Infinite State Model-Checking of Propositional Dynamic Logics

Stefan Göller and Markus Lohrey

FMI, Universität Stuttgart, Germany
{goeller, lohrey}@informatik.uni-stuttgart.de

Abstract. Model-checking problems for PDL (propositional dynamic logic) and its extension PDL$^\cap$ (which includes the intersection operator on programs) over various classes of infinite state systems (BPP, BPA, pushdown systems, prefix-recognizable systems) are studied. Precise upper and lower bounds are shown for the data/expression/combined complexity of these model-checking problems.

1 Introduction

Propositional Dynamic Logic (PDL) was introduced by Fischer and Ladner in 1979 as a modal logic for reasoning about programs [10]. In PDL, there are two syntactic entities: formulas and programs. Formulas are interpreted in nodes of a Kripke structure and can be built up from atomic propositions using boolean connectives. Programs are interpreted by binary relations over the node set of a Kripke structure and can be built up from atomic programs using the operations of union, composition, and Kleene hull (reflexive transitive closure). PDL contains two means for connecting formulas and programs: Programs may appear in modalities in front of formulas, i.e., if π is a program and φ is a formula, then $\langle\pi\rangle\varphi$ is true in a node u if there exists a node v, where φ holds and which can be reached from u via the program π. Moreover, PDL allows to construct programs from formulas using the test operator: If φ is a formula, then the program $\varphi?$ is the identity relation on the node set restricted to those nodes where φ holds. Since its invention, many different extensions of PDL were proposed, mainly by allowing further operators on programs, like for instance the converse or intersection operator, see the monograph [13] for a detailed exposition. Recently, PDL, where programs are defined via visibly pushdown automata, was investigated [18]. PDL and its variations found numerous applications, e.g., in program verification, agent-based systems, and XML-querying. In AI, PDL received attention by its close relationship to description logics and epistemic logic, see [16] for references.

In the early days of PDL, researchers mainly concentrated on satisfiability problems and axiomatization of PDL and its variants. With the emergence of automatic verification, also model-checking problems for modal logics became a central research topic, and consequently model-checking problems for PDL attracted attention [16]. In this paper, we start to investigate model-checking problems for PDL over infinite state systems. In recent years, verification of infinite state systems became a major topic in the model-checking community. Usually, infinite state systems, like for instance systems with unbounded communication buffers or unbounded stacks, are modeled by some kind of abstract machine, which defines an infinite transition system (Kripke structure): nodes correspond to system states and state transitions of the system are modeled by labeled edges. Various classes of (finitely presented) infinite transition systems were studied under the model-checking perspective in the past, see e.g. [25] for a survey. In [22] Mayr introduced a uniform classification of infinite state systems in terms of two basic operations: parallel and sequential composition. In this paper, we will mainly follow Mayr's classification.

We believe that model-checking of PDL and its variants over infinite state systems is not only a natural topic, but also a useful and applicable research direction in verification. PDL allows directly to express regular reachability properties, which were studied e.g. in [19,22,30] in the context of infinite state systems. For instance, consider the property that a process can reach a state, where a condition φ holds, via a path on which the action sequence $a_1 a_2 \cdots a_n$ is repeated cyclically. Clearly, this can be expressed in CTL (if φ can be expressed in CTL), but we think that the PDL-formula $\langle (a_1 \circ a_2 \circ \cdots \circ a_n)^* \rangle \varphi$ is a more readable specification. Secondly, and more important, the extension of PDL with the intersection operator on programs [12], PDL$^\cap$ for short, allows to formulate natural system properties that cannot be expressed in the modal μ-calculus (since they do not have the tree model property), like for instance that a system can be reset to the current state (Example 2) or that two forking processes may synchronize in the future (Example 3).

In Section 5 we study model-checking problems for PDL and its variants over infinite state systems. For infinite state systems with parallel composition, PDL immediately becomes undecidable. More precisely, we show that PDL becomes undecidable over BPP (basic parallel processes), which correspond to Petri nets, where every transition needs exactly one token for firing (Proposition 1). This result follows from the undecidability of the model-checking problem for EF (the fragment of CTL, which only contains next-modalities and the "exists finally"-modality) for Petri nets [8]. Due to this undecidability result we mainly concentrate on infinite state systems with only sequential composition. In Mayr's classification these are pushdown systems (PDS) and basic process algebras (BPA), where the latter correspond to stateless pushdown systems. Pushdown systems were used to model the state space of programs with nested procedure calls, see e.g. [9]. Model-checking problems for pushdown systems were studied for various temporal logics (LTL, CTL, modal μ-calculus) [1,9,15,28,29]. We also include prefix-recognizable systems (PRS) into our investigation [3,5], which extend pushdown systems. Model-checking problems for prefix-recognizable systems were studied e.g. in [4,14]. The decidability of PDL and even PDL$^\cap$ for prefix-recognizable systems (and hence also BPA and PDS) follows from the fact that monadic second-order logic (MSO) is decidable for these systems and that PDL$^\cap$ can be easily translated into MSO. But from the viewpoint of complexity, this approach is quite unsatisfactory, since it leads to a nonelementary algorithm. On the other hand, for PDL (without the intersection operator) it turns out that based on the techniques of Walukiewicz for model-checking CTL and EF over pushdown systems [28], we can obtain sharp (elementary) complexity bounds: Whereas test-free PDL behaves w.r.t. to the complexity of the model-checking problem exactly in the same way as EF (PSPACE-complete in most cases), PDL with the test-operator is more difficult (EXP-complete in most cases).

The analysis of $\mathrm{PDL}^{\cap}$ turns out to be more involved. This is not really surprising. $\mathrm{PDL}^{\cap}$ turned out to be notoriously difficult in the past. It does not have the tree model property, and as a consequence the applicability of tree automata theoretic methods is quite limited. Whereas PDL is translatable into the modal μ-calculus, $\mathrm{PDL}^{\cap}$ is orthogonal to the modal μ-calculus with respect to expressiveness. A very difficult result of Danecki states that satisfiability of $\mathrm{PDL}^{\cap}$ is in 2EXP [7]. Only recently, a matching lower bound was obtained by Lange and Lutz [17]. Our main result of this paper states that the expression/combined complexity of $\mathrm{PDL}^{\cap}$ (and also the test-free fragment of $\mathrm{PDL}^{\cap}$) over BPA/PDS/PRS is 2EXP-complete, whereas the data complexity goes down to EXP. For the 2EXP lower bound proof, we use a technique from [28] for describing a traversal of the computation tree of an alternating Turing machine in CTL using a pushdown. The main difficulty that remains is to formalize in $\mathrm{PDL}^{\cap}$ that two configurations of an exponential space alternating Turing machine (these machines characterize 2EXP) are successor configurations. For the upper bound, we transform a $\mathrm{PDL}^{\cap}$ formula φ into a two-way alternating tree automaton A of exponential size, which has to be tested for emptiness. Since emptiness of two-way alternating tree automata can be checked in exponential time [27], we obtain a doubly exponential algorithm. Most of the inductive construction of A from φ uses standard constructions for two-way alternating tree automata. It is no surprise that the intersection operator is the difficult part in the construction of φ. The problem is that two paths from a source node s to a target node t, where the first (resp. second) path is a witness that (s, t) belongs to the interpretation of a program π_1 (resp. π_2) may completely diverge. This makes it hard to check for an automaton whether there is both a π_1-path and a π_2-path from s to t. Our solution is based on a subtle analysis of such diverging paths in pushdown systems.

One might argue that the high complexity (2EXP-completeness) circumvents the application of $\mathrm{PDL}^{\cap}$ model checking for pushdown systems. But note that the data complexity (which is a better approximation to the "real" complexity of model-checking, since formulas are usually small) of $\mathrm{PDL}^{\cap}$ over pushdown systems is only EXP, which is the same as the data complexity of CTL [28]. Moreover, to obtain an exponential time algorithm for $\mathrm{PDL}^{\cap}$ it is not really necessary to fix the formula, but it suffices to bound the nesting depth of intersection operators in programs. One may expect that this nesting depth is small in natural formulas, like in Example 2 or 3 (where it is 1). Table 1 gives an overview on our results. Proofs can be found in the technical report [11].

2 Preliminaries

Let Σ be a finite alphabet and let ε denote the empty word. Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and let $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ be a *disjoint copy* of Σ. For a word $w = a_1 \cdots a_n \in \Sigma^*$ ($a_1, \ldots, a_n \in \Sigma$) let $w^{\mathrm{rev}} = a_n \cdots a_1$. For $L \subseteq \Sigma^*$ let $L^{\mathrm{rev}} = \{w^{\mathrm{rev}} \mid w \in L\}$. Let $R, U \subseteq A \times A$ be binary relations over the set A. Then R^* is the *reflexive and transitive closure of* R. The *composition* of R and U is $R \circ U = \{(a, c) \in A \times A \mid \exists b \in A : (a, b) \in R \wedge (b, c) \in U\}$. Let $f : A \to C$ and $g : B \to C$ be functions, where $A \cap B = \emptyset$. The *disjoint union* $f \uplus g : A \cup B \to C$ of f and g is defined by $(f \uplus g)(a) = f(a)$ for $a \in A$ and $(f \uplus g)(b) = g(b)$ for $b \in B$. Let $A^B = \{f \mid f : B \to A\}$ be the set of all functions from B to A.

We assume that the reader is familiar with standard complexity classes like P (deterministic polynomial time), PSPACE (polynomial space), EXP (deterministic exponential time), and 2EXP (deterministic doubly exponential time), see [24] for more details. Hardness results are always meant w.r.t. logspace reductions. An *alternating Turing machine (ATM)* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ where (i) $Q = Q_{\mathrm{acc}} \uplus Q_{\mathrm{rej}} \uplus Q_\exists \uplus Q_\forall$ is a finite set of *states* Q which is partitioned into *accepting* (Q_{acc}), *rejecting* (Q_{rej}), *existential* ($Q_\exists$) and *universal* ($Q_\forall$) states, (ii) Γ is a *finite tape alphabet*, (iii) $\Sigma \subseteq \Gamma$ is the *input alphabet*, (iv) $q_0 \in Q$ is the *initial state*, (v) $\Box \in \Gamma \setminus \Sigma$ is the *blank symbol*, and (vi) the map $\delta : (Q_\exists \cup Q_\forall) \times \Gamma \to \mathrm{Moves} \times \mathrm{Moves}$ with $\mathrm{Moves} = Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ assigns to every pair $(q, \gamma) \in (Q_\exists \cup Q_\forall) \times \Gamma$ a pair of *moves*. If $\delta(q, a) = ((q_1, a_1, d_1), (q_2, a_2, d_2))$, then this means that if $\mathcal{M}$ is in state q and reads the symbol a, then the left (right) successor configuration of the current configuration results by writing a_1 (a_2), the read-write head moves in direction d_1 (d_2), and the new state is q_1 (q_2). A configuration of $\mathcal{M}$ is a word from $\Gamma^* Q \Gamma^+$. A configuration c of $\mathcal{M}$, where the current state is q, is *accepting* if (i) $q \in Q_{\mathrm{acc}}$ or (ii) $q \in Q_\exists$ and there exists an accepting successor configuration of c or (iii) $q \in Q_\forall$ and both successor configurations of c are accepting. The machine $\mathcal{M}$ accepts an input w if and only if the initial configuration $q_0 w$ is accepting.

3 Propositional Dynamic Logic and Extensions

Formulas of propositional dynamic logic (PDL) are interpreted over *Kripke structures*: Let $\mathbb{P}$ be a set of *atomic propositions* and let Σ a set of *atomic programs*. A *Kripke structure* over $(\mathbb{P}, \Sigma)$ is a tuple $\mathcal{K} = (S, \{\to_\sigma \mid \sigma \in \Sigma\}, \rho)$ where (i) S is a set of *nodes*, (ii) $\to_\sigma \subseteq S \times S$ is a *transition relation* for all $\sigma \in \Sigma$ and (iii) $\rho : S \to 2^{\mathbb{P}}$ labels every node with a set of atomic propositions. Formulas and programs of the logic PDL$^\cap$ (PDL with intersection) over $(\mathbb{P}, \Sigma)$ are defined by the following grammar, where $p \in \mathbb{P}$ and $\sigma \in \Sigma$:

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\pi\rangle\, \varphi \\ \pi &::= \sigma \mid \pi_1 \cup \pi_2 \mid \pi_1 \cap \pi_2 \mid \pi_1 \circ \pi_2 \mid \pi^* \mid \varphi? \end{aligned}$$

We use the abbreviations $(\varphi_1 \wedge \varphi_2) = \neg(\neg\varphi_1 \vee \neg\varphi_2)$ and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$. Moreover, a set $\{a_1, \ldots, a_n\} \subseteq \Sigma$ of atomic programs is identified with the program

$a_1 \cup \cdots \cup a_n$. The *semantic* of PDL$^\cap$ is defined over Kripke structures. Given a Kripke structure $\mathcal{K} = (S, \{\rightarrow_\sigma \mid \sigma \in \Sigma\}, \rho)$ over $(\mathbb{P}, \Sigma)$, we define via mutual induction for each PDL$^\cap$ program π a binary relation $[\![\pi]\!]_\mathcal{K} \subseteq S \times S$ and for each PDL$^\cap$ formula φ a subset $[\![\varphi]\!]_\mathcal{K} \subseteq S$ as follows, where $\sigma \in \Sigma$, $p \in \mathbb{P}$, and $\mathrm{op} \in \{\cup, \cap, \circ\}$:

$$\begin{aligned}
[\![\sigma]\!]_\mathcal{K} &= \rightarrow_\sigma & [\![p]\!]_\mathcal{K} &= \{s \mid p \in \rho(s)\} \\
[\![\varphi?]\!]_\mathcal{K} &= \{(s,s) \mid s \in [\![\varphi]\!]_\mathcal{K}\} & [\![\neg\varphi]\!]_\mathcal{K} &= S \setminus [\![\varphi]\!]_\mathcal{K} \\
[\![\pi^*]\!]_\mathcal{K} &= [\![\pi]\!]_\mathcal{K}^* & [\![\varphi_1 \vee \varphi_2]\!]_\mathcal{K} &= [\![\varphi_1]\!]_\mathcal{K} \cup [\![\varphi_2]\!]_\mathcal{K} \\
[\![\pi_1 \ \mathrm{op}\ \pi_2]\!]_\mathcal{K} &= [\![\pi_1]\!]_\mathcal{K} \ \mathrm{op}\ [\![\pi_2]\!]_\mathcal{K} & [\![\langle\pi\rangle\varphi]\!]_\mathcal{K} &= \{s \mid \exists t : (s,t) \in [\![\pi]\!]_\mathcal{K} \wedge t \in [\![\varphi]\!]_\mathcal{K}\}
\end{aligned}$$

Note that $[\![\langle\varphi?\rangle\psi]\!]_\mathcal{K} = [\![\varphi \wedge \psi]\!]_\mathcal{K}$. For $s \in S$ we write $(\mathcal{K}, s) \models \varphi$ if and only if $s \in [\![\varphi]\!]_\mathcal{K}$. If the Kripke structure $\mathcal{K}$ is clear from the context we write $[\![\varphi]\!]$ for $[\![\varphi]\!]_\mathcal{K}$. PDL is the fragment of PDL$^\cap$, where the intersection operator $\cap$ on programs is not allowed. *Test-free PDL* (resp. *test-free* PDL$^\cap$) is the fragment of PDL (resp. PDL$^\cap$), where the test-operator "?" is not allowed. The size $|\varphi|$ of a PDL$^\cap$ formula φ and the size $|\pi|$ of a PDL$^\cap$ program π is defined as follows: $|p| = |\sigma| = 1$ for all $p \in \mathbb{P}$ and $\sigma \in \Sigma$, $|\neg\varphi| = |\varphi?| = |\varphi|+1$, $|\varphi \vee \psi| = |\varphi|+|\psi|+1$, $|\langle\pi\rangle\varphi| = |\pi|+|\varphi|$, $|\pi_1 \ \mathrm{op}\ \pi_2| = |\pi_1|+|\pi_2|+1$ for $\mathrm{op} \in \{\cup, \cap, \circ\}$, and $|\pi^*| = |\pi|+1$. The fragment EF of CTL (where only the next and "exists finally" modality is allowed) can be defined as the fragment of test-free PDL, consisting of all formulas φ such that every for every subformula of the form $\langle\pi\rangle\psi$, either $\pi \in \Sigma$ or $\pi = \Sigma^*$.

A PDL program π (where the intersection operator is not allowed) can be viewed as a regular expression and translated into a finite automaton A, where transitions are labeled by symbols from Σ and test formulas $\varphi?$. The semantic $[\![A]\!]$ of this automaton is the union of all relations $[\![c_1]\!] \circ \cdots \circ [\![c_n]\!]$, where $c_1 \cdots c_n$ labels a path from the initial state of A to a final state; note that c_i can be of the form $\varphi?$. This PDL-variant is sometimes called APDL. For PDL$^\cap$ such a translation does not exist. Moreover, PDL$^\cap$ neither possesses the finite model property nor the tree model property in contrast to PDL [13].

Given a class $\mathcal{C}$ of Kripke structures and a logic $\mathcal{L}$ (e.g. PDL or PDL$^\cap$), the *model-checking problem* asks: Given a Kripke structure $\mathcal{K} \in \mathcal{C}$, a node s of $\mathcal{K}$, and a formula $\varphi \in \mathcal{L}$, does $(\mathcal{K}, s) \models \varphi$ hold. Following Vardi [26], we distinguish between three measures of complexity:

- *Data Complexity*: The complexity of verifying for a fixed formula $\varphi \in \mathcal{L}$, whether $(\mathcal{K}, s) \models \varphi$ for a given Kripke structure $\mathcal{K} \in \mathcal{C}$ and a node s of $\mathcal{K}$.
- *Expression Complexity*: The complexity of verifying for a fixed Kripke structure $\mathcal{K} \in \mathcal{C}$ and node s, whether $(\mathcal{K}, s) \models \varphi$ for a given formula $\varphi \in \mathcal{L}$.
- *Combined Complexity*: The complexity of verifying $(\mathcal{K}, s) \models \varphi$ for a given formula $\varphi \in \mathcal{L}$, a given Kripke structure $\mathcal{K} \in \mathcal{C}$, and a node s.

Convention. In the rest of this paper, we will consider PDL$^\cap$ without atomic propositions. A Kripke structure will be just a tuple $\mathcal{K} = (S, \{\rightarrow_\sigma \mid \sigma \in \Sigma\})$ where $\rightarrow_\sigma \subseteq S \times S$. Formally, we introduce the only atomic proposition true and define

$[\![\mathrm{true}]\!]_{\mathcal{K}} = S$. This is not a restriction, since a Kripke structure $(S, \{\to_\sigma \mid \sigma \in \Sigma\}, \rho)$ (where $\rho : S \to 2^{\mathbb{P}}$, $\Sigma \cap \mathbb{P} = \emptyset$) can be replaced by the new Kripke structure $(S, \{\to_\sigma \mid \sigma \in \Sigma \cup \mathbb{P}\})$ where $\to_p = \{(s,s) \mid p \in \rho(s)\}$ for all $p \in \mathbb{P}$. For the formalisms for specifying infinite Kripke structures that we will introduce in the next section, we will see that (a finite description of) this propositionless Kripke structure can be easily computed from (a finite description of) the original Kripke structure. Moreover, in $\mathrm{PDL}^\cap$ formulas, we have to replace every occurrence of an atomic proposition p by the formula $\langle p \rangle\mathrm{true}$.

4 Infinite State Systems

In this section, we consider several formalisms for describing infinite Kripke structures. Let Σ be be a set of atomic programs and Γ be a finite alphabet.

A *basic parallel process* (BPP) is a communication free Petri net, i.e., a Petri net, where every transition needs exactly one token for firing. By labeling transitions of a Petri net with labels from Σ, one can associate an infinite Kripke structure $\mathcal{K}(\mathcal{N})$ with a BPP $\mathcal{N}$, see [21] for more details.

A *basic process algebra* (BPA) over Σ is a tuple $\mathcal{X} = (\Gamma, \Delta)$ where $\Delta \subseteq \Gamma_\varepsilon \times \Sigma \times \Gamma^*$ is a finite *transition relation*. The BPA $\mathcal{X}$ describes the Kripke structure $\mathcal{K}(\mathcal{X}) = (\Gamma^*, \{\to_\sigma \mid \sigma \in \Sigma\})$ over Σ, where $\to_\sigma = \{(\gamma w, vw) \mid w \in \Gamma^*$ and $(\gamma, \sigma, v) \in \Delta\}$ for all $\sigma \in \Sigma$. The *size* $|\mathcal{X}|$ of $\mathcal{X}$ is $|\Gamma| + |\Sigma| + \sum_{(\gamma,\sigma,v)\in\Delta} |v|$. If $(\gamma, \sigma, v) \in \Delta$, we also write $\gamma \xrightarrow{\sigma}_{\mathcal{X}} v$.

Example 1. For a finite alphabet Γ we will use the BPA $\mathrm{Tree}_\Gamma = (\Gamma, \Delta)$ over $\Gamma \cup \overline{\Gamma}$ where $\Delta = \{(\varepsilon, a, a) \mid a \in \Gamma\} \cup \{(a, \overline{a}, \varepsilon) \mid a \in \Gamma\}$. Then $\mathcal{K}(\mathrm{Tree}_\Gamma)$ is the complete tree over Γ with backwards edges.

A *pushdown system* (PDS) over Σ is a tuple $\mathcal{Y} = (\Gamma, P, \Delta)$ where (i) P is a finite set of *control states*, and (ii) $\Delta \subseteq P \times \Gamma_\varepsilon \times \Sigma \times P \times \Gamma^*$ is a finite *transition relation*. The PDS $\mathcal{Y}$ describes the Kripke structure $\mathcal{K}(\mathcal{Y}) = (P\Gamma^*, \{\to_\sigma \mid \sigma \in \Sigma\})$ over Σ, where $\to_\sigma = \{(p\gamma w, qvw) \mid w \in \Gamma^*$ and $(p, \gamma, \sigma, q, v) \in \Delta\}$ for all $\sigma \in \Sigma$. The *size* $|\mathcal{Y}|$ of $\mathcal{Y}$ is $|\Gamma| + |P| + |\Sigma| + \sum_{(p,\gamma,\sigma,q,v)\in\Delta} |v|$. If $(p, \gamma, \sigma, q, v) \in \Delta$, we also write $p\gamma \xrightarrow{\sigma}_{\mathcal{Y}} qv$. Note that a BPA is just a stateless PDS.

Example 2. Let $\mathcal{K} = (S, \{\to_\sigma \mid \sigma \in \Sigma\})$ be a deterministic Kripke structure, i.e., for every state $s \in S$ and every $\sigma \in \Sigma$ there is at most one $t \in S$ with $s \to_\sigma t$. For PDL over BPA and PDS, determinism is no restriction: it can be ensured by choosing a possibly larger set Σ' of atomic programs such that every transition of the BPA (PDS) can be labeled with a unique $\sigma' \in \Sigma'$. Every original atomic program σ can be recovered as a union of some of these new atomic programs (for PRS, this doesn't work). We now want to express that the current state $s \in S$ is a recovery state of the system in the sense that wherever we go from s, we can always move back to s. This property cannot be expressed in the modal μ-calculus unless the state s is somehow uniquely marked, e.g., by a special atomic proposition (but here, we want to define the set of all recovery states). One can show that s is a recovery state if and only if

$$(\mathcal{K}, s) \models [\Sigma^*] \bigwedge_{\sigma \in \Sigma} \Big(\langle \sigma \rangle \text{true} \;\Rightarrow\; \langle \text{true?} \cap \sigma \circ \Sigma^* \rangle \text{true} \Big).$$

Note that true? defines the identity relation on S.

Example 3. Let us consider two PDS $\mathcal{Y}_i = (\Gamma, P_i, \Delta_i)$ (with a common pushdown alphabet Γ) over Σ_i ($i \in \{1,2\}$), where $\Sigma_1 \cap \Sigma_2 = \emptyset$, and such that $\mathcal{K}(\mathcal{Y}_1)$ and $\mathcal{K}(\mathcal{Y}_2)$ are deterministic (which, by the remarks from Example 2, is not a restriction). The systems $\mathcal{Y}_1$ and $\mathcal{Y}_2$ may synchronize over states from the intersection $P_1 \cap P_2$. These two systems can be modeled by the single PDS $\mathcal{Y} = (\Gamma, P_1 \cup P_2, \Delta_1 \cup \Delta_2)$ over $\Sigma_1 \cup \Sigma_2$. In this context, it might be interesting to express that whenever $\mathcal{Y}_1$ and $\mathcal{Y}_2$ can reach a common node s, and from s, $\mathcal{Y}_i$ can reach a node s_i by a local action, then the two systems can reach from s_1 and s_2 again a common node. This property can be expressed by the PDL$^\cap$ formula

$$[\Sigma_1^* \cap \Sigma_2^*] \bigwedge_{a \in \Sigma_1, b \in \Sigma_2} \Big(\langle a \rangle \text{true} \wedge \langle b \rangle \text{true} \;\Rightarrow\; \langle a \circ \Sigma_1^* \cap b \circ \Sigma_2^* \rangle \text{true} \Big).$$

Note that $[\![a \circ \Sigma_1^* \cap b \circ \Sigma_2^*]\!]$ is in general not the empty relation, although of course $a \circ \Sigma_1^* \cap b \circ \Sigma_2^* = \emptyset$ when interpreted as a regular expression with intersection.

A relation $U \subseteq \Gamma^* \times \Gamma^*$ is *prefix-recognizable* over Γ, if $U = \bigcup_{i=1}^n R_i$ ($n \geq 1$) and $R_i = \{(uw, vw) \mid u \in U_i, v \in V_i, w \in W_i\}$ for some regular languages $U_i, V_i, W_i \subseteq \Gamma^*$ ($1 \leq i \leq n$). We briefly write $R_i = (U_i \times V_i)W_i$. A *prefix-recognizable system* (PRS) (which should not be confused with Mayr's PRS (process rewrite systems) [21]) over Σ is a pair $\mathcal{Z} = (\Gamma, \alpha)$ where α assigns to every atomic program $\sigma \in \Sigma$ a prefix-recognizable relation $\alpha(\sigma)$ over Γ, which is given by finite automata $\mathcal{A}_1^\sigma, \mathcal{B}_1^\sigma, \mathcal{C}_1^\sigma, \ldots, \mathcal{A}_{n_\sigma}^\sigma, \mathcal{B}_{n_\sigma}^\sigma, \mathcal{C}_{n_\sigma}^\sigma$ such that $\alpha(\sigma) = \bigcup_{i=1}^{n_\sigma} (L(\mathcal{A}_i^\sigma) \times L(\mathcal{B}_i^\sigma)) L(\mathcal{C}_i^\sigma)$. The PRS $\mathcal{Z}$ describes the Kripke structure $\mathcal{K}(\mathcal{Z}) = (\Gamma^*, \{\alpha(\sigma) \mid \sigma \in \Sigma\})$ over Σ. The *size* $|\mathcal{Z}|$ of $\mathcal{Z}$ is $|\Gamma| + |\Sigma| + \sum_{\sigma \in \Sigma} \sum_{i=1}^{n_\sigma} |\mathcal{A}_i^\sigma| + |\mathcal{B}_i^\sigma| + |\mathcal{C}_i^\sigma|$, where $|A|$ is the number of states of a finite automaton A.

Our definition of BPA (resp. PDS) allows transitions of the form $\varepsilon \xrightarrow{\sigma}_{\mathcal{X}} v$ (resp. $p \xrightarrow{\sigma}_{\mathcal{Y}} qv$ for control states p and q). It is easy to see that our definition describes exactly the same class of BPA (resp. PDS) as defined in [20,21] (resp. [2,21,29,28]), and there are logspace translations between the two formalisms.

Usually, in the literature a PDS $\mathcal{Y}$ describes a Kripke structure with atomic propositions from some set $\mathbb{P}$. For this purpose, $\mathcal{Y}$ contains a mapping $\varrho : P \to 2^{\mathbb{P}}$, where P is the set of control states of $\mathcal{Y}$, and one associates with the atomic proposition $\eta \in \mathbb{P}$ the set of all configurations where the current control state p satisfies $\eta \in \varrho(p)$. In our formalism, which does not contain atomic propositions, we can simulate such an atomic propositions η by introducing the new transition rule $p \xrightarrow{\eta} p$ whenever $\eta \in \varrho(p)$, see also the convention from the end of Section 3. Similar remarks apply to BPA and PRS.

Table 1 summarizes our complexity results for PDL and its variants.

Table 1.

		BPA	**PDS**	**PRS**
EF	**data**	P-complete		EXP-complete
PDL$\setminus$?	**expression**		PSPACE-complete	
	combined			EXP-complete
	data	P-complete		
PDL	**expression**		EXP-complete	
	combined			
	data	PSPACE-hard, in EXP	EXP-complete	
PDL$^\cap$	**expression**		2EXP-complete	
PDL$^\cap\setminus$?	**combined**			

5 Model-Checking PDL over Infinite State Systems

It was shown in [8] that the model-checking problem of EF over (the Kripke structures defined by) Petri nets is undecidable. A reduction of this problem to the model-checking problem of test-free PDL over BPP shows:

Proposition 1. *The model-checking problem for test-free PDL over BPP is undecidable.*

Hence, in the following we will concentrate on the (sequential) system classes BPA, PDS, and PRS. Our results for (test-free) PDL without intersection over BPA/PDS/PRS mainly use results or adapt techniques from [2,22,28,29], see Table 1. It turns out that PDL without test behaves in exactly the same way as EF, and that adding the test operator leads in most cases to a complexity jump up to EXP-completeness.

In the rest of the paper, we concentrate on PDL$^\cap$, for which we prove that the expression and combined complexity over BPA/PDS/PRS is complete for 2EXP. Our lower bound proof uses a technique from [28] for describing a traversal of the computation tree of an alternating Turing machine in CTL using a pushdown. The main difficulty that remains is to formalize in PDL$^\cap$ that two configurations of an exponential space alternating Turing machine (these machines characterize 2EXP) are successor configurations. For doing this, we adjoin to every tape cell a binary counter, which represents the position of the tape cell. This encoding of configurations is also used in the recent 2EXP lower bound proof of Lange and Lutz for satisfiability of PDL$^\cap$ [17].

Theorem 1. *There exists a fixed BPA $\mathcal{X}$ such that the following problem is* 2EXP*-hard:*
INPUT: A test-free PDL$^\cap$-formula φ.
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$?

Proof. Since 2EXP equals the class of all languages that can be accepted by an ATM in exponential space [6], we can choose a fixed $2^{p(m)} - 1$ space bounded ATM $\mathcal{M} = (Q, \Sigma_{\mathcal{M}}, \Gamma_{\mathcal{M}}, q_0, \delta, \Box)$ (where $p(m)$ is a polynomial) with a 2EXP-complete acceptance problem. The machine $\mathcal{M}$ satisfies the conventions of Section 2. Let $w \in \Sigma_{\mathcal{M}}^*$ be an input of length n. We construct a fixed BPA $\mathcal{X} = \mathcal{X}(\mathcal{M}) = (\Gamma, \Delta)$ and a test-free PDL$^\cap$-formula $\varphi = \varphi(w, \mathcal{M})$ each over $\Sigma = \Sigma(\mathcal{M})$ such that $w \in L(\mathcal{M})$ if and only if $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$. Let $N = p(n)$ and $\Omega = Q \cup \Gamma_{\mathcal{M}}$. A configuration c of $\mathcal{M}$ is a word from the language $\bigcup_{0 \le i \le 2^N - 2} \Gamma_{\mathcal{M}}^i Q \Gamma_{\mathcal{M}}^{2^N-1-i}$. We will represent $c = \gamma_0 \cdots \gamma_{i-1} q \gamma_{i+1} \cdots \gamma_{2^N-1}$ by the word

$$\gamma_0[0] \cdots \gamma_{i-1}[i-1] q[i] \gamma_{i+1}[i+1] \cdots \gamma_{2^N-1}[2^N - 1], \tag{1}$$

where $[k]$ denotes the binary representation of k ($0 \le k \le 2^N - 1$) with N bits, i.e., $[k] = \beta_0 \cdots \beta_{N-1}$ with $\beta_j \in \{0,1\}$ and $k = \sum_{j=0}^{N-1} 2^j \cdot \beta_j$. A *cell* is a string $\omega[i]$, where $\omega \in \Omega$ and $0 \le i \le 2^N - 1$. Let Moves $= Q \times \Gamma_{\mathcal{M}} \times \{\leftarrow, \rightarrow\}$ be the set of moves of $\mathcal{M}$ and let

$$\begin{aligned}\text{Dir} = \{&L(\mu_1,\mu_2), R(\mu_1,\mu_2) \mid (\mu_1,\mu_2) \in \delta(Q_\forall, \Gamma)\} \cup \\ &\{E(\mu_1), E(\mu_2) \mid (\mu_1,\mu_2) \in \delta(Q_\exists, \Gamma)\}\end{aligned}$$

be the set of *direction markers.* These symbols separate consecutive configurations of the form (1) on the pushdown. As in [28], direction markers are used in order to organize a depth-first left-to-right traversal of the computation tree of the ATM $\mathcal{M}$ on the pushdown. Let $\Gamma = \Omega \cup \{0,1\} \cup \text{Dir}$ and $\Sigma = \Gamma \cup \overline{\Gamma} \cup \{\lambda\}$, which is a fixed alphabet. We define the fixed BPA $\mathcal{X}$ to be Tree$_\Gamma$ (see Example 1) together with the rule $(\varepsilon, \lambda, \varepsilon)$, which generates a λ-labeled loop at every node. In order to define the PDL$^\cap$ formula φ, we need several auxiliary programs:

- $\overline{X} = \bigcup_{x \in X} \overline{x}$ for $X \subseteq \Gamma$: Pops a single symbol $x \in X$ from the pushdown.
- $\text{pop}_i = \overline{\{0,1\}}^i$ for all $0 \le i \le N$: Pops i bits from the pushdown.
- $\overline{\text{cell}} = \overline{\Omega} \circ \text{pop}_N$: Pops a cell $\omega[i]$ from the pushdown.
- $\overline{\text{cell}_0} = \overline{\Omega} \circ \overline{0}^N$: Pops a cell $\omega[0]$ from the pushdown.
- $\overline{\text{cell}_1} = \overline{\Gamma_{\mathcal{M}}} \circ \overline{1}^N$: Pops a cell $\gamma[2^N - 1]$ for $\gamma \in \Gamma_{\mathcal{M}}$ from the pushdown.

Next, we define a program $\overline{\text{inc}}$, which is executable only if on top of the pushdown there is a word of the form $\omega[i]\omega'[i+1]$ for some $\omega, \omega' \in \Omega$ and some $0 \le i < 2^N - 1$. The program $\overline{\text{inc}}$ pops $\omega[i]$ during it execution. In order to define $\overline{\text{inc}}$ we will use the programs $\chi_{j,\beta}$ ($0 \le j < N$, $\beta \in \{0,1\}$) which assure that, after popping j bits of the current cell, a bit β can be popped that matches the bit that can be popped after popping another j bits of the subsequent cell. Afterwards, further bits may be popped:

$$\chi_{j,\beta} = \mathrm{pop}_j \circ \overline{\beta} \circ \overline{\{0,1\}}^* \circ \overline{\Omega} \circ \mathrm{pop}_j \circ \overline{\beta} \circ \overline{\{0,1\}}^*$$

$$\begin{aligned}\overline{\mathrm{inc}} = \overline{\mathrm{cell}} \cap \Bigg[(\overline{\mathrm{cell}} \circ \overline{\mathrm{cell}}) \cap \overline{\Omega} \circ \bigcup_{i=0}^{N-1} \Big(\overline{1}^i \circ \overline{0} \circ \overline{\{0,1\}}^* \circ \overline{\mathrm{cell}} \; \cap \\ \overline{\{0,1\}}^* \circ \overline{\Omega} \circ \overline{0}^i \circ \overline{1} \circ \overline{\{0,1\}}^* \; \cap \bigcap_{j=i+1}^{N-1} (\chi_{j,0} \cup \chi_{j,1}) \Big) \Bigg] \circ \Gamma^*\end{aligned}$$

The next program $\overline{\mathrm{conf}}$ is only executable if the top of the pushdown is a legal configuration in the sense of (1), i.e.: A word of the form $\omega_0[0]\omega_1[1] \cdots \omega_{2^N-1}[2^N-1]$ is assumed to be on top of the pushdown, for exactly one $0 \leq i \leq 2^N - 2$ we have $\omega_i \in Q$, and for all other i we have $\omega_i \in \Gamma_{\mathcal{M}}$. This top configuration is being popped during execution:

$$\overline{\mathrm{conf}} = (\overline{\mathrm{cell}_0} \circ \overline{\mathrm{cell}}^*) \;\cap\; (\overline{\mathrm{inc}}^* \circ \overline{\mathrm{cell}_1}) \cap \;\; (\overline{\Gamma_{\mathcal{M}} \cup \{0,1\}}^* \circ \overline{Q} \circ \overline{\Gamma_{\mathcal{M}} \cup \{0,1\}}^*)$$

For all $\omega, \omega' \in \Omega$ the program $\pi_{\omega,\omega'}$ is only executable if the top of the pushdown is a certain suffix of a configuration of $\mathcal{M}$ followed by a direction marker $d \in \mathrm{Dir}$ and a complete configuration of $\mathcal{M}$. More precisely,

$$\omega_k[k] \cdots \omega_{2^N-1}[2^N-1] \, d \, \omega'_0[0] \cdots \omega'_{2^N-1}[2^N-1]$$

with $\omega_k = \omega$ and $\omega'_k = \omega'$ must be on top of the pushdown. During its execution, $\pi_{\omega,\omega'}$ pops $\omega_k[k]$ from the pushdown:

$$\begin{aligned}\pi_{\omega,\omega'} = \overline{\mathrm{cell}} \;\; \cap \;\; \Bigg(\bigcap_{i=0}^{N-1} \bigcup_{\beta \in \{0,1\}} \overline{\omega} \circ \mathrm{pop}_i \circ \overline{\beta} \circ \overline{\{0,1\}}^* \circ \\ \overline{\mathrm{cell}}^* \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{cell}}^* \circ \overline{\omega'} \circ \mathrm{pop}_i \circ \overline{\beta} \circ \overline{\{0,1\}}^* \Bigg) \circ \Gamma^*\end{aligned}$$

The program $\pi_= = \bigcup_{\omega \in \Omega} \pi_{\omega,\omega}$ checks whether the content of the top cell $\omega[k]$ equals the content of the k-th cell of the subsequent configuration. Now we define a program check_μ for $\mu \in \mathrm{Moves}$, which is only executable if cdc' is on top of the pushdown, where: (i) c and c' are configurations of $\mathcal{M}$ in the sense of (1), (ii) $d \in \mathrm{Dir}$, and (iii) $\mathcal{M}$ moves from configuration c' to configuration c by the move μ. We restrict ourselves to the case where $\mu = (q, a, \leftarrow)$:

$$\lambda \cap \Big(\overline{\mathrm{conf}} \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \;\cap\; \pi_=^* \circ \bigcup_{p \in Q, b, c \in \Gamma_{\mathcal{M}}} (\pi_{q,c} \circ \pi_{c,p} \circ \pi_{a,b}) \circ \pi_=^* \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \Big) \circ \Gamma^*$$

The rest of the proof is analogous to Walukiewicz's proof for the EXP lower bound for CTL over PDS [28]. Using the direction markers, we can define a program traverse, whose execution simulates on the pushdown a single step in a depth-first left-to-right traversal of the computation tree of $\mathcal{M}$. Using a program init, which pushes the initial configuration on the pushdown, we get $(\varepsilon, \varepsilon) \in [\![\mathrm{init} \circ \mathsf{traverse}^*]\!]$ if and only if the initial configuration $q_0 w$ is accepting. □

In the rest of this paper, we sketch a 2EXP upper bound for the combined complexity of $\mathrm{PDL}^{\cap}$ over PRS. For this, we need the concept of two-way alternating tree automata. Vardi and Kupferman [15] reduced the model-checking problem of the modal μ-calculus over PRS to the emptiness problem for two-way alternating tree automata, and thereby deduced an EXP upper bound for the former problem, see also [4,29]. Our general strategy for model-checking $\mathrm{PDL}^{\cap}$ over PRS is the same.

Let Γ be a finite alphabet. A *Γ-tree* is a suffix-closed subset $T \subseteq \Gamma^*$, i.e., if $aw \in T$ for $w \in \Gamma^*$ and $a \in \Gamma$, then $w \in T$. Elements of T are called *nodes*. An *infinite path* in the tree T is an infinite sequence $u_1, u_2, \ldots$ of nodes such that $u_1 = \varepsilon$ and for all $i \geq 1$, $u_{i+1} = a_i u_i$ for some $a_i \in \Gamma$. A *Σ-labeled Γ-tree*, where Σ is a finite alphabet, is a pair (T, λ), where T is a Γ-tree and $\lambda : T \to \Sigma$ is a labeling function. The *complete Γ-tree* is the $\Gamma \uplus \{\bot\}$-labeled Γ-tree $(\Gamma^*, \lambda_\Gamma)$ where $\lambda_\Gamma(\varepsilon) = \bot$ and $\lambda_\Gamma(aw) = a$ for $a \in \Gamma$ and $w \in \Gamma^*$. For a finite set X, let $\mathcal{B}^+(X)$ be the set of all *positive boolean formulas over X*; note that true and false are positive boolean formulas. A subset $Y \subseteq X$ *satisfies* $\theta \in \mathcal{B}^+(X)$, if θ becomes true when assigning true to all elements in Y. Let $\mathrm{ext}(\Gamma) = \Gamma \uplus \{\varepsilon, \downarrow\}$ and define for all $u \in \Gamma^*, a \in \Gamma$: $\varepsilon u = u$, $\downarrow au = u$, whereas $\downarrow\varepsilon$ is undefined.

A *two-way alternating tree automaton* (TWATA) over Γ is a triple $\mathcal{T} = (S, \delta, \mathrm{Acc})$, where S is a finite *set of states*, $\delta : S \times (\Gamma \cup \{\bot\}) \to \mathcal{B}^+(S \times \mathrm{ext}(\Gamma))$ is the *transition function*, and $\mathrm{Acc} : S \to \{0, \ldots, m\}$ is the *priority function*, where $m \in \mathbb{N}$. Let $u \in \Gamma^*$ and $s \in S$. An *(s,u)-run* of $\mathcal{T}$ (over the complete Γ-tree $(\Gamma^*, \lambda_\Gamma)$) is a $(S \times \Gamma^*)$-labeled Ω-tree $R = (T_R, \lambda_R)$ for some finite set Ω such that: (i) $\varepsilon \in T_R$, (ii) $\lambda_R(\varepsilon) = (s, u)$, and (iii) if $\alpha \in T_R$ with $\lambda_R(\alpha) = (s', v)$ and $\delta(s', \lambda_\Gamma(v)) = \theta$, then there is a subset $Y \subseteq S \times \mathrm{ext}(\Gamma)$ that satisfies the formula θ and for all $(s'', e) \in Y$ there exists $\omega \in \Omega$ with $\omega\alpha \in T_R$ and $\lambda_R(\omega\alpha) = (s'', ev)$. An (s,u)-run $R = (T_R, \lambda_R)$ of $\mathcal{T}$ is *successful* if for every infinite path $w_1, w_2, \ldots$ of T_R, $\min(\{\mathrm{Acc}(s') \mid \lambda_R(w_i) \in \{s'\} \times \Gamma^* \text{ for infinitely many } i\})$ is even. Let $[\![\mathcal{T}, s]\!] = \{u \in \Gamma^* \mid \text{there is a successful } (s,u)\text{-run of } \mathcal{T}\}$. The *size* $|\mathcal{T}|$ of $\mathcal{T}$ is $|\Gamma| + |S| + \sum_{\theta \in \mathrm{ran}(\delta)} |\theta| + |\mathrm{Acc}|$, where $|\mathrm{Acc}| := \max\{\mathrm{Acc}(s) \mid s \in S\}$. For TWATAs $\mathcal{T}_i = (S_i, \delta_i, \mathrm{Acc}_i)$ $(i \in \{1,2\})$ over Γ let $\mathcal{T}_1 \uplus \mathcal{T}_2 = (S_1 \uplus S_2, \delta_1 \uplus \delta_2, \mathrm{Acc}_1 \uplus \mathrm{Acc}_2)$ be their *disjoint union*. Note that in our definition a TWATA over an alphabet Γ only runs on the complete Γ-tree. Hence, our definition is a special case of the definition in [14,15,27], where also runs of TWATAs on arbitrarily labeled trees are considered. Using [27], we obtain:

Theorem 2 ([27]). *For a given TWATA $\mathcal{T} = (S, \delta, \mathrm{Acc})$ and a state $s \in S$, it can be checked in time exponential in $|S| \cdot |\mathrm{Acc}|$ whether $\varepsilon \in [\![\mathcal{T}, s]\!]$.*

It should be noted that the size of a positive boolean formula that appears in the transition function δ of a TWATA $\mathcal{T} = (Q, \delta, \mathrm{Acc})$ can be exponential in $|Q|$, but the size of δ only appears polynomially in the upper bound for emptiness (and not exponentially, which would lead to a 2EXP upper bound for emptiness).

Let $\mathcal{T} = (S, \delta, \mathrm{Acc})$ be a TWATA over Γ. A *nondeterministic finite automaton* (briefly NFA) A over $\mathcal{T}$ is a pair $(Q, \to_A)$ where Q is a finite state set and all transitions are of the form $p \xrightarrow{a}_A q$ for $p, q \in Q, a \in \Gamma \cup \overline{\Gamma}$ or $p \xrightarrow{\mathcal{T},s}_A q$ for

$p, q \in Q, s \in S$. The latter transitions are called *test-transitions*. Let $A^{\downarrow}$ (resp. $A^{\uparrow}$) be the NFA over $\mathcal{T}$ that results from A by removing all transitions with a label from Γ (resp. $\overline{\Gamma}$), i.e., we only keep test-transitions and transitions with a label from $\overline{\Gamma}$ (resp. Γ). Let $\Rightarrow_A \subseteq (\Gamma^* \times Q) \times (\Gamma^* \times Q)$ be the smallest relation with:

- $(u, p) \Rightarrow_A (au, q)$ whenever $u \in \Gamma^*$ and $p \xrightarrow{a}_A q$
- $(au, p) \Rightarrow_A (u, q)$ whenever $u \in \Gamma^*$ and $p \xrightarrow{\bar{a}}_A q$
- $(u, p) \Rightarrow_A (u, q)$ whenever $u \in [\![\mathcal{T}, s]\!]$ and $p \xrightarrow{\mathcal{T}, s}_A q$

Let $[\![A, p, q]\!] = \{(u, v) \in \Gamma^* \times \Gamma^* \mid (u, p) \Rightarrow_A^* (v, q)\}$ for $p, q \in Q$.

We will inductively transform a given PDL$^{\cap}$-formula (resp. PDL$^{\cap}$-program) into an equivalent TWATA (resp. NFA over a TWATA). In order to handle the intersection operator on programs, we first have to describe a general automata theoretic construction: Let $\mathcal{T} = (S, \delta, \mathrm{Acc})$ be a TWATA over Γ and let $A = (Q, \rightarrow_A)$ be an NFA over $\mathcal{T}$. Let $\mathrm{hop}_A \subseteq \Gamma^* \times Q \times Q$ be the smallest set such that:

- for all $u \in \Gamma^*$ and $q \in Q$ we have $(u, q, q) \in \mathrm{hop}_A$
- if $(au, p', q') \in \mathrm{hop}_A$, $p \xrightarrow{a}_A p'$, and $q' \xrightarrow{\bar{a}}_A q$, then $(u, p, q) \in \mathrm{hop}_A$
- if $(u, p, r), (u, r, q) \in \mathrm{hop}_A$, then $(u, p, q) \in \mathrm{hop}_A$
- if $u \in [\![\mathcal{T}, s]\!]$, $p \xrightarrow{\mathcal{T}, s}_A q$, then $(u, p, q) \in \mathrm{hop}_A$

Intuitively, $(u, p, q) \in \mathrm{hop}_A$ if and only if we can walk from node u of the complete Γ-tree back to u along a path consisting of nodes from $\Gamma^* u$. At the beginning of this walk, the automaton A is initialized in state p, each time we move in the tree from v to av (resp. av to u) we read a (resp. $\bar{a}$) in A, and A ends in state q. Formally, we have:

Lemma 1. *We have $(u,p,q) \in \mathrm{hop}_A$ if and only if there exist $n \geq 1$, $u_1, \ldots, u_n \in \Gamma^* u$, and $q_1, \ldots, q_n \in Q$ such that $u_1 = u_n = u$, $q_1 = p$, $q_n = q$, and $(u_1, q_1) \Rightarrow_A (u_2, q_2) \cdots \Rightarrow_A (u_n, q_n)$.*

The inductive definition of the set hop_A can be translated into a TWATA:

Lemma 2. *There exists a TWATA $\mathcal{U} = (S', \delta', \mathrm{Acc}')$ with state set $S' = S \uplus (Q \times Q)$ such that (i) $[\![\mathcal{U}, s]\!] = [\![\mathcal{T}, s]\!]$ for $s \in S$, (ii) $[\![\mathcal{U}, (p, q)]\!] = \{u \in \Gamma^* \mid (u, p, q) \in \mathrm{hop}_A\}$ for $(p, q) \in Q \times Q$, and (iii) $|\mathrm{Acc}'| = |\mathrm{Acc}|$.*

Define a new NFA $B = (Q, \rightarrow_B)$ over the TWATA $\mathcal{U}$ by adding to A for every pair $(p, q) \in Q \times Q$ the test-transition $p \xrightarrow{\mathcal{U}, (p,q)} q$.

Lemma 3. *Let $u, v \in \Gamma^*$ and $p, q \in Q$. Then the following statements are equivalent:*

- $(u, v) \in [\![A, p, q]\!]$
- $(u, v) \in [\![B, p, q]\!]$

- *there exist a common suffix w of u and v and a state $r \in Q$ with $(u,w) \in [\![B^{\downarrow}, p, r]\!]$ and $(w,v) \in [\![B^{\uparrow}, r, q]\!]$*

Let $\mathrm{drop}_B \subseteq \Gamma^* \times Q \times Q$ be the smallest set such that:

- for all $u \in \Gamma^*$ and $p \in Q$ we have $(u,p,p) \in \mathrm{drop}_B$
- if $(u,p',q') \in \mathrm{drop}_B$, $p \xrightarrow{\bar{a}}_B p'$, and $q' \xrightarrow{a}_B q$, then $(au,p,q) \in \mathrm{drop}_B$
- if $s' \in S'$, $u \in [\![\mathcal{U}, s']\!]$, $p \xrightarrow{\mathcal{U},s'}_B r$, and $(u,r,q) \in \mathrm{drop}_B$, then $(u,p,q) \in \mathrm{drop}_B$
- if $s' \in S'$, $u \in [\![\mathcal{U}, s']\!]$, $r \xrightarrow{\mathcal{U},s'}_B q$, and $(u,p,r) \in \mathrm{drop}_B$, then $(u,p,q) \in \mathrm{drop}_B$

Lemma 4. *We have $(u,p,q) \in \mathrm{drop}_B$ if and only if there exist $r \in Q$ and a suffix v of u such that $(u,v) \in [\![B^{\downarrow}, p, r]\!]$ and $(v,u) \in [\![B^{\uparrow}, r, q]\!]$.*

Again, the inductive definition of drop_B can be translated into a TWATA:

Lemma 5. *There exists a TWATA $\mathcal{V} = (S'', \delta'', \mathrm{Acc}'')$ with state set $S'' = S' \uplus (Q \times Q)$ such that: (i) $[\![\mathcal{V}, s']\!] = [\![\mathcal{U}, s']\!]$ for every state $s' \in S'$ of $\mathcal{U}$, (ii) $[\![\mathcal{V}, (p,q)]\!] = \{u \in \Gamma^* \mid (u,p,q) \in \mathrm{drop}_B\}$ for every state $(p,q) \in Q \times Q$, and (iii) $|\mathrm{Acc}''| = |\mathrm{Acc}|$.*

Let $C = (Q, \rightarrow_C)$ be the NFA over the TWATA $\mathcal{V}$ that results from B by adding for every pair $(p,q) \in Q \times Q$ the test-transition $p \xrightarrow{\mathcal{V},(p,q)} q$. For $u,v \in \Gamma^*$ let $\inf(u,v)$ the longest common suffix of u and v.

Lemma 6. *Let $u,v \in \Gamma^*$ and $p,q \in Q$. Then the following statements are equivalent:*

- *$(u,v) \in [\![A, p, q]\!]$*
- *$(u,v) \in [\![C, p, q]\!]$*
- *there exists $r \in Q$ with $(u, \inf(u,v)) \in [\![C^{\downarrow}, p, r]\!]$ and $(\inf(u,v), v) \in [\![C^{\uparrow}, r, q]\!]$*

Now we are ready to prove the announced 2EXP upper bound for the combined complexity of PDL$^{\cap}$ over PRS. Let $\mathcal{Z} = (\Gamma, \alpha)$ be a PRS and φ be a PDL$^{\cap}$ formula each over Σ. We translate $\mathcal{Z}$ and φ into a TWATA $\mathcal{T} = (S, \delta, \mathrm{Acc})$ over Γ together with a state $s \in S$ such that $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$ if and only if $\varepsilon \in [\![\mathcal{T}, s]\!]$. The number of states of $\mathcal{T}$ will be exponentially in the size of the formula φ and polynomially in the size of $\mathcal{Z}$ and the size of the priority function Acc will be linear in the size of φ, which proves a 2EXP upper bound by Theorem 2. From now on any occurring TWATA is implicitly over Γ and the size of the priority function is at least 1. The construction of $\mathcal{T}$ is done inductively over the structure of the formula φ. More precisely, (i) for every subformula ψ of φ we construct a TWATA $\mathcal{T}(\psi)$ together with a state s of $\mathcal{T}(\psi)$ such that $[\![\psi]\!] = [\![\mathcal{T}(\psi), s]\!]$ and (ii) for every program π that occurs in φ we construct an NFA $A(\pi)$ over a TWATA $\mathcal{T}(\pi)$ such that $[\![\pi]\!] = [\![A(\pi), p, q]\!]$ for states p and q of $A(\pi)$.

The case $\psi = \mathrm{true}$ is clear, the case $\psi = \psi_1 \wedge \psi_2$ can be skipped since $\psi_1 \wedge \psi_2 \Leftrightarrow \langle \psi_1 ? \rangle \psi_2$. If $\psi = \neg\theta$, then we apply the standard complementation procedure [23], where all positive boolean formulas in the right-hand side of the transition function are dualized and the acceptance condition is complemented by incrementing the priority of every state. If $\psi = \langle\pi\rangle\theta$, then we have already constructed $A(\pi)$, $\mathcal{T}(\pi)$, and $\mathcal{T}(\theta)$ such that $[\![\pi]\!] = [\![A(\pi), p, q]\!]$ for two states p and q of $A(\pi)$ and $[\![\theta]\!] = [\![\mathcal{T}(\theta), s]\!]$ for a state s of $\mathcal{T}(\theta)$. Basically, the TWATA $\mathcal{T}(\psi)$ results from the disjoint union of $A(\pi)$ and $\mathcal{T}(\pi) \uplus \mathcal{T}(\theta)$, additionally $\mathcal{T}(\psi)$ can move from state q to state s. It remains to construct $A(\pi)$ and $\mathcal{T}(\pi)$ for a PDL$^\cap$ subprogram π of φ.

Case $\pi = \psi?$: We can assume that there exists a TWATA $\mathcal{T}(\psi)$ and a state r of $\mathcal{T}(\psi)$ such that $[\![\psi]\!] = [\![\mathcal{T}(\psi), r]\!]$. The TWATA $\mathcal{T}(\pi)$ is $\mathcal{T}(\psi)$. The automaton $A(\pi)$ has two states p and q with the only transition $p \xrightarrow{\mathcal{T},r} q$.

Case $\pi = \sigma \in \Sigma$: Assume that $\alpha(\sigma) = \bigcup_{i=1}^{n}(L(A_i') \times L(B_i'))L(C_i')$. Define the homomorphism $h : \Gamma^* \to \overline{\Gamma}^*$ by $h(a) = \overline{a}$ for all $a \in \Gamma$. From the representation of $\alpha(\sigma)$ we can construct finite automata A_i, B_i, C_i such that $L(A_i) = h(L(A_i'))$, $L(B_i) = L(B_i')^{\mathrm{rev}}$, and $L(C_i) = h(L(C_i'))$. Basically, the automaton $A(\pi)$ first chooses nondeterministically an $i \in \{1, \ldots, n\}$ and then simulates the automaton A_i, until a current tree node $u \in \Gamma^*$ belongs to the language $L(C_i)$. Then it continues by simulating the automaton B_i. Whether the current tree node $u \in \Gamma^*$ belongs to $L(C_i)$ has to be checked by the TWATA $\mathcal{T}(\pi)$, which can be built from the automaton C_i.

Case $\pi = \pi_1 \cup \pi_2$, $\pi = \pi_1 \circ \pi_2$, or $\pi = \chi^*$: We construct $A(\pi)$ by using the standard automata constructions for union, concatenation, and Kleene-star. We set $\mathcal{T}(\pi_1 \cup \pi_2) = \mathcal{T}(\pi_1 \circ \pi_2) = \mathcal{T}(\pi_1) \uplus \mathcal{T}(\pi_2)$ and $\mathcal{T}(\chi^*) = \mathcal{T}(\chi)$.

It remains to construct $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$. For this, we use the hop/drop-construction described above: Assume that the NFA $A(\pi_i) = (Q_i, \to_i)$ $(i \in \{1, 2\})$ over the TWATA $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$ is already constructed. Thus, $[\![A(\pi_i), p_i, q_i]\!] = [\![\pi_i]\!]$ for some states $p_i, q_i \in Q_i$. We first construct the NFA $C(\pi_i)$ over the TWATA $\mathcal{V}(\pi_i) = (S_i'', \delta_i'', \mathrm{Acc}_i'')$ as described in Lemma 6. Note that the state set of $C(\pi_i)$ is Q_i (the state set of $A(\pi_i)$) and that $|S_i''| = |S_i| + 2 \cdot |Q_i|^2$. Let $\mathcal{T}(\pi_1 \cap \pi_2) = \mathcal{V}(\pi_1) \uplus \mathcal{V}(\pi_2)$. The NFA $A(\pi_1 \cap \pi_2)$ is the product automaton of $C(\pi_1)$ and $C(\pi_2)$, where test-transitions can be done asynchronously: Let $A(\pi_1 \cap \pi_2) = (Q_1 \times Q_2, \to)$, where for $a \in \Gamma \cup \overline{\Gamma}$ we have $(r_1, r_2) \xrightarrow{a} (r_1', r_2')$ if and only if $r_i \xrightarrow{a}_{C(\pi_i)} r_i'$ for $i \in \{1, 2\}$. Finally, for a state s of $\mathcal{V}(\pi_1) \uplus \mathcal{V}(\pi_1)$ we have the test-transition $(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2), s} (r_1', r_2')$ if and only if for some $i \in \{1, 2\}$: s is a state of $\mathcal{V}(\pi_i)$, $r_i \xrightarrow{\mathcal{V}(\pi_i), s}_{C(\pi_i)} r_i'$, and $r_{3-i} = r_{3-i}'$.

Lemma 7. *We have* $[\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$. *Moreover, if* $A(\pi_i) = (Q_i, \to_i)$, $A(\pi_1 \cap \pi_2) = (Q, \to)$, $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$, *and* $\mathcal{T}(\pi_1 \cap \pi_2) = (S, \delta, \mathrm{Acc})$, *then:* $|Q| = |Q_1| \cdot |Q_2|$, $|S| = |S_1| + |S_2| + 2 \cdot |Q_1|^2 + 2 \cdot |Q_2|^2$, *and* $|\mathrm{Acc}| = \max\{|\mathrm{Acc}_1|, |\mathrm{Acc}_2|\}$.

Recall that we want to check $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$ for the PRS $\mathcal{Z}$ and the PDL$^\cap$ formula φ. A careful analysis of the constructions above allows to prove inductively:

Lemma 8. *If* $|\mathcal{Z}|$ *and* $|\varphi|$ *are sufficiently large, then:*

– *For every subformula ψ of φ with $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$ we have $|S| \leq |\mathcal{Z}|^{2\cdot|\psi|^2}$ and $|\mathrm{Acc}| \leq |\psi|$.*
– *For every subprogram π of φ with $A(\pi) = (Q, \rightarrow)$ and $\mathcal{T}(\pi) = (S, \delta, \mathrm{Acc})$ we have $|Q| \leq |\mathcal{Z}|^{|\pi|}$, $|S| \leq |\mathcal{Z}|^{2\cdot|\pi|^2}$, and $|\mathrm{Acc}| \leq |\pi|$.*

From our construction, Lemma 8, and Theorem 2 we get:

Theorem 3. *It can be checked in* 2EXP*, whether $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$ for a given PRS $\mathcal{Z}$ and a given PDL$^{\cap}$ formula φ. For a fixed PDL$^{\cap}$ formula, it can be checked in* EXP*, whether $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$ for a given PRS $\mathcal{Z}$.*

For the data complexity of test-free PDL$^{\cap}$ over PDA we can prove a matching EXP lower bound, by translating the fixed CTL formula from Walukiewicz's lower bound proof for the data complexity of CTL over PDS [28] into a fixed test-free PDL$^{\cap}$ formula. For the data complexity of test-free PDL$^{\cap}$ over BPA we can only prove a lower bound of PSPACE by a reduction from the universality problem from non-deterministic finite automata. Altogether, we obtain the results for PDL$^{\cap}$ in Table 1.

One might ask, whether an elementary upper bound also holds for the model-checking problem of PDL with the complement operator on programs over pushdown systems. But this model-checking problem allows to express emptiness for a given extended regular expression (i.e., regular expression where the complement operator is allowed), which is a well known nonelementary problem.

6 Open Problems

On the technical side it remains to close the gap between PSPACE and EXP for the data complexity of PDL$^{\cap}$ over BPA. Another fruitful research direction might be to extend PDL$^{\cap}$ by a unary fixpoint operator. The resulting logic is strictly more expressive than PDL$^{\cap}$ and the modal μ-calculus. We are confident that our upper bounds for PDL$^{\cap}$ from Theorem 3 can be extended to this logic.

References

1. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst*, 27(4):786–818, 2005.
2. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR'97*, LNCS 1243, pages 135–150. Springer, 1997.
3. A. Blumensath. Prefix-recognizable graphs and monadic second-order logic. Tech. Rep. 2001-06, RWTH Aachen, Germany, 2001.
4. T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. *ENTCS*, 68(6), 2002.
5. D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290(1):79–115, 2002.

6. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. Assoc. Comput. Mach.*, 28(1):114–133, 1981.
7. R. Danecki. Nondeterministic propositional dynamic logic with intersection is decidable. In *Proc. 5th Symp. Computation Theory*, LNCS 208, pages 34–53, 1984.
8. J. Esparza. On the decidabilty of model checking for several mu-calculi and petri nets. In *Proc. CAAP '94*, LNCS 787, pages 115–129. Springer, 1994.
9. J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
10. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.
11. S. Göller and M. Lohrey. Infinite State Model-Checking of Propositional Dynamic Logics Technical Report 2006/04, University of Stuttgart, Germany, 2006. ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2006-04/.
12. D. Harel. Recurring dominoes: making the highly undecidable highly understandable. *Ann. Discrete Math.*, 24:51–72, 1985.
13. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* Foundations of computing. The MIT Press, 2000.
14. O. Kupferman, N. Piterman, and M. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proc. CAV 2002*, LNCS 2404, pages 371–385, 2002.
15. O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proc. CAV 2000*, LNCS 1855, pages 36–52. Springer, 2000.
16. M. Lange. Model checking propositional dynamic logic with all extras. *J. Appl. Log.*, 4(1):39–49, 2005.
17. M. Lange and C. Lutz. 2-Exptime lower bounds for propositional dynamic logics with intersection. *J. Symb. Log.*, 70(4):1072–1086, 2005.
18. Ch. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *Proc. FOSSACS 2006*, LNCS 3921 pages 292–306. Springer, 2006.
19. D. Lugiez and P. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1–2):89–115, 2002.
20. R. Mayr. Strict lower bounds for model checking BPA. *ENTCS*, 18, 1998.
21. R. Mayr. Process rewrite systems. *Inf. Comput.*, 156(1):264–286, 2000.
22. R. Mayr. Decidability of model checking with the temporal logic EF. *Theor. Comput. Sci.*, 256(1-2):31–62, 2001.
23. D. Muller and P. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54(2-3):267–276, 1987.
24. C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.
25. W. Thomas. Some perspectives of infinite-state verification. In *Proc. ATVA 2005*, LNCS 3707, pages 3–10. Springer, 2005.
26. M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. STOC 1982*, pages 137–146. ACM Press, 1982.
27. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. ICALP '98*, LNCS 1443, pages 628–641. Springer, 1998.
28. I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proc. FSTTCS 2000*, LNCS 1974, pages 127–138. Springer, 2000.
29. I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.
30. S. Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products.* Dissertation, RWTH Aachen, Germany, 2005.