Model Checking Propositional Dynamic Logic with All Extras

Martin Lange

University of Munich, Institut für Informatik, Oettingenstr. 67, D-80538 München, Germany

Abstract

This paper presents a model checking algorithm for Propositional Dynamic Logic (PDL) with looping, repeat, test, intersection, converse, program complementation as well as context-free programs. The algorithm shows that the model checking problem for PDL remains PTIME-complete in the presence of all these operators, in contrast to the high increase in complexity that they cause for the satisfiability problem.

Key words: Propositional Dynamic Logic, Model Checking, Complexity

1 Introduction

Propositional Dynamic Logic (PDL) (1; 2; 3) was introduced by Fischer and Ladner (4) in the late 70s as a formalism for reasoning about programs. Soon afterwards the logic was outdated for that purpose through the introduction of the modal μ-calculus, a much more expressive logic with just little higher complexity. Also, other temporal logics like LTL or CTL have had greater success as specification logics because of their expressive power or just because of a syntax that is more appealing to non-logicians.

However, PDL has, by now, become a standard logic that, on the whole, is far from being outdated. There is hardly any other logic (apart from general-purpose predicate logic) that occurs in, has links to, and is used at the same time in different areas within computer science, artificial intelligence, mathematics, philosophy, and linguistics. It can be used in program verification (5), to describe the dynamic evolution of agent-based systems (6), for planning

Email address: `Martin.Lange@ifi.lmu.de` (Martin Lange).

(7) or knowledge engineering (8; 9), it has links to epistemic logics (10), it is closely related to description logics (11), etc.

The different contexts in which PDL or a close relative of it is used have led to the development of a number of extensions. PDL, in its pure form, is a multi-modal logic in which the accessibility relations in Kripke structures form a Kleene algebra, that is the closure of a finite set of binary relations under the operations union, relation composition and finite iteration. Due to PDL's original purpose, the elements of the Kleene algebra are called programs. Consequently, we call the nodes of a Kripke structure states.

There are two ways to enhance PDL: adding new operators on the formula level or on the program level. Here we consider PDL with the additional formulas $\texttt{repeat}(\alpha)$ and $\texttt{loop}(\alpha)$ (12). The former postulates the possibility to iterate α ad infinitum – clearly something useful in the specification of reactive systems. For example $\neg\langle(a \cup b)^*\rangle\texttt{repeat}(a)$ says that there is no run on which b happens finitely many times only.

The formula $\texttt{loop}(\alpha)$ is true in states of a Kripke structure that can run program α and get back to themselves. Hence, PDL with the loop-construct loses the tree model property and might therefore be less attractive for program verification because its formulas are not invariant under bisimulation anymore. However, in description logics programs correspond to roles, and with the loop-construct it is possible to define self-application as a concept. For example, the formula $\texttt{part} \sqsubseteq \neg\texttt{loop}(\texttt{before}^+)$ could say in an assembly process the relation determining the order in which parts are assembled, is well-founded.

Other interesting logics are obtained by enriching the Kleene algebra. The test-operator turns a formula into a program that stalls in a state not satisfying the formula. It can be used to model conditional branching: $\texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta \equiv \varphi?; \alpha \cup (\neg\varphi)?; \beta$.

The converse-operator runs a program backwards (12).It can be used to form consistency checks for example: $\texttt{is\_son\_of} \sqsubseteq (\texttt{father} \sqcup \texttt{mother})^{-1}$.

The intersection operator (13) can be used to reason about the parallel execution of two programs. For instance, $[(\texttt{read} \cup \texttt{write})^*]\neg\langle\texttt{read} \cap \texttt{write}\rangle\texttt{tt}$ could say that it is never possible that some data is both read and written at the same time.

The negation operator forms the complement of a program. The formula $[\overline{\texttt{is\_relative}}]\texttt{employable}$ for example could be used to check that every individual A who is not a relative of B can be employed by B.

Another way of enriching the program part of PDL has been taken by considering non-regular PDL (14). There, programs are composed from atomic ones as words of a context-free language – as opposed to regular languages obtained by using union, concatenation and the Kleene star only.

Clearly, adding operators to a logic can increase the complexity of the logic's decision problems. This is the case for PDL's satisfiability problem which is EXPTIME-complete in the presence of test and converse (4; 15; 16; 17). Adding the intersection operator on programs makes it 2EXPTIME-complete (18; 19). It becomes undecidable in the presence of the negation operator (3) or when non-regular programs are introduced (14).

The other main problem associated with a logic is the model checking problem: given an interpretation for a formula, does it satisfy the formula? Its importance varies with the context in which PDL is considered. Program verification for example is unimaginable without it. Description logic research has mainly focused on satisfiability problems, but their model checking problems also find a number of applications like contextual reasoning (20), information retrieval (21), etc. Epistemic properties are also used for correctness specifications and are verified using model checking (22; 23)

The model checking problem for pure PDL is PTIME-complete. For the lower bound even less is needed. Model checking modal logic K, i.e. PDL with one atomic program and no operations on it, is already PTIME-hard. Inclusion in PTIME was proved in (4). In fact, model checking PDL is possible even in linear time. Here we show that PDL's model checking problem remains in PTIME in the presence of the operators mentioned above. This is still true if programs are allowed to be combinations of context-free ones using all of the operators above.

We present a global model checking algorithm for Propositional Dynamic Logic with all the extras mentioned so far that only requires simple manipulations – known from linear algebra – on adjacency matrices representing Kripke structures.

2 Preliminaries

2.1 Adjacency Matrices

Let $\mathbb{B}$ be the boolean lattice of values $\{0, 1\}$ with partial order $0 \leq 1$, joins $a \vee b$, meets $a \wedge b$ and complementation $\overline{a}$. $\mathbb{B}^{n \times n}$ denotes the set of all matrices of size $n \times n$ for some $n \in \mathbb{N}$ with entries from $\mathbb{B}$. We will use capital letters like A for matrices. Their entries will either be denoted using indexing, e.g.

$(A)ij$, or corresponding lower case letters. E.g. a_{ij} denotes A's entry in the i-th row and the j-th column for $0 \leq i, j < n$.

The pointwise partial order on $\mathbb{B}^{n\times n}$ is defined by: $A \leq B$ iff for all $i, j = 0, \ldots, n-1$: $a_{ij} \leq b_{ij}$. $\mathbb{B}^{n\times n}$ together with $\leq$ also forms a boolean lattice. Joins and meets in $\mathbb{B}^{n\times n}$ are defined using $\vee$ and $\wedge$ pointwise, too.

The height of a lattice is the number of different elements in a maximal $\leq$-chain. The height of $\mathbb{B}^{n\times n}$ is therefore n^2.

There are two distinguished elements of $\mathbb{B}^{n\times n}$: $\mathbf{0}_n$ is the zero matrix with all entries 0, and $\mathbf{1}_n$ is the identity matrix with all entries 0 apart from those on the main diagonal which are set to 1.

A matrix of type $\mathbb{B}^{n\times 1}$, or $\mathbb{B}^n$ for short, is called a vector. We use letters $u, v, \ldots$ to denote vectors and subscripted letters u_i for their components.

2.2 Kripke Structures

Let $\mathcal{P} = \{p, q, \ldots\}$ be a finite set of *propositional constants*, and let $\Sigma = \{a, b, \ldots\}$ be a finite set of *atomic program names*. A Kripke structure is a triple $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \{\mathcal{I}_q \mid q \in \mathcal{P}\})$ with $\mathcal{S}$ being a set of *states*, $\xrightarrow{a}$ for every $a \in \Sigma$ is a binary relation on states, and $\mathcal{I}_q : 2^{\mathcal{S}}$ an interpretation of the proposition q in $\mathcal{K}$. We will restrict ourselves to finite structures.

Note that a Kripke structure is nothing more than a directed graph with labelled nodes and edges. Since it is assumed to be finite, $\mathcal{S}$ can be linearly ordered as $\{s_0, s_1, \ldots, s_{n-1}\}$ for some $n \in \mathbb{N}$. Furthermore, each subset of states $S \subseteq \mathcal{S}$ can be represented by a vector $v_S \in \mathbb{B}^n$. Thus, in the following we will assume a Kripke structure to be given as a sequence $\{\xrightarrow{a} \mid a \in \Sigma\}$ of adjacency matrices of type $\mathbb{B}^{n\times n}$ representing the accessibility relations, and a sequence $\{\mathcal{I}_q \mid q \in \mathcal{P}\}$ of vectors of type $\mathbb{B}^n$ representing the interpretation of each atomic proposition. For better readability we continue to write $s_i \in S$ instead of $(u_S)_i = 1$ where u_S represents $S \subseteq \mathcal{S}$. The same holds for pairs of states related by an accessibility relation and adjacency matrices: $s_i \xrightarrow{a} s_j$ instead of $(\xrightarrow{a})_{ij} = 1$.

2.3 Context-Free Grammars

Before we can define the syntax and semantics of PDL formally, we recall the definition of context-free grammars which are used to derive complex programs in PDL formulas.

A context-free grammar (CFG) (24) is a quadruple $G = (N, \Sigma, S, P)$ with N being a finite set of variable symbols, Σ a finite set of terminal symbols, $S \in N$ the starting symbol, and $P : N \to 2^{(N \cup \Sigma)^*}$ the set of production rules. P can be regarded as a system of equations over the variables N, with right-hand sides built from variables and atomic letters using language composition and union. The language $L(G) \subseteq \Sigma^*$ generated by G is the projection of P's least fixpoint solution onto the starting symbol S.

Given a CFG G, we write $|G|$ to denote its size, measured as the sum of the sizes of each production rule.

2.4 Propositional Dynamic Logic with All Extras

Formulas φ and *programs* α of PDL are defined simultaneously as follows.

$$\begin{array}{lcl} \varphi & ::= & q \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle\alpha\rangle\varphi \mid \texttt{loop}(\alpha) \mid \texttt{repeat}(\alpha) \\ \alpha & ::= & a \mid \alpha \cup \alpha \mid \alpha \cap \alpha \mid \alpha;\alpha \mid \alpha^* \mid \overline{\alpha} \mid \alpha^c \mid \varphi? \mid G \end{array}$$

where q ranges over $\mathcal{P}$, a ranges over Σ, and G is a context-free grammar (CFG) over the set Σ of terminal symbols.

Other formula operators can be introduced as abbreviations: $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$, $[\alpha]\varphi := \neg\langle\alpha\rangle\neg\varphi$, $\texttt{tt} := q \vee \neg q$ for some $q \in \mathcal{P}$, and $\texttt{ff} := \neg\texttt{tt}$.

PDL formulas are interpreted over Kripke structures $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \{\mathcal{I}_q \mid q \in \mathcal{P}\})$. The semantics of a PDL formula and a PDL program is explained by simultaneous induction on the size of formulas, resp. programs. Let $s, t \in \mathcal{S}$.

$$
\begin{array}{lll}
s \xrightarrow{\alpha;\beta} t & \text{iff} & \exists u \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} u \text{ and } u \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cup\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ or } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha\cap\beta} t & \text{iff} & s \xrightarrow{\alpha} t \text{ and } s \xrightarrow{\beta} t \\
s \xrightarrow{\alpha^*} t & \text{iff} & \exists n \in \mathbb{N}, s \xrightarrow{\alpha^n} t \text{ where} \\
 & & \forall s, t \in \mathcal{S} : s \xrightarrow{\alpha^0} s, \text{ and } s \xrightarrow{\alpha^{n+1}} t \text{ iff } s \xrightarrow{\alpha;\alpha^n} t \\
s \xrightarrow{\overline{\alpha}} t & \text{iff} & \text{not } s \xrightarrow{\alpha} t \\
s \xrightarrow{\alpha^c} t & \text{iff} & t \xrightarrow{\alpha} s \\
s \xrightarrow{\varphi?} s & \text{iff} & s \models \varphi \\
s \xrightarrow{G} t & \text{iff} & \exists w \in L(G), \text{ s.t. } w = a_1 \dots a_n \text{ for some } n \in \mathbb{N} \\
 & & \text{and } s \xrightarrow{a_1;\dots;a_n} t
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{K}, s \models q & \text{iff} & s \in \mathcal{I}_q \\
\mathcal{K}, s \models \varphi \vee \psi & \text{iff} & \mathcal{K}, s \models \varphi \text{ or } \mathcal{K}, s \models \psi \\
\mathcal{K}, s \models \neg\varphi & \text{iff} & \mathcal{K}, s \not\models \varphi \\
\mathcal{K}, s \models \langle\alpha\rangle\varphi & \text{iff} & \exists t \in \mathcal{S} \text{ s.t. } s \xrightarrow{\alpha} t \text{ and } t \models \varphi \\
\mathcal{K}, s \models \texttt{loop}(\alpha) & \text{iff} & s \xrightarrow{\alpha} s \\
\mathcal{K}, s \models \texttt{repeat}(\alpha) & \text{iff} & \exists s_0, s_1, s_2, \dots, \text{ s.t. } s = s_0 \text{ and } \forall i \in \mathbb{N} : s_i \xrightarrow{\alpha} s_{i+1}
\end{array}
$$

Later we will represent the semantics of a formula φ w.r.t. a Kripke structure $\mathcal{K}$ – i.e. the set of its states satisfying it – by a boolean vector as mentioned above. In such a case, $\mathcal{K}, s \models \varphi$ is a synonym for inclusion of s in this set.

3 Operations on Matrices and Vectors

The model checking algorithm for PDL uses adjacency matrices and boolean vectors to represent programs and sets of states. Manipulations of these are carried out using the following operations, most of which are standard.

Definition 1 *Let $A, B, C \in \mathbb{B}^{n\times n}$, and $u, v \in \mathbb{B}^n$ for some $n \in \mathbb{N}$.*

union:	$C = A \vee B$	*iff*	$\forall i,j = 0,\dots,n-1$: $c_{ij} = a_{ij} \vee b_{ij}$
intersection:	$C = A \wedge B$	*iff*	$\forall i,j = 0,\dots,n-1$: $c_{ij} = a_{ij} \wedge b_{ij}$
composition:	$C = A \times B$	*iff*	$\forall i,j = 0,\dots,n-1$: $c_{ij} = 1$ *iff* $\exists k$ *s.t.* $a_{ik} = b_{kj} = 1$
+-closure:	$C = A^+$	*iff*	$C = \bigvee_{k\geq 1} A^k$ *where* $A^1 := A$ *and* $A^{k+1} := A \times A^k$
$*$-closure:	$C = A^*$	*iff*	$C = \mathbf{1}_n \vee A^+$
converse:	$C = A^c$	*iff*	$\forall i,j = 0,\dots,n-1$: $c_{ij} = a_{ji}$
negation:	$C = \overline{A}$	*iff*	$\forall i,j = 0,\dots,n-1$: $c_{ij} = \overline{a_{ij}}$
negation:	$u = \overline{v}$	*iff*	$\forall i = 0,\dots,n-1$: $u_i = \overline{v_i}$
diamond:	$u = A \times v$	*iff*	$\forall i = 0,\dots,n-1$: $u_i = \bigvee_{j=0}^{n-1} a_{ij} \wedge v_j$
diag:	$u = \mathit{diag}(A)$	*iff*	$\forall i = 0,\dots,n-1$: $u_i = a_{ii}$
tilt:	$A = \mathit{tilt}(v)$	*iff*	$\forall i = 0,\dots,n-1$: $a_{ii} = v_i$ *and* $\forall j \neq i$: $a_{ij} = 0$
lasso:	$u = A^{\multimap}$	*iff*	$u = A^* \times \mathit{diag}(A^+)$

For the overall complexity of the model checking procedure for PDL it is crucial that these operations can be computed efficiently.

Lemma 2 *For matrices in $\mathbb{B}^{n\times n}$ and vectors in $\mathbb{B}^n$, the operations union, intersection, composition, closure, converse, negation, diamond, diag, tilt, and lasso can be carried out in time polynomial in n.*

PROOF. Union and intersection are defined pointwise, i.e. they require time $O(n^2)$. Composition is the usual matrix product with $\wedge$ as scalar multiplication and $\vee$ as scalar addition. Thus, it can be done in time $O(n^3)$ or better using a technique like Strassen's algorithm (25).

The transitive closure A^+ of a matrix A does not need a possibly unbounded union. Instead, it can be computed using Warshall's algorithm in time $O(n^3)$ (26), and, hence, the reflexive and transitive closure A^* as well.

The converse of a matrix is easily built in time $O(n^2)$ by swapping the indices of each entry. Negation takes time $O(n^2)$ on matrices and $O(n)$ on vectors by changing every entry. The diamond computation is the normal product of a matrix with a vector and can, hence, be done in time $O(n^2)$. The diag operation simply returns the main diagonal of a matrix as a vector, hence, it is possible in time $O(n)$. The tilt operation takes a vector, makes it the main diagonal of a matrix and sets all other entries to 0. It is possible in time $O(n^2)$.

Given all this, the lasso operation can be carried out in time $O(n^3)$, too.

This covers the cases of all program operators apart from context-free grammars. Those can also be computed in polynomial time using fixpoint iteration. For this to work, we need to know that the corresponding mapping defined by a context-free grammar is monotonic.

Lemma 3 *The operations union and composition are monotonic on $\mathbb{B}^{n\times n}$ w.r.t. $\leq$.*

PROOF. The operations $\vee$ and $\wedge$ are monotonic on $\mathbb{B}$, hence, monotonicity carries over to $\mathbb{B}^{n\times n}$ for any operation defined only in terms of these.

Lemma 4 *Given matrices $\xrightarrow{a} \in \mathbb{B}^{n\times n}$ for any $a \in \Sigma$, and a context-free grammar $G = (N, \Sigma, S, P)$, it is possible to compute $\xrightarrow{G}$ in time polynomial in n.*

PROOF. We will associate with G a system of equations $\|G\|$ over the variables in N of type $\mathbb{B}^{n\times n}$. For each $X \in N$, $\|G\|$ contains an equation of the form $X = \|\bigcup P(X)\|$ where

$$\begin{array}{lll} \|a\| & := \xrightarrow{a} & \text{if } a \in \Sigma \\ \|X\| & := X & \text{if } X \in N \\ \|w \cup v\| & := \|w\| \vee \|v\| & \\ \|yw\| & := \|y\| \times \|w\| & \text{if } y \in \Sigma \cup N \end{array}$$

According to Lemma 3, all right-hand sides of $\|G\|$ are monotonic in each variable. According to the Knaster-Tarski-Theorem (27), $\|G\|$ possesses a unique least solution that maps each $X \in N$ to an element of $\mathbb{B}^{n\times n}$. If $N = \{X_1, \ldots, X_m\}$ then we write $\mu(X_1, \ldots, X_m).\|G\|$ to denote this solution and $\mu X_i.\|G\|$ for its projection onto X_i. Now, $\xrightarrow{G} = \mu S.\|G\|$ follows immediately from the equation $\mu S.\|G\| = \bigcup_{w\in L(G)} \xrightarrow{w}$ which holds true by definition.

What remains to be seen is how $\mu S.\|G\|$ can be computed efficiently using simultaneous fixpoint iteration in the boolean lattice $\mathbb{B}^{n\times n}$. Let for all $i =$

$1, \ldots, m$:

$$\begin{aligned} \mu^0(X_1, \ldots, X_m).\|G\| &:= \mathbf{0}_n \\ \mu^{k+1}(X_1, \ldots, X_m).\|G\| &:= \|G\|\big[\mu^k X_1.\|G\|/X_1, \ldots, \mu^k X_m.\|G\|/X_m\big] \end{aligned}$$

where the latter denotes simultaneous substitution of smaller approximants for the according variables. Because of monotonicity we have

$$\mu(X_1, \ldots, X_m).\|G\| = \bigcup_{k \in \mathbb{N}} \mu^k(X_1, \ldots, X_m).\|G\|$$

However, the height of $\mathbb{B}^{n \times n}$ is n^2 and, hence, the fixpoint is found after no more than n^2 iterations. Evaluating each right-hand side of an equation can be done in time $O(|G| \cdot n^3)$ according to Lemma 2. Therefore, $\mu S.\|G\|$ can be computed in time $O(|G| \cdot n^5)$.

4 The Model Checking Problem

Proposition 5 *The model checking problem for PDL is PTIME-hard.*

In fact, it does not take much to achieve PTIME-hardness. The model checking problem for modal logic K can be shown to be PTIME-hard by reduction from the alternating graph reachability problem (28): given a $k \in \mathbb{N}$ and a graph $G = (V, E)$ with two nodes s, t, two players alternatingly move a pebble along an edge starting from s. The question is to decide whether or not the first player can force the pebble onto node t.

Modal logic K can be obtained from PDL by replacing $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ syntactically with $\Diamond\varphi$ and $\Box\varphi$. This means there is only one atomic program whose name is irrelevant and no program constructs. Then, the first player has a winning strategy from s iff

$$s \models q_t \vee \Diamond(q_t \vee \Box(q_t \vee \Diamond(q_t \vee \Box(q_t \vee \ldots \Diamond(q_t \vee \Box q_t) \ldots))))$$

where q_t is true exactly in node t, and the depth of this formula is $|V|$.

This result is well-known. We only include it here in order to stress the following result: model checking PDL remains in PTIME even if all the extra program constructs and formula operators mentioned above are allowed.

Figure 1 presents the model checking algorithm `MC`. It assumes a finite Kripke structure $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \{\mathcal{I}_q \mid q \in \mathcal{P}\})$ to be given, and recurses on the structure of the input formula just like a standard model checking procedure for multi-modal logic. It deals with complex programs using the procedure

Prog which returns the adjacency matrix representing the accessibility relation of any program. Finally, the procedure CFG uses the standard fixpoint iteration from Lemma 4 to compute the least fixpoint of the equational system corresponding to a context-free grammar G.

W.l.o.g. we assume $G = (\{X_1, \ldots, X_k\}, \Sigma, X_1, P)$ for some $k \in \mathbb{N}$, s.t. for all $i = 1, \ldots, k$: $P(X_i) = \phi_i$ for some expression ϕ_i of the form $w_1 \cup \ldots \cup w_m$ with $w_j \in (\Sigma \cup \{X_1, \ldots, X_k\})^*$. Function eval takes an expression of the form

$$(A_{11} \times \ldots \times A_{1m_1}) \vee \ldots \vee (A_{l1} \times \ldots \times A_{lm_l})$$

over matrices and simply evaluates it using the operations union and composition.

Theorem 6 *Given a Kripke structure $\mathcal{K} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \{\mathcal{I}_q \mid q \in \mathcal{P}\})$, a formula φ or a program α, we have $s \in$* MC(φ) *iff $\mathcal{K}, s \models \varphi$, and* Prog($\alpha$) $= \xrightarrow{\alpha}$.

PROOF. Assume $\mathcal{S} = \{s_0, \ldots, s_{n-1}\}$. The claim is proved by simultaneous induction on the structure of the formula φ and the program α. We deal with the formulas first.

Formulas: The claim is trivially true for atomic propositions, and follows immediately from the hypothesis for the cases of $\varphi = \psi_0 \vee \psi_1$ and $\varphi = \neg\psi$.

Case $\varphi = \langle\alpha\rangle\psi$. According to the hypothesis concerning programs, Prog(α) correctly computes the adjacency matrix $\xrightarrow{\alpha}$. Furthermore, $s_j \in$ MC(ψ) iff $s_j \models \psi$ for any $s_j \in \mathcal{S}$. Now take any state $s_i \in \mathcal{S}$. We have $s_i \in$ MC(φ) iff

$$\bigvee_{k=0}^{n-1} (\xrightarrow{\alpha})_{ik} \;\wedge\; \mathtt{MC}(\psi)_k \;=\; 1$$

Hence, $s_i \in$ MC(φ) iff there is a $s_k \in$ MC(ψ) with $s_i \xrightarrow{\alpha} s_k$, i.e. $s_i \models \langle\alpha\rangle\psi$.

Case $\varphi =$ loop(α). Again, Prog(α) yields a representation for $\xrightarrow{\alpha}$ according to the hypothesis. Then $s_i \in \mathit{diag}(\xrightarrow{\alpha})$ iff $s_i \xrightarrow{\alpha} s_i$ iff $s_i \models$ loop(α).

Case $\varphi =$ repeat(α). By the hypothesis we have Prog(α) $= \xrightarrow{\alpha}$. Now, $\xrightarrow{\alpha}{}^+$ represents the transitive closure of $\xrightarrow{\alpha}$, and, hence, $\mathit{diag}(\xrightarrow{\alpha}{}^+)$ represents all

```
global $\{\xrightarrow{a} : \mathbb{B}^{n\times n} \mid a \in \Sigma\}, \{\mathcal{I}_q : \mathbb{B}^n \mid q \in \mathcal{P}\}$

procedure MC($\varphi$): $\mathbb{B}^n$
  case $\varphi$ of
      $q$                      $\longrightarrow$  $I_q$
      $\psi_0 \vee \psi_1$     $\longrightarrow$  MC($\psi_0$) $\vee$ MC($\psi_1$)
      $\neg\psi$               $\longrightarrow$  $\overline{\texttt{MC}(\psi)}$
      $\langle\alpha\rangle\psi$  $\longrightarrow$  Prog($\alpha$) $\times$ MC($\psi$)
      loop($\alpha$)           $\longrightarrow$  $diag$(Prog($\alpha$))
      repeat($\alpha$)         $\longrightarrow$  Prog($\alpha$)$^{\multimap}$

procedure Prog($\alpha$): $\mathbb{B}^{n\times n}$
  case $\alpha$ of
      $a$                      $\longrightarrow$  $\xrightarrow{a}$
      $\beta_0 \cup \beta_1$   $\longrightarrow$  Prog($\beta_0$) $\vee$ Prog($\beta_1$)
      $\beta_0 \cap \beta_1$   $\longrightarrow$  Prog($\beta_0$) $\wedge$ Prog($\beta_1$)
      $\beta_0;\beta_1$        $\longrightarrow$  Prog($\beta_0$) $\times$ Prog($\beta_1$)
      $\beta^*$                $\longrightarrow$  Prog($\beta$)$^*$
      $\overline{\beta}$       $\longrightarrow$  $\overline{\texttt{Prog}(\beta)}$
      $\beta^c$                $\longrightarrow$  Prog($\beta$)$^c$
      $\varphi?$               $\longrightarrow$  $tilt$(MC($\varphi$))
      $G$                      $\longrightarrow$  CFG($\|G\|$)

procedure CFG($E = \{X_i = \phi_i \mid i = 1,\ldots,k\}$): $\mathbb{B}^{n\times n}$
  for $i = 1,\ldots,k$
    $X_i^0 := \mathbf{0}_n$
  $j := 0$
  repeat
    $j := j + 1$
    for $i = 1,\ldots,k$
      $X_i^j :=$ eval($\phi_i\Big[X_1^{j-1}/X_1,\ldots,X_k^{j-1}/X_k\Big]$)
  until for all $i = 1,\ldots,k$: $X_i^j = X_i^{j-1}$
  return $X_1^j$
```

Fig. 1. A model checking procedure for PDL.

the states that are reachable from themselves through an arbitrary and non-zero number of α-steps. Finally, $\xrightarrow{\alpha}^{\multimap}$ represents all states from which an α-cycling state is reachable via α-steps. Clearly, these are all the states in a finite model from which an infinite sequence of α-transitions emerges. Therefore, $s \in$ MC(repeat(α)) iff $\mathcal{K}, s \models$ repeat(α).

Programs: Again, the claim is trivially true for atomic programs $\alpha = a$, and follows immediately from the hypothesis for the cases of $\alpha = \beta_0 \cup \beta_1$, $\alpha = \beta_0 \cap \beta_1$, $\alpha = \overline{\beta}$, and $\alpha = \beta^c$.

Case $\alpha = \beta_0; \beta_1$. By the hypothesis we have $\texttt{Prog}(\beta_i) = \xrightarrow{\beta_i}$ for $i = 0, 1$. Moreover, for all $i, j = 0, \ldots, n-1$ we have $(s_i, s_j) \in \texttt{Prog}(\alpha)$ iff $(\xrightarrow{\beta_0} \times \xrightarrow{\beta_1})_{ij} = 1$ iff there is a k s.t. $(\xrightarrow{\beta_0})_{ik} = (\xrightarrow{\beta_1})_{kj} = 1$ iff there is a state s_k s.t. $s_i \xrightarrow{\beta_0} s_k$ and $s_k \xrightarrow{\beta_1} s_j$ iff $s_i \xrightarrow{\beta_0;\beta_1} s_j$.

Case $\alpha = \beta^*$. The claim follows immediately from the hypothesis and the fact that B^* represents the reflexive and transitive closure of B.

Case $\alpha = \varphi?$. According to the hypothesis we have $s \in \texttt{MC}(\varphi)$ iff $\mathcal{K}, s \models \varphi$. Then for any $i, j = 0, \ldots, n-1$ we have $\mathit{tilt}(\texttt{MC}(\varphi))_{ij} = 1$ iff $i = j$ and $\mathcal{K}, s_i \models \varphi$, i.e. $s_i \xrightarrow{\varphi?} s_j$.

Case $\alpha = G$. This case does not need the hypothesis. Instead, note that procedure `CFG` iteratively computes the representations of approximants X_i^j to the languages $L(G_i)$ where $G_i := (\{X_1, \ldots, X_k\}, \Sigma, X_i, P)$. Furthermore, it returns X_1^j only if $X_i^j = X_i^{j-1}$ for all i, i.e. when the least fixpoint is found. But termination of this procedure is guaranteed by Lemma 3 and the fact that $\mathbb{B}^{n \times n}$ has finite height only.

Theorem 7 *The model checking problem for PDL is in PTIME.*

PROOF. Let $\mathcal{K}$ be a Kripke structure with state set $\mathcal{S}$, $n := |\mathcal{S}|$, and φ be a PDL formula. According to Theorem 6, $\texttt{MC}(\varphi)$ computes all the states s s.t. $\mathcal{K}, s \models \varphi$. Note that each subformula and subprogram of φ is only visited once by either `MC`, `Prog` or `CFG`. Furthermore, according to Lemmas 2 and 4, all the operations needed in each case can be done in time at most $O(|\varphi| \cdot n^5)$. Thus, the overall running time of algorithm `MC` is bounded by $O(|\varphi|^2 \cdot n^5)$.

5 Conclusion

We have shown that the model checking problem for Propositional Dynamic Logic is still in PTIME even in the presence of additional formula or program operators. The presented algorithm is global in the sense that it computes, given a Kripke structure and a PDL formula, all states satisfying the formula. It remains to be seen whether this algorithm can be transformed into a local one, i.e. one that traverses the Kripke structure on demand only. We believe that transforming context-free grammars into Greibach normal form (29) is a helpful step towards a local algorithm.

This could also solve the question of whether or not there is an asymptotically better algorithm than the one presented here.

It also remains to be seen how the set of program operators can be enriched whilst still having a polynomial time model checking problem. A natural way is to consider richer classes of formal languages generated by alternating context-free grammars (30), conjunctive grammars (31), context-sensitive grammars, etc.

Another program construct that is not considered here but has occurred in the literature is the interleaving operator (32). The interleaving of programs α and β is the union over all sequences of atomic steps within α and β, preserving their respective orders. We did non include it here because the interpretation of the combination of interleaving and intersection would be arbitrary.

References

[1] D. Kozen, J. Tiuryn, Logics of programs, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier and MIT Press, New York, USA, 1990, Ch. 14, pp. 789–840.

[2] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.

[3] D. Harel, Dynamic logic, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, Vol. II: Extensions of Classical Logic, Reidel, Dordrecht, 1984, Ch. 10, pp. 497–604.

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

[5] M. Y. Vardi, P. Wolper, Automata-theoretic techniques for modal logic of programs, Journal of Computer and System Sciences 32 (1986) 183–221.

[6] J.-J. C. Meyer, Dynamic logic reasoning about actions and agents, in: Proc. Workshop on Logic-Based Artificial Intelligence, Washington, D.C., USA, 1999.

[7] L. Spalazzi, P. Traverso, A dynamic logic for acting, sensing, and planning, JLC: Journal of Logic and Computation 10.

[8] F. van Harmelen, J. Balder, $(ML)^2$: a formal language for KADS models of expertise, Knowledge Acquisition 4 (1992) 127–161.

[9] D. Fensel, The Knowledge Acquisition and Representation Language KARL, Kluwer Academic, New York, 1995.

[10] H. P. van Ditmarsch, W. van der Hoek, B. P. Kooi, Concurrent dynamic epistemic logic for MAS, in: Proc. 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, Melbourne, Australia, 2003, pp. 201–208.

[11] G. D. Giacomo, M. Lenzerini, Boosting the correspondence between description logics and propositional dynamic logics, in: Proc. of the 12th National Conference on Artificial Intelligence, AAAI'94, AAAI-Press/the MIT-Press, 1994, pp. 205–212.

[12] R. S. Streett, Propositional dynamic logic of looping and converse, in: Proc. 13th Symp. on Theory of Computation, STOC'81, ACM, Milwaukee, Wisconsin, 1981, pp. 375–383.

[13] D. Harel, Recurring dominoes: Making the highly undecidable highly understandable, Annals of Discrete Mathematics 24 (1985) 51–72.

[14] D. Harel, A. Pnueli, J. Stavi, Propositional dynamic logic of nonregular programs, Journal of Computer and System Sciences 26 (2) (1983) 222–243.

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

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

[17] M. Y. Vardi, The taming of converse: Reasoning about two-way computations, in: R. Parikh (Ed.), Proc. Workshop on Logic of Programs, Vol. 193 of LNCS, Springer, Brooklyn, NY, 1985, pp. 413–424.

[18] S. Danecki, Nondeterministic propositional dynamic logic with intersection is decidable, in: A. Skowron (Ed.), Proc. 5th Symp. on Computation Theory, Vol. 208 of LNCS, Springer, Zaborów, Poland, 1984, pp. 34–53.

[19] M. Lange, A lower complexity bound for propositional dynamic logic with intersection, to appear in: R. A. Schmidt, I. Pratt-Hartmann, M. Reynolds, H. Wansing (Eds.), Advances in Modal Logic Volume 5, King's College Publications, 2005.

[20] K. Striegnitz, Model checking for contextual reasoning in NLG, in: P. Blackburn, M. Kohlhase (Eds.), Proc. of Inference in Computational Semantics, ICoS-3, 2001, pp. 101–115.

[21] M. Bertini, A. D. Bimbo, W. Nunziati, Model checking for detection of sport highlights, in: Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, MIR'03, ACM, 2003, pp. 215–222.

[22] W. van der Hoek, M. Wooldridge, Model checking knowledge and time, in: D. Bosnacki, S. Leue (Eds.), Proc. 9th Int. SPIN Workshop on Model Checking of Software, SPIN'02, Vol. 2318 of LNCS, 2002, pp. 95–111.

[23] W. Penczek, A. Lomuscio, Verifying epistemic properties of multi-agent systems via bounded model checking, Fundamenta Informatica 55 (2) (2003) 167–185.

[24] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, 1996.

[25] V. Strassen, Gaussian elimination is not optimal, Numerische Mathe-

matik 13 (1969) 354–356.

[26] S. Warshall, A theorem on boolean matrices, Journal of the ACM 9 (1) (1962) 11–12.

[27] A. Tarski, A lattice-theoretical fixpoint theorem and its application, Pacific Journal of Mathematics 5 (1955) 285–309.

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

[29] S. A. Greibach, A new normal form theorem for context-free phrase structure grammars, Journal of the ACM 12 (1) (1965) 42–52.

[30] O. H. Ibarra, T. Jiang, H. Wang, A characterization of exponential-time languages by alternating context-free grammars, TCS 99 (2) (1992) 301–315.

[31] A. Okhotin, Conjunctive grammars, Journal of Automata, Languages and Combinatorics 6 (4) (2001) 519–535.

[32] A. J. Mayer, L. J. Stockmeyer, The complexity of PDL with interleaving, TCS 161 (1–2) (1996) 109–122.