a)} \varphi_{p,b,M} \Big) \ \wedge \\ & & \quad \bigwedge_{q \in Q, a \in \Gamma} (\ m_{q,a} \rightarrow q \wedge [x]a \ ) \ \wedge \\ & & \quad \bigwedge_{a \in \Gamma} (\ (a \wedge \neg h \wedge \varphi_{\mathit{first}}) \rightarrow [\alpha_=]a) \Big) \end{array}$$

As in the original reduction, to describe acceptance of the ATM it suffices to state that the rejecting state q_r never appears:

$$\varphi_{acc} := [u^*]\neg q_r .$$

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

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

The following lemma can be proved analogously to Lemma 2.

LEMMA 3. *$w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable in a tree model.*

Since $|\varphi_{\mathcal{M},w}|$ is polynomial in $|\mathcal{M}|$ and $|w|$, Lemma 2 yields Theorem 3.

Note that we do not obtain an analogue of Corollary 1 since it is essential for the modified reduction to use more than a single program. Indeed, we leave the complexity of tree satisfiability in test-free IPDL with a single atomic program as an open problem.

§5. Conclusion. We have proved that satisfiability in IPDL is 2-ExpTime-hard, and that this lower complexity bound applies even if we disallow the test operator and consider only tree structures. As further work, it would be interesting to look for fragments of IPDL that are still ExpTime-complete, and thus not harder than PDL. Indeed, it seems possible that the removal of *any* program operator from IPDL, except test, results in a logic that is not 2-ExpTime-hard. This is obviously true for the intersection operator, as its removal yields plain PDL [9] which is ExpTime-complete. It is also true for the Kleene star since test-free IPDL without Kleene-star is known to be PSpace-complete [22]. For IPDL without either composition or union, the complexity of satisfiability is open. We believe that IPDL without composition is ExpTime-complete.

REFERENCES

[1] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Maarx, and M. de Rijke, *PDL for ordered trees*, ***Journal of Applied Non-Classical Logic***, (2005), To appear.

[2] N. Alechina, S. Demri, and M. de Rijke, *A modal perspective on path constraints*, ***Journal of Logic and Computation***, vol. 13 (2003), no. 6, pp. 939–956.

[3] F. Baader, *Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles*, ***Proceedings of the twelfth international joint conference on artificial intelligence (IJCAI-91)*** (Sydney, Australia), 1991, pp. 446–451.

[4] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider, ***The description logic handbook: Theory, implementation and applications***, Cambridge University Press, 2003.

[5] D. Calvanese, G. De Giacomo, and M. Lenzerini, *On the decidability of query containment under constraints*, ***Proceedings of the 17th acm sigact sigmod sigart symposium on principles of database systems (pods'98)***, 1998, pp. 149–158.

[6] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, ***Journal of the ACM***, vol. 28 (1981), no. 1, pp. 114–133.

[7] R. Danecki, *Nondeterministic propositional dynamic logic with intersection is decidable*, ***Proceedings of the fifth symposium on computation theory*** (Zaborów, Poland) (A. Skowron, editor), LNCS, vol. 208, Springer, December 1984, pp. 34–53.

[8] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt, *The complexity of concept languages*, ***Information and Computation***, vol. 134 (1997), no. 1, pp. 1–58.

[9] M. J. Fischer and R. E. Ladner, *Propositional dynamic logic of regular programs*, ***Journal of Computer and System Sciences***, vol. 18 (1979), no. 2, pp. 194–211.

[10] G. De Giacomo and M. Lenzerini, *Boosting the correspondence between description logics and propositional dynamic logics*, ***Proceedings of the twelfth national conference on artificial intelligence (aaai'94). volume 1***, AAAI Press, 1994, pp. 205–212.

[11] R. Goldblatt, *Parallel action: Concurrent dynamic logic with independent modalities*, ***Studia Logica***, vol. 51 (1992), no. 3–4, pp. 551–578.

[12] D. Harel, *Dynamic logic*, ***Handbook of philosophical logic, volume ii*** (D. M. Gabbay and F. Guenthner, editors), D. Reidel Publishers, 1984, pp. 496–604.

[13] D. Harel, D. Kozen, and J. Tiuryn, ***Dynamic logic***, MIT Press, 2000.

[14] D. Harel, R. Rosner, and M. Vardi, *On the power of bounded concurrency III: Reasoning about programs*, ***Proceedings of the 5th annual IEEE symposium on logic in computer science*** (J. C. Mitchell, editor), IEEE Computer Society Press, 1990, pp. 478–488.

[15] D. Harel and R. Sherman, *Looping versus repeating in dynamic logic*, ***Information and control***, vol. 55 (1982), pp. 175–192.

[16] J. Johannsen and M. Lange, *CTL^+ is complete for double exponential time*, ***Icalp: Annual international colloquium on automata, languages and programming***, 2003.

[17] K. ABRAHAMSON, *Decidability and expressiveness of logics of processes*, ***Ph.D. thesis***, University of Washington, Seattle, 1980.

[18] D. C. KOZEN and J. TIURYN, *Logics of programs*, ***Handbook of theoretical computer science*** (J. van Leewen, editor), vol. B: Formal Models and Semantics, The MIT Press, 1990, pp. 789–840.

[19] M. LANGE, *A lower complexity bound for propositional dynamic logic with intersection*, ***Advances in modal logic volume 5*** (R. A. Schmidt, I. Pratt-Hartmann, M. Reynolds, and H. Wansing, editors), King's College Publications, 2005, To appear.

[20] C. LUTZ and U. SATTLER, *Mary likes all cats*, ***Proceedings of the 2000 international workshop in description logics (DL2000)*** (F. Baader and U. Sattler, editors), CEUR-WS (http://ceur-ws.org/), no. 33, 2000, pp. 213–226.

[21] C. LUTZ and D. WALTHER, *PDL with negation of atomic programs*, ***Journal of Applied Non-Classical Logic***, (2005), To appear.

[22] F. MASSACCI, *Decision procedures for expressive description logics with role intersection, composition and converse*, ***Proceedings of the seventeenth international conference on artificial intelligence (IJCAI-01)*** (San Francisco, CA) (B. Nebel, editor), Morgan Kaufmann Publishers, Inc., August 4–10 2001, pp. 193–198.

[23] A. J. MAYER and L. J. STOCKMEYER, *The complexity of PDL with interleaving*, ***Theoretical Computer Science***, vol. 161 (1996), no. 1-2, pp. 109–122.

[24] D. PELEG, *Concurrent dynamic logic*, ***Proceedings of the seventeenth annual ACM symposium on theory of computing, providence, rhode island, may 6–8, 1985***, ACM Press, 1985, pp. 232–239.

[25] V. PRATT, *Considerations on floyd-hoare logic*, ***Focs: Ieee symposium on foundations of computer science (focs)***, 1976.

[26] K. D. SCHILD, *A correspondence theory for terminological logics: Preliminary report*, ***Proceedings of the twelfth international joint conference on artificial intelligence (ijcai-91)*** (John Mylopoulos and Ray Reiter, editors), Morgan Kaufmann, 1991, pp. 466–471.

[27] ———, *Combining terminological logics with tense logic*, ***Progress in artificial intelligence – 6th portuguese conference on artificial intelligence, EPIA'93*** (M. Filgueiras and L. Damas, editors), Lecture Notes in Artificial Intelligence, vol. 727, Springer-Verlag, 1993, pp. 105–120.

[28] R. S. STREETT, *Propositional dynamic logic of looping and converse is elementarily decidable*, ***Information and Control***, vol. 54 (1982), no. 1–2, pp. 121–141.

[29] M. Y. VARDI, *The taming of converse: Reasoning about two-way computations*, ***Proceedings of the conference on logic of programs*** (Rohit Parikh, editor), LNCS, vol. 193, Springer, 1985, pp. 413–424.

[30] M. Y. VARDI and L. STOCKMEYER, *Improved upper and lower bounds for modal logics of programs*, ***Proceedings of the seventeenth annual ACM symposium on theory of computing, providence, rhode island, may 6–8, 1985***, ACM Press, 1985, pp. 240–251.

INSTITUT FÜR INFORMATIK,
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN,
GERMANY

INSTITUT FÜR INFORMATIK,
TECHNISCHE UNIVERSITÄT DRESDEN,
GERMANY