Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Infinite State Model-Checking of Propositional Dynamic Logics[1]

Stefan Göller and Markus Lohrey

Report Nr. 2006/04

Institut für Formale
Methoden der Informatik
Universitätsstraße 38
D–70569 Stuttgart

November 28, 2006

CR: F.1.3, F.4.1

[1]This work was supported by the German Research Foundation (DFG) project GELO

Abstract

Model-checking problems for propositional dynamic logic (PDL) and its extension $\mathrm{PDL}^{\cap}$ (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 another 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 operator or intersection of programs, see the monograph [13] for a detailed exposition. Recently, PDL, where programs are defined via visibly pushdown automata, was investigated [19].

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 [17] 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 [17]. 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. [27] for a survey. In [23] 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.

Let us first briefly summarize our results before arguing for the usefulness of model-checking PDL 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. This result follows easily 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 BPA (basic process algebras), 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) [2, 9, 16, 30, 31]. 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 in [4, 15]. The decidability of PDL for prefix-recognizable systems (and hence also BPA and PDS) follows easily from the fact that monadic second-order logic (MSO) is decidable for these systems and that PDL can be easily translated into MSO. But from the viewpoint of complexity, this approach to PDL model checking is quite unsatisfactory, since it leads to a nonelementary algorithm.

Our investigation of the complexity of model-checking problems follows Vardi's methodology from [28]. For a logic $\mathcal{L}$ and a class of (finitely presented infinite) systems $\mathcal{C}$, there are three different ways of measuring the complexity of the model-checking problem for $\mathcal{L}$ and $\mathcal{C}$: (i) One may fix a formula $\varphi \in \mathcal{L}$ and consider the complexity of verifying for a given (finite description of an infinite) system $S \in \mathcal{C}$ whether $S \models \varphi$; thus, only the system belongs to the input (data complexity or structure complexity). (ii) One may fix a system $S \in \mathcal{C}$ and consider the complexity of verifying for a given formula $\varphi \in \mathcal{L}$, whether $S \models \varphi$; thus, only the formula belongs to the input (expression complexity). (iii) Finally, both the system and the formula may belong to the input (combined complexity).

In Section 7 we investigate the test-free fragment of PDL over BPA, PDS, and PRS. For all these three kinds of systems, we show that the data/expression/combined complexity of test-free PDL is exactly the same as for EF, which is PSPACE in most cases. This changes when we allow the test operator. In Section 8 we study full PDL. Except for the data complexity of PDL over BPA (which is polynomial time), all complexities become EXP-complete. The results from Section 7 and 8 are mainly derived from known results about CTL and EF model-checking for pushdown systems [30].

In Section 9 we move to PDL with the intersection operator on programs, briefly PDL$^\cap$ [12]. This logic 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, PDL$^\cap$ is orthogonal to the modal μ-calculus with respect to expressiveness. A very difficult result of Danecki states that satisfiability of PDL$^\cap$ is in $2\mathsf{EXP}$ [7]. Only recently, a matching lower bound was obtained by Lange and Lutz [18]. Our main result from Section 9 states that the expression/combined complexity of PDL$^\cap$ (and also the test-free fragment of PDL$^\cap$) over BPA/PDS/PRS is $2\mathsf{EXP}$ complete, whereas the data complexity goes down to EXP. For the $2\mathsf{EXP}$ lower bound proof, we combine ideas from Walukiewicz's EXP lower bound proof for CTL over PDS [30] with the recent $2\mathsf{EXP}$ lower bound proof for satisfiability of PDL$^\cap$ [18]. For the upper bound, we transform a 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 [29], 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 problems is that two paths from a source node s and 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 careful analysis of such diverging paths in pushdown systems.

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 [20, 23, 32] 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 $a_1a_2\cdots a_n$ 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 PDL$^\cap$ with intersection of programs allows to formulate natural synchronization properties between several processes, which usually cannot be expressed in the modal μ-calculus, since they do not have the tree model property, see Example 4.2. One might argue that the high complexity (2EXP completeness) circumvents the application of 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 PDL$^\cap$ over pushdown systems is only EXP, which is the same as the data complexity of CTL [30]. Moreover, to obtain an exponential time algorithm for 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 4.2 (where it is 1). Table 1 gives an overview on our results.

2 Preliminaries

2.1 General notations

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 any finite word $w = w_1w_2\cdots w_n \in \Sigma^*$ $(w_1,\ldots,w_n \in \Sigma)$ let $w^{\mathrm{rev}} = w_nw_{n-1}\cdots w_1$ denote the *reversal* of w and for any language $L \subseteq \Sigma^*$ let $L^{\mathrm{rev}} = \{w^{\mathrm{rev}} \mid w \in L\}$ denote the *reversal* of L. For an arbitrary set X let $\mathrm{id}_X : X \to X$ denote the *identity function* on X, i.e. $\mathrm{id}_X(x) = x$ for all $x \in X$. Let A be a set and $R, U \subseteq A \times A$ be binary relations over 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 A, B, C, and D be sets, $A \cap B = \emptyset$ and let $f : A \to C$ and $g : B \to D$ be functions. Then the *disjoint union* $f \uplus g : A \cup B \to C \cup D$ of f and g is defined as $(f \uplus g)(a) = f(a)$ for all $a \in A$ and $(f \uplus g)(b) = g(b)$ for all $b \in B$. Finally, let $A^B = \{f \mid f : B \to A\}$ be the set of all functions from B to A.

2.2 Turing machines and complexity theory

We assume the reader is familiar with Turing machines and has some basic background in complexity theory [25]. More precisely, we assume the reader knows basic complexity classes such as L (deterministic logarithmic space), P (deterministic polynomial time), PSPACE (polynomial space), EXP (deterministic exponential time), and $2\mathsf{EXP}$ (deterministic double exponential time).

When proving lower bounds, we will often deal with alternating Turing machines. An *alternating Turing machine (ATM)* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ where (i) $Q = Q_{\text{acc}} \uplus Q_{\text{rej}} \uplus Q_{\exists} \uplus Q_{\forall}$ 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 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 $\delta : (Q_{\exists} \cup Q_{\forall}) \times \Gamma \to \text{Moves} \times \text{Moves}$ with $\text{Moves} = Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ assigns to every pair $(q, \gamma) \in (Q_{\exists} \cup Q_{\forall}) \times \Gamma$ a pair of *moves*. Assume a configuration c of $\mathcal{M}$ is in current state $q \in Q_{\exists} \cup Q_{\forall}$ and scans a symbol $\gamma \in \Gamma$. If $\delta(q, \gamma) = (\mu_1, \mu_2)$ and c_i the successor configuration of c by making the move μ_i ($i \in \{1, 2\}$) we also write $c \vdash_{\mathcal{M}}^{\mu_i} c_i$. Moreover, we call c_1 the *left successor configuration* and c_2 the *right successor configuration* of c. We call a configuration c of $\mathcal{M}$ in current state $q \in Q$ *accepting* if and only if one of the following three conditions holds:

- $q \in Q_{\text{acc}}$ or
- $q \in Q_{\exists}$ and there exists an accepting successor configuration of c or
- $q \in Q_{\forall}$ and all successor configurations of c are accepting.

We say an input $w \in \Sigma^*$ is *accepted* by an ATM $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ if and only if $q_0 w$ is an accepting configuration. By $L(\mathcal{M}) \subseteq \Sigma^*$ we denote the *language* accepted by $\mathcal{M}$. It is well known that alternating polynomial space equals EXP and alternating exponential space equals $2\mathsf{EXP}$ [6]. Whenever we talk about hardness we mean hardness w.r.t logarithmic space reductions.

2.3 Finite automata

A *nondeterministic finite automaton* (NFA for short) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ where the following five conditons are true: (i) Q is a finite *set of states*, (ii) Σ is a finite alphabet, (iii) $q_0 \in Q$ is the *initial state*, (iv) $F \subseteq Q$ is the set of *final states* and (v) $\delta : Q \times \Sigma_\varepsilon \to 2^Q$ is the *transition function*. We denote by $L(\mathcal{A})$ the accepted language of an NFA. The *size* $|\mathcal{A}|$ of an NFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ is defined as $|Q| + |\Sigma|$.

3 Logic

In this section $\mathbb{P}$ is always a finite set of *atomic propositions* and Σ is a always a finite set of *atomic programs*.

3.1 Kripke structures

A *Kripke structure* over $(\mathbb{P}, \Sigma)$ is a tuple $\mathcal{K} = (S, \{\rightarrow_\sigma \mid \sigma \in \Sigma\}, \rho)$ where (i) S is a (possibly infinite) set of *nodes*, (ii) $\rightarrow_\sigma \subseteq S \times S$ is a *transition relation* for all $\sigma \in \Sigma$ and (iii) $\rho : S \rightarrow 2^{\mathbb{P}}$ labels every node with a set of atomic propositions.

3.2 Propositional Dynamic Logic and extensions

PDL-*formulas* φ and PDL-*programs* π over $(\mathbb{P}, \Sigma)$ are defined by the following grammar, where $p \in \mathbb{P}$ and $\sigma \in \Sigma$:

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

We introduce the following abbreviations: $(\varphi_1 \wedge \varphi_2) = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$. We abbreviate the program $\bigcup_{\sigma\in\Sigma} \sigma$ by Σ. We abbreviate a program $\pi_1 \circ \pi_2 \circ \ldots \circ \pi_k$ by $\bigcirc_{i=1}^{k}\pi_i$ and the k-fold iteration of a program π by π^k. If a PDL program is of the form $\varphi?$ then we call φ a *test formula*. We call a PDL formula φ *test-free* if and only if φ does not contain any test subformulas. If a PDL formula φ is of the form then $\langle\pi\rangle\varphi'$ we call φ a *box formula* and π a *box program*. The *semantic* of PDL 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 program π a binary relation $[\![\pi]\!]_{\mathcal{K}} \subseteq S \times S$ and for each PDL formula φ a subset $[\![\varphi]\!]_{\mathcal{K}} \subseteq S$ as follows, where $\sigma \in \Sigma$ and $p \in \mathbb{P}$:

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

Hence, we can also write $\langle\varphi?\rangle\psi$ for $\varphi\wedge\psi$. 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 $[\![\varphi]\!]_{\mathcal{K}}$. The size $|\varphi|$ of a PDL-formula φ and the size $|\pi|$ of a PDL-program π is defined by mutual induction as follows: $|p| = |\sigma| = 1$ for all $p \in \mathbb{P}$ and $\sigma \in \Sigma$, $|\neg\varphi| = |\varphi| + 1$, $|\varphi \vee \psi| = |\varphi| + |\psi| + 1$, $|\langle\pi\rangle\varphi| = |\pi| + |\varphi|$, $|\pi_1 \cup \pi_2| = |\pi_1 \circ \pi_2| = |\pi_1| + |\pi_2| + 1$, $|\pi^*| = |\pi| + 1$, and $|\varphi?| = |\varphi| + 1$.

3.2.1 APDL

We can as well represent PDL-programs by NFAs. The set of all APDL formulas Φ and the set of all APDL automata Π over $(\mathbb{P}, \Sigma)$ are the smallest sets which fulfill the following properties: (i) $\mathbb{P} \subseteq \Phi$, (ii) if $\varphi, \psi \in \Phi$ then we have $\varphi \vee \psi, \neg\varphi \in \Phi$, (iii) if $\mathcal{A} \in \Pi$ and $\varphi \in \Phi$ then we have $\langle \mathcal{A} \rangle \varphi \in \Phi$, and (iv) if $\mathcal{A}$ is an NFA over a finite subset of (the infinite alphabet) $\Sigma \cup \{\varphi? \mid \varphi \in \Phi\}$ then we have $\mathcal{A} \in \Pi$. Again, the semantic of APDL 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 APDL automaton $\mathcal{A}$ a binary relation $[\![\mathcal{A}]\!]_\mathcal{K} \subseteq S \times S$ and for each APDL formula φ a subset $[\![\varphi]\!]_\mathcal{K} \subseteq S$ as follows, where $p \in \mathbb{P}$ and $\sigma \in \Sigma$:

$$\begin{aligned}
[\![\sigma]\!]_\mathcal{K} &= \rightarrow_\sigma \\
[\![\varphi?]\!]_\mathcal{K} &= \{(s,s) \mid s \in [\![\varphi]\!]_\mathcal{K}\} \\
[\![\mathcal{A}]\!]_\mathcal{K} &= \bigcup_{a_1 a_2 \cdots a_n \in L(\mathcal{A})} [\![a_1]\!]_\mathcal{K} \circ [\![a_2]\!]_\mathcal{K} \circ \cdots \circ [\![a_n]\!]_\mathcal{K} \\
[\![p]\!]_\mathcal{K} &= \{s \mid p \in \rho(s)\} \\
[\![\neg\varphi]\!]_\mathcal{K} &= S \setminus [\![\varphi]\!]_\mathcal{K} \\
[\![\varphi_1 \vee \varphi_2]\!]_\mathcal{K} &= [\![\varphi_1]\!]_\mathcal{K} \cup [\![\varphi_2]\!]_\mathcal{K} \\
[\![\langle \mathcal{A} \rangle \varphi]\!]_\mathcal{K} &= \{s \mid \exists t : (s,t) \in [\![\mathcal{A}]\!]_\mathcal{K} \wedge t \in [\![\varphi]\!]_\mathcal{K}\}
\end{aligned}$$

As above, for $s \in S$ we write $(\mathcal{K}, s) \models \varphi$ if and only if $s \in [\![\varphi]\!]_\mathcal{K}$. Note that in the definition of $[\![\mathcal{A}]\!]_\mathcal{K}$ the alphabet of $\mathcal{A}$ may contain test formulas $\varphi?$. Recall that $[\![\varphi?]\!]_\mathcal{K}$ is the restriction of id_S to all nodes in $[\![\varphi]\!]_\mathcal{K}$. The *size* $|\varphi|$ of an APDL formula φ is inductively defined as follows: $|p| = 1$ for all $p \in \mathbb{P}$, $|\neg\varphi| = |\varphi| + 1$, $|\varphi \vee \psi| = |\varphi| + |\psi| + 1$, and finally $|\langle \mathcal{A} \rangle \varphi| = |\mathcal{A}| + |\varphi|$ where $|\mathcal{A}|$ equals the size of the NFA $\mathcal{A}$ defined in Subsection 2.3.

Remark 3.1. *For every PDL program π over $\Pi = (\mathbb{P}, \Sigma)$ we can compute in logarithmic space an APDL automaton $\mathcal{A}_\pi$ over Π such that for all Kripke structures $\mathcal{K}$ over Π we have $[\![\pi]\!]_\mathcal{K} = [\![\mathcal{A}_\pi]\!]_\mathcal{K}$. If π is test-free then $\mathcal{A}_\pi$ is just an ordinary NFA over Σ. This translation is nothing but the standard translation from regular expressions to finite automata.*

3.2.2 The extension $\mathrm{PDL}^\cap$

The logic $\mathrm{PDL}^\cap$ is the extension of PDL by the intersection of programs and allows to ensure that two programs start and end in the same nodes of the Kripke structure respectively. Syntactically, $\mathrm{PDL}^\cap$ formulas φ and $\mathrm{PDL}^\cap$-programs π over $(\mathbb{P}, \Sigma)$ are given as follows, where $p \in \mathbb{P}$ and $\sigma \in \Sigma$:

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

Given a Kripke structure $\mathcal{K} = (S, \{\rightarrow_\sigma \mid \sigma \in \Sigma\}, \rho)$ over $(\mathbb{P}, \Sigma)$ the semantic of $\mathrm{PDL}^\cap$ is defined as for PDL with the addition of the intersection of programs:

$$[\![\pi_1 \cap \pi_2]\!]_\mathcal{K} = [\![\pi_1]\!]_\mathcal{K} \cap [\![\pi_2]\!]_\mathcal{K}$$

In contrast to PDL, PDL$^\cap$ cannot be translated into APDL. Moreover, PDL$^\cap$ neither possesses the finite model property nor the tree model property in contrast to PDL [13].

Conventions In the rest of this paper, we will consider PDL (and hence PDL$^\cap$) without atomic propositions. A Kripke structure will be just a tuple $\mathcal{K} = (S, \{\to_\sigma \mid \sigma \in \Sigma\})$ where $\to_\sigma \subseteq S \times S$. Formally, we introduce the only atomic proposition $\mathtt{true}$ and define $[\![\mathtt{true}]\!]_\mathcal{K} = S$. 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 $p \in \mathbb{P}$. By the formalisms for specifying infinite Kripke structures that we will introduce in Section 4, 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 PDL formulas, we have to replace every occurrence of an atomic proposition p by the formula $\langle p\rangle\mathtt{true}$. Hence, for the rest of this paper, we will talk about PDL formulas and PDL programs *only* over a set of atomic programs Σ defined by the following grammar, where $\sigma \in \Sigma$ and $[\![\mathtt{true}]\!]_\mathcal{K} = S$:

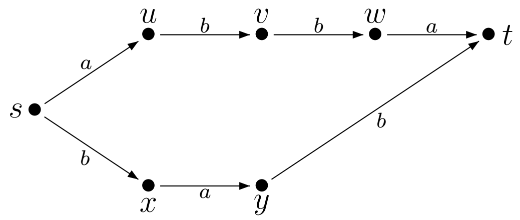
$$
\begin{aligned}
\varphi &::= \mathtt{true} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\pi\rangle\,\varphi \\
\pi &::= \sigma \mid \pi_1 \cup \pi_2 \mid \pi_1 \circ \pi_2 \mid \pi^* \mid \varphi?
\end{aligned}
$$

Note that $[\![\mathtt{true}? \cap \pi]\!]$ is the restriction of id_S to those nodes where $\langle\pi\rangle\mathtt{true}$ is true.

Example 3.2. *Let the program π over $\{a, b\}$ be defined as*

$$\pi = a \circ (a \cup b)^* \cap b \circ (a \cup b)^*$$

Note that $[\![\pi]\!]$ is in general not the empty relation, although of course $a \circ (a \cup b)^* \cap b \circ (a \cup b)^*$ represents the empty set when interpreted as a regular expression with intersection. Let the Kripke structure $\mathcal{K}$ be given as follows:



We have $(s, t) \in [\![\pi]\!]_\mathcal{K}$ and $s \in [\![\langle\pi\rangle\mathtt{true}]\!]_\mathcal{K}$.

Example 3.3. *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. 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).* Claim*: Let $s \in S$. The node s is a recovery state if and only if*

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

Proof. (only-if) Assume s is a recovery state. Let $\rightarrow = \bigcup_{\sigma \in \Sigma} \rightarrow_\sigma$. We have to show that for every state $t \in S$ such that $s \rightarrow^* t$ we have $(\mathcal{K}, t) \models \bigwedge_{\sigma \in \Sigma} \Big(\langle \sigma \rangle \mathtt{true} \;\Rightarrow\; \langle \mathtt{true}? \cap \sigma \circ \Sigma^* \rangle \mathtt{true} \Big)$. Assume $s \rightarrow^* t$ and $(\mathcal{K}, t) \models \langle \sigma \rangle \mathtt{true}$ for some $\sigma \in \Sigma$. This implies $t \xrightarrow{\sigma} u$ for some node $u \in S$. Since s is a recovery state and $s \rightarrow^* u$, we have $u \rightarrow^* s$. Altogether we have $t \xrightarrow{\sigma} u \rightarrow^* s \rightarrow^* t$, hence $(\mathcal{K}, t) \models \langle \mathtt{true}? \cap \sigma \circ \Sigma^* \rangle \mathtt{true}$.

(if) Assume $(\mathcal{K}, s) \models [\Sigma^*] \bigwedge_{\sigma \in \Sigma} \Big(\langle \sigma \rangle \mathtt{true} \;\Rightarrow\; \langle \mathtt{true}? \cap \sigma \circ \Sigma^* \rangle \mathtt{true} \Big)$. Let $t \in S$ be such that $s \rightarrow^* t$. By induction on the minimal $n \in \mathbb{N}$ such that $s \rightarrow^n t$ we prove $t \rightarrow^* s$. The case $n = 0$ is trivial. Now assume that $n > 0$, $\sigma \in \Sigma$ and $t' \in S$ such that $s \rightarrow^{n-1} t' \xrightarrow{\sigma} t$ is a path from s to t of minimal length n. Since $(\mathcal{K}, t') \models \langle \mathtt{true}? \cap \sigma \circ \Sigma^* \rangle \mathtt{true}$ and $\mathcal{K}$ is deterministic we have $t \rightarrow^* t'$. By induction hypothesis we have $t' \rightarrow^* s$, hence $t \rightarrow^* t' \rightarrow^* s$. $\square$

3.3 The logic CTL and its fragment EF

Let Σ be a set of atomic programs. Formulas φ over Σ of the logic CTL are defined by the following grammar, where $\sigma \in \Sigma$:

$$\begin{aligned} \varphi \;::=\; & \mathtt{true} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle \sigma \rangle \varphi \mid \\ & \exists(\varphi_1 \mathcal{U} \varphi_2) \mid \exists\neg(\varphi_1 \mathcal{U} \varphi_2) \end{aligned}$$

We introduce the abbreviations $\mathtt{false} = \neg\mathtt{true}$, $[\sigma]\varphi = \neg\langle\sigma\rangle\neg\varphi$ where $\sigma \in \Sigma$, and $\exists \mathrm{F} \varphi = \exists(\mathtt{true}\; \mathcal{U}\; \varphi)$. Given a Kripke structure $\mathcal{K} = (S, \{\rightarrow_\sigma \mid \sigma \in \Sigma\})$ over Σ let $\rightarrow = \bigcup_{\sigma \in \Sigma} \rightarrow_\sigma$ be the union of all relations $\rightarrow_\sigma$. A *maximal path* in $\mathcal{K}$ is a path in the relation $\rightarrow$ that is either infinite or ends in a node without any outgoing transitions. We define for every CTL formula φ a subset $[\![\varphi]\!]_\mathcal{K}$ inductively as follows, where $\sigma \in \Sigma$:

$$\begin{aligned} [\![\mathtt{true}]\!]_\mathcal{K} &= S \\ [\![\neg\varphi]\!]_\mathcal{K} &= S \setminus [\![\varphi]\!]_\mathcal{K} \\ [\![\varphi_1 \vee \varphi_2]\!]_\mathcal{K} &= [\![\varphi_1]\!]_\mathcal{K} \cup [\![\varphi_2]\!]_\mathcal{K} \\ [\![\langle\sigma\rangle\varphi]\!]_\mathcal{K} &= \{s \mid \exists t : s \rightarrow_\sigma t \text{ and } t \in [\![\varphi]\!]_\mathcal{K}\} \\ [\![\exists(\varphi_1 \mathcal{U} \varphi_2)]\!]_\mathcal{K} &= \{s \mid \text{there exists a finite path } s = u_0 \rightarrow u_1 \rightarrow \cdots \rightarrow u_n \; (n \geq 0) \\ &\qquad \text{such that } u_n \in [\![\varphi_2]\!]_\mathcal{K} \text{ and for all } i < n : u_i \in [\![\varphi_1]\!]_\mathcal{K}\} \\ [\![\exists\neg(\varphi_1 \mathcal{U} \varphi_2)]\!]_\mathcal{K} &= \{s \mid \text{there exists a maximal path } (u_i)_{i \geq 0} \text{ such that } s = u_0 \\ &\qquad \text{and for all } i \geq 0 : u_i \in [\![\varphi_2]\!]_\mathcal{K} \Rightarrow \exists j < i : u_j \notin [\![\varphi_1]\!]_\mathcal{K}\} \end{aligned}$$

As above, for $s \in S$ we write $(\mathcal{K}, s) \models \varphi$ if and only if $s \in [\![\varphi]\!]_{\mathcal{K}}$. The *size* $|\varphi|$ of a CTL formula φ is inductively defined as follows: $|\mathtt{true}| = 1$, $|\neg\varphi| = |\varphi| + 1$, $|\varphi_1 \vee \varphi_2| = |\varphi_1| + |\varphi_2| + 1$, $|\langle\sigma\rangle\varphi| = |\varphi| + 1$, and finally $|\exists(\varphi_1 \mathcal{U} \varphi_2)| = |\exists\neg(\varphi_1 \mathcal{U} \varphi_2)| = |\varphi_1| + |\varphi_2| + 1$.

Formulas of *modal logic* over the set of atomic programs Σ form a fragment of CTL and are defined by the following grammar, where $\sigma \in \Sigma$:

$$\varphi ::= \mathtt{true} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\sigma\rangle\varphi$$

The logic *EF* extends modal logic with the addition of $\exists \mathrm{F}\varphi$, i.e., EF formulas φ over Σ are defined by the following grammar:

$$\varphi ::= \mathtt{true} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\sigma\rangle\varphi \mid \exists \mathrm{F}\varphi$$

Remark 3.4. *The following problem can be computed in logarithmic space:*
INPUT: An EF formula φ over Σ.
OUTPUT: A test-free PDL formula ψ over Σ such that for all Kripke structures $\mathcal{K}$ over Σ we have $[\![\varphi]\!]_{\mathcal{K}} = [\![\psi]\!]_{\mathcal{K}}$.

Given a class $\mathcal{C}$ of Kripke structures and any logic $\mathcal{L}$ from above, 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 [28], we distinguish between three measures of complexity:

- *Data Complexity*: One fixes a formula $\varphi \in \mathcal{L}$ and considers the complexity of verifying for a Kripke structure $\mathcal{K} \in \mathcal{C}$ and a node s of $\mathcal{K}$ whether $(\mathcal{K}, s) \models \varphi$.
- *Expression Complexity*: One fixes a Kripke structure $\mathcal{K} \in \mathcal{C}$ and a node s of $\mathcal{K}$ and considers the complexity of verifying for a given formula $\varphi \in \mathcal{L}$ whether $(\mathcal{K}, s) \models \varphi$.
- *Combined Complexity*: A Kripke structure $\mathcal{K} \in \mathcal{C}$, a node s of $\mathcal{K}$, as well as a formula $\varphi \in \mathcal{L}$ belong to the input and one considers the complexity of verifying $(\mathcal{K}, s) \models \varphi$.

In [17] it was shown that over finite systems, the combined complexity of PDL and $\mathrm{PDL}^{\cap}$ (and further extensions) is complete for P.

4 Infinite state systems

In this section we introduce several classes of infinite state systems. In the following let Σ always be a set of atomic programs and Γ be a finite alphabet.

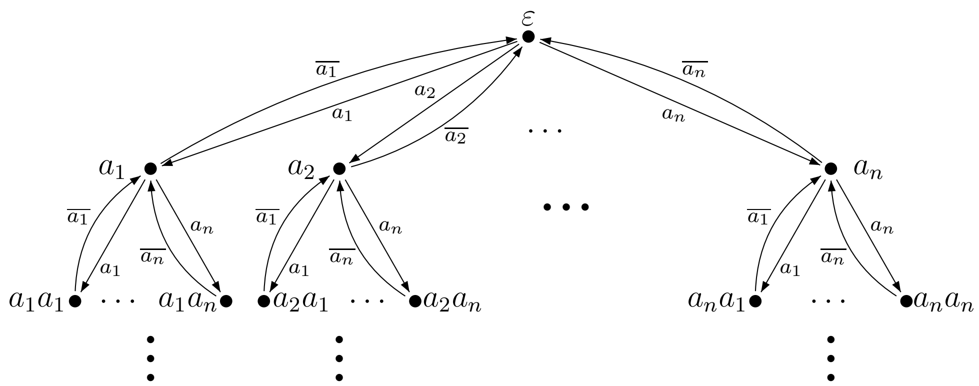
4.1 Basic process algebras

A *basic process algebra* (BPA for short) over Σ is a tuple $\mathcal{X} = (\Gamma, \Delta)$ where $\Delta \subseteq \Gamma_\varepsilon \times \Sigma \times \Gamma^*$ is a finite *transition relation.* A BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ describes the Kripke structure $\mathcal{K}(\mathcal{X}) = (\Gamma^*, \{\to_\sigma \mid \sigma \in \Sigma\})$ over Σ where for all $\sigma \in \Sigma$ we have:

$$\to_\sigma \quad = \quad \{(aw, vw) \mid \ w \in \Gamma^* \text{ and } (a, \sigma, v) \in \Delta\}$$

The *size* $|\mathcal{X}|$ is defined as $|\Gamma| + \sum_{(a,\sigma,v)\in\Delta} |v| + |\Sigma|$. If $(a, \sigma, v) \in \Delta$, we also write $a \xrightarrow{\sigma}_{\mathcal{X}} v$.

Example 4.1. *For a finite alphabet* $\Gamma = \{a_1, \ldots, a_n\}$ *we give the BPA* $\mathrm{Tree}_\Gamma = (\Gamma, \Delta)$ *over* $\Gamma \cup \overline{\Gamma}$ *where* $\Delta = \{(\varepsilon, a, a) \mid a \in \Gamma\} \cup \{(a, \overline{a}, \varepsilon) \mid a \in \Gamma\}$*. Hence,* $\mathcal{K}(\mathrm{Tree}_\Gamma)$ *looks as follows:*



The BPA Tree_Γ will occur several times in the rest of this paper.

4.2 Pushdown systems

A *pushdown system* (PDS for short) 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.* A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ describes the Kripke structure $\mathcal{K}(\mathcal{Y}) = (P\Gamma^*, \{\to_\sigma \mid \sigma \in \Sigma\})$ over Σ where for all $\sigma \in \Sigma$ we define:

$$\to_\sigma \quad = \quad \{(paw, qvw) \mid w \in \Gamma^* \text{ and } (p, a, \sigma, q, v) \in \Delta\}$$

The *size* $|\mathcal{Y}|$ is defined as $|\Gamma| + |P| + \sum_{(p,a,\sigma,q,v)\in\Delta} |v| + |\Sigma|$. If $(p, a, \sigma, q, v) \in \Delta$ we also write $pa \xrightarrow{\sigma}_{\mathcal{Y}} qv$.

Note that every BPA $\mathcal{X} = (\Gamma, \Delta_{\mathcal{X}})$ can be translated into an equivalent PDS $\mathcal{Y}(\mathcal{X}) = (\Gamma, \{p\}, \Delta)$ (each over Σ) with a single state p, where

$$\Delta = \{(p, a, \sigma, p, v) \mid (a, \sigma, v) \in \Delta_{\mathcal{X}}\}.$$

Note that a BPA is a PDS with a single control state. Moreover BPA are also known as *stateless pushdown systems.*

Example 4.2. *Let us consider two deterministic pushdown systems given by* $\mathcal{Y}_i = (\Gamma, P_i, \Delta_i)$ *over* Σ_i $(i \in \{1,2\})$ *with a common stack alphabet* Γ*, where* $\Sigma_1 \cap \Sigma_2 = \emptyset$*. Determinism means that every node of* $\mathcal{K}(\mathcal{Y}_i)$ *has at most one outgoing* a*-transition for every* $a \in \Sigma_i$*. The systems* $\mathcal{Y}_1$ *and* $\mathcal{Y}_2$ *may synchronize over states in the intersection* $P_1 \cap P_2$*. These two systems can be modeled by a single pushdown system* $\mathcal{Y} = (\Gamma, P_1 \cup P_2, \Delta_1 \cup \Delta_2)$ *over* $\Sigma = \Sigma_1 \cup \Sigma_2$*. In this context, it might be interesting to express that whenever* $\mathcal{Y}_1$ *and* $\mathcal{Y}_2$ *are in the same configuration* c *(a configuration corresponds to a node in* $\mathcal{K}(\mathcal{Y})$*), and* $\mathcal{Y}_i$ *can reach from* c *a configuration* c_i *by a local action (from* Σ_i*), then the two systems can reach from* c_1 *and* c_2 *again a common configuration. This property can be expressed by the following PDL*$^\cap$ *formula:*

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

Note that since $\mathcal{Y}_1$ *(resp.* $\mathcal{Y}_2$*) is assumed to be deterministic, every node of* $\mathcal{K}(\mathcal{Y}_i)$ *has at most one* a*-successor (resp.* b*-successor) for every* $a \in \Sigma_1$ *(resp.* $b \in \Sigma_2$*).*

4.3 Prefix-recognizable systems

We call $U \subseteq \Gamma^* \times \Gamma^*$ a *prefix-recognizable relation* over Γ if $U = \bigcup_{i=1}^n R_i$ for some $n \geq 1$ 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 $R_i = (U_i \times V_i)W_i$. Let $\mathrm{PRR}(\Gamma)$ denote the set of prefix-recognizable relations over Γ. A *prefix-recognizable system (PRS for short)* [1] over Σ is a tuple $\mathcal{Z} = (\Gamma, \alpha)$ where $\alpha : \Sigma \to \mathrm{PRR}(\Gamma)$ assigns to every atomic program $\sigma \in \Sigma$ a prefix-recognizable relation $\alpha(\sigma)$ such that $\alpha(\sigma) = \bigcup_{i=1}^{n_\sigma}(L(\mathcal{A}_i^\sigma) \times L(\mathcal{B}_i^\sigma))L(\mathcal{C}_i^\sigma)$ over Γ which is given by a tuple $(\mathcal{A}_1^\sigma, \ldots, \mathcal{A}_{n_\sigma}^\sigma, \mathcal{B}_1^\sigma, \ldots, \mathcal{B}_{n_\sigma}^\sigma, \mathcal{C}_1^\sigma, \ldots, \mathcal{C}_{n_\sigma}^\sigma)$ of NFAs. A PRS (Γ, α) over Σ describes the Kripke structure $(\Gamma^*, \{\alpha(\sigma) \mid \sigma \in \Sigma\})$ over Σ. The *size* $|\mathcal{Z}|$ of a PRS $\mathcal{Z} = (\Gamma, \alpha)$ over Σ is defined as $|\Gamma| + |\Sigma| + \sum_{\sigma\in\Sigma}\sum_{i=1}^{n_\sigma} |\mathcal{A}_i^\sigma| + |\mathcal{B}_i^\sigma| + |\mathcal{C}_i^\sigma|$.

Example 4.3. *Given the PRS* $\mathcal{Z} = (\{0,1\}, \alpha)$ *over* $\{0, 1, \preceq\}$ *such that* $\alpha(b) = (\{\varepsilon\} \times \{b\})\,\{0,1\}^*$ *for* $b \in \{0,1\}$ *and* $\alpha(\preceq) = (\{\varepsilon\} \times \{0,1\}^*)\,\{0,1\}^*$ *the described Kripke structure* $\mathcal{K}(\mathcal{Z})$ *is the complete binary tree with the prefix relation* $\preceq$*.*

Note that every PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ can be translated into an equivalent PRS $\mathcal{Z}(\mathcal{Y}) = (\Gamma \cup P, \alpha)$ (each over Σ), where for all $\sigma \in \Sigma$ we have

$$\alpha(\sigma) = \bigcup_{(p,a,\sigma,q,v)\in\Delta} (\{pa\} \times \{qv\})\,\Gamma^*.$$

Also note that every node of the PRS from Example 4.3 has infinitely many outgoing transitions, whereas every node of every PDS has both finitely many incoming and outgoing transitions.

[1] PRS (prefix recognizable systems) should not be confused with PRS hierarchy defined by Mayr in [22].

Depending on the context, the notion BPA (PDS, PRS) will either refer to a concrete basic process algebra (pushdown system, prefix-recognizable system) or to the class of infinite state systems described by basic process algebras (pushdown systems, prefix-recognizable systems). Moreover, when we mean a concrete BPA (PDS, PRS) we might also refer to the Kripke structure described by it.

Comparison to other definitions Note that our definition of basic process algebras allows transitions of the form $\varepsilon \xrightarrow{\sigma}_{\mathcal{X}} v$ and our definition of pushdown systems allows transitions of the form $p \xrightarrow{\sigma}_{\mathcal{Y}} qv$ where p and q are control states. It is easy to see that our definitions describe exactly the same class of basic process algebras as defined in [21, 22] and the same class of pushdown systems as defined in [31, 1, 30, 22]. Moreover, there are logspace translations between the two formalisms. When defining pushdown systems $\mathcal{Y}$ over a set of atomic propositions $\mathbb{P}$ and a set of atomic programs Σ with control states P it is well-established to introduce a mapping $\varrho : P \to 2^{\mathbb{P}}$ which assings to every control state $p \in P$ a subset of $\mathbb{P}$. If $p \in \varrho(q)$ for a control state q of $\mathcal{Y}$ and $p \in \mathbb{P}$, then all configurations of $\mathcal{Y}$ (a configuration corresponds to a node in $\mathcal{K}(\mathcal{Y})$) in current state q satisfy p. Given such a mapping $\varrho : P \to 2^{\mathbb{P}}$ our definition from above can imitate it by introducing fresh transition rules in $\mathcal{Y}$ which generate correspondingly labeled self-loops at exactly those nodes of the Kripke structure $\mathcal{K}(\mathcal{Y})$ where the atomic propositions are satisfied. More precisely, we extend the set of atomic programs Σ by $\mathbb{P}$ (assuming $\mathbb{P}$ and Σ to be disjoint) and introduce for all control states $p \in P$ the transition rule $p \xrightarrow{\eta} p$ if and only if $\eta \in \varrho(p)$. Similar remarks apply to basic process algebras and prefix-recognizable systems.
Table 1 summarizes our results for model-checking Propositional Dynamic Logic and its variants over the systems BPA, PDS and PRS.

Another way of describing the classes of infinite state systems BPA, PDS, and PRS is by defining them via transition systems generated by certain term rewriting rules where only sequential composition is allowed. Besides sequential composition also parallel composition was used for classifying infinite state systems. Mayr [22] introduced the hierarchy of *process rewrite systems* as a unified framework, where both sequential and parallel composition is allowed. This hierarchy is known to be strict w.r.t. bisimulation equivalence. Also Petri nets are a particular class in Mayr's hierarchy.

4.4 Petri nets and communication-free nets

A *Petri net* $\mathcal{N}$ over a set of atomic programs Σ is a tuple (S, T, γ, ρ) where (i) S is a finite set of *places*, (ii) T is a finite set of *transitions*, (iii) $\gamma : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a *weight function*, and (iv) $\rho : T \to \Sigma$ labels each transition with an atomic program. A *marking* M is a mapping $M : S \to \mathbb{N}$, thus $M \in \mathbb{N}^S$. Let M_1 and M_2 be markings. Then we write $M_1 \xrightarrow{\sigma} M_2$ if and only if there exists a transition $t \in T$ such that (i) $M_1(s) \geq \gamma(s,t)$ for all $s \in S$ and (ii) $M_2(s) = M_1(s) + \gamma(t,s) - \gamma(s,t)$. A Petri net $\mathcal{N} = (S, T, \gamma, \rho)$ over Σ describes the Kripke

Table 1: Model-checking PDL over BPA, PDS, and PRS.

<table>
<tr><th></th><th></th><th>BPA</th><th>PDS</th><th>PRS</th></tr>
<tr><td rowspan="3">EF, PDL\?</td><td>data</td><td>P-complete</td><td rowspan="3">PSPACE-complete</td><td>EXP -complete</td></tr>
<tr><td>expression</td><td rowspan="2"></td><td></td></tr>
<tr><td>combined</td><td>EXP-complete</td></tr>
<tr><td rowspan="3">PDL</td><td>data</td><td>P-complete</td><td colspan="2" rowspan="3">EXP-complete</td></tr>
<tr><td>expression</td><td rowspan="2"></td></tr>
<tr><td>combined</td></tr>
<tr><td rowspan="3">PDL$^\cap$, PDL$^\cap$\?</td><td>data</td><td colspan="3">EXP-complete</td></tr>
<tr><td>expression</td><td colspan="3" rowspan="2">2EXP-complete</td></tr>
<tr><td>combined</td></tr>
</table>

structure $\mathcal{K}(\mathcal{N}) = (\mathbb{N}^S, \{\to_\sigma \mid \sigma \in \Sigma\})$ where $\to_\sigma = \{(M_1, M_2) \in \mathbb{N}^S \times \mathbb{N}^S \mid M_1 \xrightarrow{\sigma} M_2\}$ for all $\sigma \in \Sigma$.

Petri nets can also be described by transitions systems which are generated by rewriting rules where only parallel composition is allowed on the left and on the right hand side of each rewriting rule. A subclass of Petri nets, called *basic parallel processes* (BPP for short), is defined by allowing parallel composition only at the right hand side of each rewriting rule. This class is also known as the class of *communication-free nets* to be defined below. When referring to this class we write BPP.

A *communication-free net* over Σ is a Petri net $\mathcal{N} = (S, T, \gamma, \rho)$ over Σ such that for all $t \in T$ there exists a unique $s \in S$ such that (i) $\gamma(s,t) = 1$ and for all $s' \neq s$ we have $\gamma(s', t) = 0$.

5 Undecidability of test-free PDL over BPP

It is shown in [23] that there already exists a fixed EF formula such that the model-checking problem for Petri nets becomes undecidable. By a simple reduction from this problem one sees that the model-checking problem of test-free PDL over BPP is undecidable. Note that in [23] it is shown that the combined complexity of the model-checking problem of EF over BPP is PSPACE-complete.

Theorem 5.1 (Proposition 36 in [23])**.** *There exists a fixed EF formula φ over Σ such that the following problem is undecidable:*
INPUT: A Petri net $\mathcal{N} = (S, T, \gamma, \rho)$ over Σ and an initial marking $M \in \mathbb{N}^S$.
QUESTION: $(\mathcal{K}(\mathcal{N}), M) \models \varphi$?

Corollary 5.2. *The following problem is undecidable:*
INPUT: A communication-free net $\mathcal{N} = (S, T, \gamma, \rho)$ *over* Σ*, an initial marking* $M \in \mathbb{N}^S$ *and a test-free PDL formula* φ *over* Σ*.*
QUESTION: $(\mathcal{K}(\mathcal{N}), M) \models \varphi$*?*

Proof. Given the fixed EF formula φ of Theorem 5.1 and a Petri net $\mathcal{N}' = (S', T', \gamma', \rho')$ each over Σ' and an initial marking $M' \in \mathbb{N}^{S'}$, we construct (in logarithmic space) a test-free PDL formula $\|\varphi\|$ and a communication-free net $\mathcal{N} = (S, T, \gamma, \rho)$ over Σ such that:

$$(\mathcal{K}(\mathcal{N}'), M') \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{N}), M) \models \|\varphi\|$$

We put $T = \Sigma = T' \cup (S' \times T')$, $S = S' \uplus \{s_0\}$ for a fresh place s_0. Moreover we put $\rho = \mathrm{id}_T = \mathrm{id}_\Sigma$, $M(s) = M'(s)$ for all $s \in S$, $M(s_0) = 1$ and define γ as follows, where $s \in S$ and $t \in T$:

$$\gamma(s,t) = \begin{cases} 1 & \text{if } s \in S' \text{ and } t = (s, t') \text{ for a } t' \in T' \\ 1 & \text{if } s = s_0 \text{ and } t \in T' \\ 0 & \text{else} \end{cases}$$

and

$$\gamma(t,s) = \begin{cases} \gamma'(t,s) & \text{if } t \in T' \text{ and } s \in S' \\ 1 & \text{if } s = s_0 \text{ and } t \in T' \\ 0 & \text{else} \end{cases}$$

Clearly, $\mathcal{N}$ is a communication-free net. Let us assume that $S' = \{s_1, \ldots, s_n\}$. We translate the formula φ into $\|\varphi\|$ inductively as follows:

$$\begin{aligned}
\|\mathtt{true}\| &= \mathtt{true} \\
\|\neg\varphi\| &= \neg\|\varphi\| \\
\|\varphi_1 \vee \varphi_2\| &= \|\varphi_1\| \vee \|\varphi_2\| \\
\|\langle\sigma\rangle\varphi\| &= \left\langle \bigcup_{t \in \rho'^{-1}(\sigma)} \left(\bigcirc_{i=1}^{n} (s_i, t)^{\gamma'(s_i,t)} \right) \circ t \right\rangle \|\varphi\| \\
\|\exists \mathrm{F}\varphi\| &= \left\langle \left(\bigcup_{\substack{\sigma \in \Sigma' \\ t \in \rho'^{-1}(\sigma)}} \left(\bigcirc_{i=1}^{n} (s_i, t)^{\gamma'(s_i,t)} \right) \circ t \right)^{*} \right\rangle \|\varphi\|
\end{aligned}$$

It is not difficult to see that we have

$$(\mathcal{K}(\mathcal{N}'), M') \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{N}), M) \models \|\varphi\|.$$

□

6 A technique for lower bounds

Since we can consider a BPA, PDS, or PRS as a system $\mathcal{S}$ that manipulates a stack containing symbols of a finite alphabet we will, on a meta level, for the rest of this section talk about *stack systems*. For several lower bound proofs, we will use a technique introduced by Walukiewicz by which he showed the EXP-hardness of model-checking CTL properties on pushdown systems [30] . For a fixed space-bounded alternating Turing machine $\mathcal{M}$ this technique translates the behaviour of $\mathcal{M}$ on an input w into a stack system $\mathcal{S}$ that traverses the computation tree of $\mathcal{M}$ on input w in a depth first search manner assured by a forumla φ. For an input w of $\mathcal{M}$ the content of the stack of $\mathcal{S}$ is a string that represents a path from the initial configuration $\mathrm{init}(w)$ to a reachable configuration by separating all intermediate configurations on this path by certain markers to be described below. Depending on the space bound of $\mathcal{M}$ simple or more involved representations of the configurations of $\mathcal{M}$ as the a content of the stack of $\mathcal{S}$ are used. Left to any existential or universal configuration c we put so called *direction markers*. In case c is universal a direction marker contains the information whether the acceptance of the left or the right successor configuration of c is checked and which move is done. In case c is existential, the direction marker contains the information about the move that is done in order to reach the left or the right successor configuration of c which the acceptance is checked for. For universal configurations $c_\forall$ in current state $q \in Q_\forall$ and scanning a symbol $\gamma \in \Gamma$ such that $\delta(q,\gamma) = (\mu_1,\mu_2)$ for $\mu_1,\mu_2 \in$ Moves the direction marker $L(\mu_1,\mu_2)$ $(R(\mu_1,\mu_2))$ is used to separate $c_\forall$ from its left (right) successor configuration c, i.e. $c_\forall \vdash^{\mu_1}_{\mathcal{M}} c$ $(c_\forall \vdash^{\mu_2}_{\mathcal{M}} c)$. For existential configurations $c_\exists$ in current state $q \in Q_\exists$ and scanning a symbol $\gamma \in \Gamma$ such that $\delta(q,\gamma) = (\mu_1,\mu_2)$ the marker $E(\mu_1)$ $(E(\mu_2))$ is used to separate $c_\exists$ from its left (right) successor c, i.e. $c_\exists \vdash^{\mu_1}_{\mathcal{M}} c$ $(c_\exists \vdash^{\mu_2}_{\mathcal{M}} c)$. Let $\mathrm{Dir}_\forall = \{L(\mu_1,\mu_2) \mid (\mu_1,\mu_2) \in \delta(Q_\forall,\Gamma)\} \cup \{R(\mu_1,\mu_2) \mid (\mu_1,\mu_2) \in \delta(Q_\forall,\Gamma)\}$ denote the set of *universal direction markers* and $\mathrm{Dir}_\exists = \{E(\mu_1), E(\mu_2) \mid (\mu_1,\mu_2) \in \delta(Q_\exists,\Gamma)\}$ denote the set of *existential direction markers*. As in [30] these direction markers are used in order to organize a depth-first left-to-right traversal of the computation tree of $\mathcal{M}$ on input w on the stack of $\mathcal{S}$. Our stack system $\mathcal{S}$ has to be able to do and our formula φ has to assure the following:

- When the top of the stack of $\mathcal{S}$ is an accepting configuration c this accepting configuration has to be popped from the stack. This removal confirms that c is accepting.
- When the top of the stack of $\mathcal{S}$ consists of $E(\mu) \in \mathrm{Dir}_\exists$ followed by an existential configuration $c_\exists$, all these parts have to be popped from the stack, thus confirming that $c_\exists$ is an accepting configuration of $\mathcal{M}$.
- The situation is a bit different when the top of the stack of $\mathcal{S}$ is $L(\mu_1,\mu_2) \in \mathrm{Dir}_\forall$ followed by a universal configuration $c_\forall$. In this case it is already assured that the left successor configuration c_1 of $c_\forall$ (i.e. $c_\forall \vdash^{\mu_1}_{\mathcal{M}} c_1$) is accepting as it has been popped from the stack. All $\mathcal{S}$ now has to do is to check that $c_\forall$'s right successor configuration c_2 (i.e. $c_\forall \vdash^{\mu_2}_{\mathcal{M}} c_2$) is accepting too. Hence, $L(\mu_1,\mu_2)$ has to be replaced by $R(\mu_1,\mu_2)$ and c_2 has to be pushed on top of the stack. For this, an arbitrary configuration c is pushed on the stack and then $\mathcal{S}$ verifies whether indeed $c_\forall \vdash^{\mu_2}_{\mathcal{M}} c$ holds.

- When $R(\mu_1, \mu_2) \in \mathrm{Dir}_\forall$ followed by a universal configuration $c_\forall$ is on top of the stack all these parts have to be popped from the stack, thus confirming $c_\forall$ to be accepting.
- When the top of the stack of $\mathcal{S}$ is a universal configuration $c_\forall$ in current state $q \in Q_\forall$ and scanning a symbol $\gamma \in \Gamma$ such that $\delta(q, \gamma) = (\mu_1, \mu_2)$ and $c_\forall \vdash^{\mu}_{\mathcal{M}} c_1$, the string $c_1 L(\mu_1, \mu_2)$ has to be pushed on top of the stack, thus initiating to check the acceptance of c_1.
- Analogously, when the top of the stack of $\mathcal{S}$ is an existential configuration $c_\exists$ in current state $q \in Q_\exists$ and scanning a symbol $\gamma \in \Gamma$ such that $\delta(q, \gamma) = (\mu_1, \mu_2)$, a string $cE(\mu)$ has to be pushed on top of the stack where $c_\exists \vdash^{\mu}_{\mathcal{M}} c$ for a $\mu \in \{\mu_1, \mu_2\}$, thus initiating to check the acceptance of c.

The behavior of $\mathcal{S}$ and the assurance of the formula φ express the fact that there exists a path through the computation tree of $\mathcal{M}$ such that the initial configuration $\mathrm{init}(w)$ is accepting, i.e., after pushing and popping configurations separated by apropriate direction markers on and from the stack a finite number of times, the stack eventually becomes empty. Depending on what sorts of lower bounds we want to prove (data, expression, or combined complexity) our stack system $\mathcal{S}$ and our formula φ depend on (the size of) the input w of $\mathcal{M}$ or not.

7 Model-checking test-free PDL

7.1 Lower Bounds

A *context-free grammar* (CFG for short) is a quadruple $G = (V, \Sigma, S, P)$ where (i) V is a finite set of *variable symbols*, (ii) Σ is a finite *terminal alphabet*, (iii) $S \in V$ is the *starting variable*, and (iv) $P : V \to 2^{(V \cup \Sigma)^*}$ is the set of *production rules*. We denote with $L(G)$ the *language* generated by the context-free grammar G. The size $|G|$ of G is defined as $|V| + |\Sigma| + \sum_{X \in V} \sum_{v \in P(X)} |v|$.

Theorem 7.1 ([14]). *The following problem is* P*-complete:*
INPUT: A CFG $G = (V, \emptyset, S, P)$ *with empty terminal alphabet.*
QUESTION: $L(G) \neq \emptyset$*? (i.e.* $\varepsilon \in L(G)$*?)*

By a simple reduction from the non-emptiness problem for context-free grammars, we can prove:

Theorem 7.2. *For the fixed EF formula* $\varphi = \langle \mathrm{init} \rangle (\exists \mathrm{F}\, \neg \langle \mathrm{loop} \rangle \mathtt{true})$ *over* $\Sigma = \{\mathrm{init}, \mathrm{derive}, \mathrm{loop}\}$ *the following problem is* P*-hard:*
INPUT: A BPA $\mathcal{X} = (\Gamma, \Delta)$ *over* Σ*.*
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$*?*

Proof. We give a logarithmic space reduction from the non-emptiness problem for context-free grammars to the above problem. Thus, the theorem will follow from Theorem 7.1. Let

$G = (V, \emptyset, S, P)$ be a CFG over the empty terminal alphabet. We construct a BPA $\mathcal{X} = \mathcal{X}(G) = (V, \Delta)$ over Σ where

$$\Delta = \{(\varepsilon, \mathrm{init}, S)\} \cup \{(X, \mathrm{derive}, w) \mid X \in V, w \in P(X)\} \cup \{(X, \mathrm{loop}, X) \mid X \in V\}.$$

It is obvious that we have:

$$L(G) \neq \emptyset \quad \Leftrightarrow \quad \varepsilon \in L(G) \quad \Leftrightarrow \quad (\mathcal{K}(\mathcal{X}), \varepsilon) \models \langle \mathrm{init} \rangle (\exists \mathrm{F}\, \neg \langle \mathrm{loop} \rangle \mathtt{true})$$

□

Hence, we get a P lower bound for the data complexity of EF over BPA. Over PDS, Bouajjani et al. proved a PSPACE lower bound for the data complexity of EF:

Theorem 7.3 ([1]). *There exists a fixed* EF*-formula φ over Σ such that the following problem is* PSPACE*-hard:*
INPUT: A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ and a control state $p_0 \in P$.
QUESTION: $(\mathcal{K}(\mathcal{Y}), p_0) \models \varphi$?

Concerning expression complexity of EF over BPA, a PSPACE lower bound was shown by Mayr:

Theorem 7.4 ([21]). *There exists a fixed BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ such that the following problem is* PSPACE*-hard:*
INPUT: An EF formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$?.
Moreover, this lower bound even holds if the input φ is a formula of modal logic.

For the rest of this section, our goal is to show an EXP lower bound of the data complexity of EF over PRS. We will proceed in several steps. First we show an EXP lower bound for the combined complexity of test-free PDL over PRS.

Theorem 7.5. *The following problem is* EXP*-hard:*
INPUT: A PRS $\mathcal{Z} = (\Gamma, \alpha)$ and a test-free PDL-program π each over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \pi \rangle \mathtt{true}$?

Proof. Let $\mathcal{M} = (Q, \Sigma_{\mathcal{M}}, \Gamma_{\mathcal{M}}, q_0, \delta, \Box)$ be a fixed $p(m)$-space-bounded ATM for a polynomial $p(m)$ with an EXP-hard acceptance problem. Moreover, let $w \in \Sigma_{\mathcal{M}}^*$ be an input of length n. We will construct in logarithmic space a PRS $\mathcal{Z} = \mathcal{Z}(\mathcal{M}, w) = (\Gamma, \alpha)$ and a test-free PDL-program $\pi = \pi(\mathcal{M}, w)$ each over a set of atomic programs Σ such that:

$$w \in L(\mathcal{M}) \quad \Leftrightarrow \quad (\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \pi \rangle \mathtt{true}$$

We will use the technique described in Subsection 6. Let $N = p(n), \Omega = Q \cup \Gamma_{\mathcal{M}}$ and let Dir denote the set of all direction markers of $\mathcal{M}$. The alphabet we will use in order to represent a sequence of configurations of $\mathcal{M}$ separated by direction markers is $\Gamma = \Omega \cup \mathrm{Dir} = Q \cup \Gamma_{\mathcal{M}} \cup \mathrm{Dir}$.

We represent a configuration of $\mathcal{M}$ as an element of the language $\bigcup_{i=0}^{N-1} \Gamma_{\mathcal{M}}^{i} Q \Gamma_{\mathcal{M}}^{N-i}$. We introduce the set Σ of atomic programs and assign to each atomic program $\sigma \in \Sigma$ a prefix-recognizable relation $\alpha(\sigma)$ to be implicitly given below. First, we introduce the atomic program popacc which allows to pop an accepting configuration from the stack:

$$\alpha(\text{popacc}) \quad = \quad \bigcup_{i=0}^{N-1} \left(\Gamma_{\mathcal{M}}^{i} Q_{\text{acc}} \Gamma_{\mathcal{M}}^{N-i} \times \{\varepsilon\}\right) \Gamma^{*}$$

For all direction markers $d \in \text{Dir}$ we introduce the atomic program separates_d which checks whether the top of the stack is a word in the language $\Omega^{N+1}\, d\, \Omega^{N+1}$:

$$\alpha(\text{separates}_d) \quad = \quad (\{\varepsilon\} \times \{\varepsilon\})\, \Omega^{N+1}\, d\, \Omega^{N+1} \Gamma^{*}$$

For all $q \in Q$ and $\gamma \in \Gamma_{\mathcal{M}}$ we introduce the program $\text{check}_{q,\gamma}$ which allows to check if the top configuration on the stack is in state q and the read-write head of $\mathcal{M}$ is currently scanning γ:

$$\alpha(\text{check}_{q,\gamma}) \quad = \quad (\{\varepsilon\} \times \{\varepsilon\})\, \Gamma_{\mathcal{M}}^{*}\, q\gamma\, \Gamma^{*}$$

For all $0 \leq j \leq N$ and $\omega, \omega' \in \Omega$ we introduce the atomic program $\text{match}_{j,\omega,\omega'}$ which allows to check if the top configuration has the symbol ω at position $j+1$ and the subsequent configuration on the stack has the symbol ω' at position $j+1$:

$$\alpha(\text{match}_{j,\omega,\omega'}) \quad = \quad (\{\varepsilon\} \times \{\varepsilon\})\, \Omega^{j} \omega \Omega^{N-j}\ \text{Dir}\ \Omega^{j} \omega' \Omega^{N-j} \Gamma^{*}$$

We now introduce the atomic program popright which pops a sequence $R(\mu_1, \mu_2)v$ with $v \in \Omega^{N+1}$ and $R(\mu_1, \mu_2) \in \text{Dir}_{\forall}$ from the stack. The word v should (this will be assured by π) be a universal configuration for which both left and right successor configurations are accepting. In analogy, for the existential case, the atomic program popexist pops a sequence $E(\mu)v$ with $v \in \Omega^{N+1}$ and $E(\mu) \in \text{Dir}_{\exists}$ from the stack:

$$\alpha(\text{popright}) \quad = \quad \left(\bigcup_{R(\mu_1,\mu_2) \in \text{Dir}_{\forall}} R(\mu_1, \mu_2) \Omega^{N+1} \times \{\varepsilon\} \right) \Gamma^{*}$$

$$\alpha(\text{popexist}) \quad = \quad \left(\bigcup_{E(\mu) \in \text{Dir}_{\exists}} E(\mu) \Omega^{N+1} \times \{\varepsilon\} \right) \Gamma^{*}$$

In the same manner for all $(\mu_1, \mu_2) \in \delta(Q_{\forall}, \Gamma_{\mathcal{M}})$ we introduce the atomic program $\text{pushleft}_{\mu_1,\mu_2}$ which, assuming the top of the stack to be a universal configuration, pushs the left successor configuration of the top configuration (this will be assured by π) on top of the stack:

$$\alpha(\text{pushleft}_{\mu_1,\mu_2}) \quad = \quad \left(\{\varepsilon\} \times \Omega^{N+1} L(\mu_1, \mu_2)\right) \Gamma^{*}$$

Analogously for all $(\mu_1, \mu_2) \in \delta(Q_\exists, \Gamma_\mathcal{M})$ and $i \in \{1,2\}$ we introduce the atomic program $\mathrm{pushexist}_{\mu_i}$:

$$\alpha(\mathrm{pushexist}_{\mu_i}) \;=\; \big(\{\varepsilon\} \times \Omega^{N+1} E(\mu_i)\big)\, \Gamma^*$$

Once the left successor configuration of a universal configuration is accepting (i.e. it has just been popped from the stack), we can pop the top direction marker $L(\mu_1, \mu_2)$ from the stack and push its right successor configuration with the apropriate direction marker $R(\mu_1, \mu_2)$ on top of the stack. Hence, for all $(\mu_1, \mu_2) \in \delta(Q_\forall, \Gamma_\mathcal{M})$ we add the atomic program $\mathrm{leftright}_{\mu_1,\mu_2}$ as follows:

$$\alpha(\mathrm{leftright}_{\mu_1,\mu_2}) \;=\; \big(L(\mu_1, \mu_2) \times \Omega^{N+1} R(\mu_1, \mu_2)\big)\, \Gamma^*$$

The atomic program init adds the initial configuration $q_0 w \Box^{N-n}$ on top of the empty stack:

$$\alpha(\mathrm{init}) \;=\; \big(\{\varepsilon\} \times \{q_0 w \Box^{N-n}\}\big)\, \{\varepsilon\}$$

The atomic program empty inserts the simple loop $(\varepsilon, \varepsilon)$.

$$\alpha(\mathrm{empty}) \;=\; (\{\varepsilon\} \times \{\varepsilon\})\, \{\varepsilon\}$$

We now give some auxiliary programs which we will later put together:

- For all $d \in \mathrm{Dir}$ we introduce the program $\mathrm{testsucc}_d$. It checks the following: The top of the stack is of the form $c_1 d c_2$ and c_1 is the successor configuration of c_2 w.r.t. the direction marker d. We give the program $\mathrm{testsucc}_d$ only for the case that d is of the form $L(\mu_1, \mu_2)$ where $\mu_1 = (q', \gamma', \leftarrow)$ for some $q' \in Q$ and $\gamma' \in \Gamma_\mathcal{M}$. The other cases can be handled analogously:

$$\begin{aligned}\mathrm{testsucc}_d \;=\; &\mathrm{separates}_d \circ \bigcup_{1 \le i \le N-1} \bigcirc_{j=0}^{i-2} \left(\cup_{\gamma \in \Gamma_\mathcal{M}} \mathrm{match}_{j,\gamma,\gamma}\right) \circ \\ &\left(\bigcup_{\substack{\gamma, \gamma'' \in \Gamma_\mathcal{M} \\ q \in Q}} \mathrm{match}_{i-1,q',\gamma''} \circ \mathrm{match}_{i,\gamma'',q} \circ \mathrm{match}_{i+1,\gamma',\gamma} \right) \circ \\ &\bigcirc_{j=i+2}^{N} \left(\cup_{\gamma \in \Gamma_\mathcal{M}} \mathrm{match}_{j,\gamma,\gamma}\right)\end{aligned}$$

- The following program traverses the configuration tree of $\mathcal{M}$ along one edge:

$$
\begin{aligned}
\text{traverse} \;=\;& \text{popacc} \cup \text{popright} \cup \text{popexist} \\
& \cup \bigcup_{\gamma \in \Gamma} \Bigg(\bigcup_{\substack{q \in Q_\forall \\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \text{check}_{q,\gamma} \circ \text{pushleft}_{\mu_1,\mu_2} \circ \text{testsucc}_{L(\mu_1,\mu_2)} \\
& \quad \cup \bigcup_{\substack{q \in Q_\exists, i \in \{1,2\} \\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \text{check}_{q,\gamma} \circ \text{pushexist}_{\mu_i} \circ \text{testsucc}_{E(\mu_i)} \\
& \quad \cup \bigcup_{(\mu_1,\mu_2) \in \delta(Q_\forall,\gamma)} \text{leftright}_{\mu_1,\mu_2} \circ \text{testsucc}_{R(\mu_1,\mu_2)} \Bigg)
\end{aligned}
$$

Our final program π is:

$$\pi \;=\; \text{init} \circ \text{traverse}^* \circ \text{empty}$$

Finally, we have:

$$w \in L(\mathcal{M}) \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \pi \rangle \texttt{true}$$

□

We need the following lemma to prove an EXP lower bound for the data complexity of EF over PRS:

Lemma 7.6. *There exists a fixed EF-formula φ over a fixed set Θ of atomic programs such that the following problem is computable in logarithmic space:*
INPUT: A PRS $\mathcal{Z} = (\Gamma, \alpha)$ and a test-free PDL program π each over some Σ.
OUTPUT: A PRS $\mathcal{Z}' = (\Gamma', \alpha')$ over Θ such that:

$$(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \pi \rangle \texttt{true} \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Z}'), \varepsilon) \models \varphi$$

Proof. According to Remark 3.1 we can compute in logarithmic space an APDL automaton $\mathcal{A}_\pi = (Q, \Sigma, q_0, F, \delta)$ (without test-symbols) such that $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \pi \rangle \texttt{true}$ if and only if $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle \mathcal{A}_\pi \rangle \texttt{true}$. We can assume $Q \cap \Gamma = \emptyset$. Define $M = Q \times \Sigma_\varepsilon \times Q$. For a prefix-recognizable relation $S = \bigcup_{i \in I} (U_i \times V_i) W_i \in \mathrm{PRR}(\Gamma)$ and two states $q, q' \in Q$ let $(q,q')S \in \mathrm{PRR}(\Gamma \cup Q)$ denote the following prefix-recognizable relation:

$$(q, q')S \;=\; \bigcup_{i \in I} (\{q\} U_i \times \{q'\} V_i) W_i$$

For each $m \in M$ we define the prefix-recognizable relation $R_m \in \mathrm{PRR}(\Gamma \cup Q)$ as follows:

$$R_m = \begin{cases} (q, q')\alpha(\sigma) & \text{if } m = (q, \sigma, q') \in Q \times \Sigma \times Q \text{ and } q' \in \delta(q, \sigma) \\ (\{q\} \times \{q'\})\Gamma^* & \text{if } m = (q, \varepsilon, q') \in Q \times \{\varepsilon\} \times Q \text{ and } q' \in \delta(q, \varepsilon) \\ \emptyset & \text{else} \end{cases}$$

Define $\Theta = \{\text{init}, \text{sim}, \text{final}\}$. We put $\alpha'(\text{init}) = (\{\varepsilon\} \times \{q_0\})\{\varepsilon\}$, $\alpha'(\text{sim}) = \bigcup_{m \in M} R_m$ and $\alpha'(\text{final}) = \bigcup_{q \in F}(\{q\} \times \{q\})\Gamma^*$. It is easy to see that the translation can be carried out in logarithmic space and that for $\varphi = \langle\text{init}\rangle(\exists\mathrm{F}\langle\text{final}\rangle\mathtt{true})$ we have:

$$(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle\pi\rangle\mathtt{true} \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Z}), \varepsilon) \models \langle\mathcal{A}_\pi\rangle\mathtt{true} \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Z}'), \varepsilon) \models \varphi$$

□

Corollary 7.7. *There exists a fixed EF formula φ over a fixed set of atomic programs Σ such that the following problem is* EXP*-hard:*
INPUT: A PRS $\mathcal{Z} = (\Gamma, \alpha)$ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$?

Proof. A direct consequence from Theorem 7.5 and Lemma 7.6. □

Hence, we get an EXP lower bound for the data complexity of EF over prefix-recognizable systems.

7.2 Upper bounds

Again in [30], Walukiewicz shows that the combined complexity of EF over pushdown systems is in PSPACE:

Theorem 7.8 ([30])**.** *The following problem is* PSPACE*-complete:*
INPUT: A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ, a control state $p \in P$, and an EF *forumula φ over Σ.*
QUESTION: $(\mathcal{K}(\mathcal{Y}), p) \models \varphi$?

Using Theorem 7.8, we can show:

Theorem 7.9. *The following problem is in* PSPACE*:*
INPUT: A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ, a control state $p \in P$, and a test-free PDL formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Y}), p_0) \models \varphi$?

Proof. We construct in polynomial time a PDS $\mathcal{Y}' = (\Gamma, P', \Delta')$ over Σ', a control state $p_0' \in P'$, and an EF-formula $\|\varphi\|$ over Σ' such that:

$$(\mathcal{K}(\mathcal{Y}), p_0) \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Y}'), p_0') \models \|\varphi\|$$

Thus, the theorem follows from Theorem 7.8. W.l.o.g. we can assume φ to be a box formula. Let $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ an enumeration of the set of all box subformulas of φ such that: φ_i is a strict subformula of φ_j implies $i < j$. For all i let $\mathcal{A}_i = (Q_i, \Sigma, q_i, F_i, \delta_i)$ be an APDL automaton for the box program of φ_i. We define $Q = \biguplus_{i \in I} Q_i$, $P' = (P \times Q) \uplus \{p_0'\}$ for a fresh control state p_0', and $\Sigma' = Q \cup \{\varepsilon\}$. The transition relation Δ' of $\mathcal{Y}'$ consists of $p_0' \xrightarrow{\varepsilon}_{\mathcal{Y}'} (p_0, q_n)$ and the following kinds of rules:

- $(p,q) \xrightarrow{q}_{\mathcal{Y}'} (p,q)$ for all $(p,q) \in P \times Q$.
- $(p,q)a \xrightarrow{\varepsilon}_{\mathcal{Y}'} (p',q')v$ if and only if there exists a transition $pa \xrightarrow{\sigma}_{\mathcal{Y}} p'v$ and $q' \in \delta_i(q,\sigma)$ where $q,q' \in Q_i$ $(1 \leq i \leq n)$.
- $(p,q) \xrightarrow{\varepsilon}_{\mathcal{Y}'} (p,q')$ if and only if $q' \in \delta_i(q,\varepsilon)$ where $q,q' \in Q_i$ and $p \in P$ $(1 \leq i \leq n)$.
- $(p,q) \xrightarrow{\varepsilon}_{\mathcal{Y}'} (p,q_j)$ where $q \in F_i, p \in P$ and $1 \leq j < i \leq n$.

We give the definition of $\|\varphi\|$ inductively:

$$\begin{aligned}
\|\texttt{true}\| &= \texttt{true} \\
\|\neg\varphi\| &= \neg\|\varphi\| \\
\|\varphi_1 \vee \varphi_2\| &= \|\varphi_1\| \vee \|\varphi_2\| \\
\|\langle\mathcal{A}_i\rangle\varphi\| &= \langle\varepsilon\rangle \left(\langle q_i\rangle\texttt{true} \wedge \exists\mathrm{F} \left(\bigvee_{q\in F_i} \langle q\rangle\texttt{true} \wedge \|\varphi\| \right)\right)
\end{aligned}$$

It is easy to see that the translation is computable in polynomial time and that we have:

$$(\mathcal{K}(\mathcal{Y}), p_0) \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Y}'), p_0') \models \|\varphi\|$$

$\square$

Hence, we get a PSPACE upper bound for the combined complexity of test-free PDL over pushdown systems.

In [26] it is shown that each prefix-recognizable system $\mathcal{Z}$ can effectively be translated into a pushdown system $\mathcal{Y}$ with ε-transitions, such that every node in $\mathcal{K}(\mathcal{Y})$ either has no outgoing ε-transitions or no outgoing non-ε-transitions, see also [3]. Moreover, one gains the prefix-recognizable system $\mathcal{Z}$ by factoring out the ε-transitions of $\mathcal{Y}$ as follows: One takes only those nodes in $\mathcal{K}(\mathcal{Y})$ without outgoing ε-transitions (we call them *ε-free nodes*) and adds a σ-transition between two nodes if and only if there exists a path between them in $\mathcal{K}(\mathcal{Y})$ consisting of one σ-transition followed by arbitrary many ε-transitions. It is now easy to see that Theorem 7.9 implies a PSPACE upper bound for the expression complexity of test-free PDL over prefix-recognizable systems: Given a fixed PRS $\mathcal{Z} = (\Gamma, \alpha)$ and a test-free PDL formula φ each over Σ, we effectively compute in constant time a fixed pushdown system $\mathcal{Y}$ with ε-transitions and the properties from above. We add transitions to $\mathcal{Y}$ such that each ε-free node of $\mathcal{K}(\mathcal{Y})$ has a self-loop labeled by $\# \notin \Sigma$. By this modification we obtain the fixed pushdown system $\mathcal{Y}'$ over $\Sigma \cup \{\#\}$ with ε-transitions. Finally, we replace each occurring atomic program $\sigma \in \Sigma$ in φ by $\sigma \circ \varepsilon^* \circ \#$ and hereby receive φ'. Clearly this construction can be carried out in polynomial time and for some fixed node c of $\mathcal{K}(\mathcal{Y}')$ (which corresponds to the node ε of $\mathcal{K}(\mathcal{Z})$). we have:

$$(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{Y}'), c) \models \varphi'$$

Hence, we get the following corollary:

Table 2: EF and test-free PDL over BPA,PDS and PRS.

<table>
<tr><th></th><th></th><th>BPA</th><th>PDS</th><th>PRS</th></tr>
<tr><td rowspan="3">EF, PDL\?</td><td>data</td><td>P-complete</td><td rowspan="3">PSPACE-complete</td><td>EXP -complete</td></tr>
<tr><td>expression</td><td>PSPACE-complete</td><td>PSPACE-complete</td></tr>
<tr><td>combined</td><td>PSPACE-complete</td><td>EXP-complete</td></tr>
</table>

Corollary 7.10. *For every fixed PRS $\mathcal{Z} = (\Gamma, \alpha)$ over Σ the following problem is in* PSPACE*:*
INPUT: A test-free PDL formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$?

Hence, we get a PSPACE upper bound for the expression complexity of test-free PDL over prefix-recognizable systems.

We obtain the complexity results for model-checking EF and test-free PDL over BPA, PDS, and PRS in Table 2; the P and EXP upper bounds are treated in Section 8.2.

8 Model-checking full PDL

8.1 Lower Bounds

In [30], Walukiewicz has shown that the combined complexity of CTL over pushdown systems is EXP-complete. His lower bound proof easily shows that the EXP lower bound already holds for a fixed pushdown system:

Theorem 8.1 ([30])**.** *There exists a fixed PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ and a control state $p \in P$ such that the following problem is* EXP*-hard:*
INPUT: A CTL formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Y}), p) \models \varphi$?

Looking at the construction in [30], the CTL formula φ over Σ establishing the EXP lower bound for model-checking the fixed pushdown system $\mathcal{Y} = (\Gamma, P, \Delta)$ and a control state $p_0 \in P$ is of the form $\varphi = \exists(\psi_1 \mathcal{U} \psi_2)$, where ψ_1 and ψ_2 are EF formulas. It is easy to see that the PDL formula $\|\varphi\| = \langle (\|\psi_1\|? \circ \Sigma)^* \rangle \|\psi_2\|$ is equivalent to φ, where $\|\psi_1\|$ ($\|\psi_2\|$) is a logarithmic space computable test-free PDL formula equivalent to ψ_1 (ψ_2) see by Remark 3.4. We now construct the formula φ' by taking $\|\varphi\|$ and replacing any atomic program $\sigma \in \Sigma$ by

$$\bigcup_{p\gamma \xrightarrow{\sigma}_{\mathcal{Y}} qv} \overline{p} \circ \overline{\gamma} \circ v^{\mathrm{rev}} \circ q \cup \bigcup_{p \xrightarrow{\sigma}_{\mathcal{Y}} qv} \overline{p} \circ v^{\mathrm{rev}} \circ q.$$

We finally have the following, where the fixed BPA $\mathrm{Tree}_{\Gamma\cup P}$ is from Example 4.1:

$$(\mathcal{K}(\mathcal{Y}), p_0) \models \varphi \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathrm{Tree}_{\Gamma\cup P}), p_0) \models \varphi'$$

Hence, we obtain an EXP lower bound for the expression complexity of PDL over BPA.

Corollary 8.2. *There exists a fixed BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ and a fixed node $w \in \Gamma^*$ such that the following problem is* EXP*-hard:*
INPUT: A PDL formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{X}), w) \models \varphi$?

Also the CTL formula used by Walukiewicz in his EXP lower bound proof for the combined complexity of CTL over PDS [30] can be easily adapted to be of fixed size:

Theorem 8.3 ([30])**.** *There exists a fixed CTL-formula φ over some fixed Σ such that the following problem is* EXP*-hard:*
INPUT: A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ and a control state $p \in P$.
QUESTION: $(\mathcal{K}(\mathcal{Y}), p) \models \varphi$?

Since the fixed CTL formula φ establishing the EXP lower bound for model checking pushdown systems is of the form $\varphi = \exists(\psi_1 \mathcal{U} \psi_2)$ for two EF formulas ψ_1 and ψ_2 we can, as discussed above, translate this formula into an equivalent fixed PDL formula. Hence, we obtain an EXP lower bound for the data complexity of PDL over pushdown systems.

Corollary 8.4. *There exists a fixed PDL-formula φ over Σ such that the following problem is* EXP*-hard:*
INPUT: A PDS $\mathcal{Y} = (\Gamma, P, \Delta)$ over Σ and a control state $p \in P$.
QUESTION: $(\mathcal{K}(\mathcal{Y}), p) \models \varphi$?

8.2 Upper bounds

Since there is a polynomial time translation from PDL into (the alternation-free fragment of the) modal μ-calculus [13], upper bounds for the modal μ-calculus carry over to PDL. Since for a fixed μ-calculus formula the model-checking problem over BPA is in P [31], the data complexity of model-checking PDL over BPA is in P too. Since the combined complexity of model-checking the modal μ-calculus over PRS is in EXP [16, 4], we also obtain an EXP upper bound for the combined complexity of model-checking PDL over PRS.

Table 3 summarizes the results for model-checking full PDL over BPA, PDS, and PRS.

9 Model-checking PDL with intersection

9.1 Lower bounds

First, we give an EXP lower bound for the data complexity of test-free $\mathrm{PDL}^{\cap}$ over basis parallel processes.

Table 3: Full PDL over BPA, PDS, and PRS.

		BPA	**PDS**	**PRS**
PDL	**data**	P-complete		
	expression		EXP	
	combined			

Theorem 9.1. *There exists a fixed test-free PDL$^\cap$ formula φ over some fixed Σ such that the following problem is* EXP*-hard:*
INPUT: A BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$?

Proof. Let $\mathcal{M} = (Q, \Sigma_\mathcal{M}, \Gamma_\mathcal{M}, q_0, \delta, \Box)$ be fixed $p(n)$ space bounded ATM for some polynomial p with an EXP-hard acceptance problem. Moreover, we assume that $\mathcal{M}$ fulfills the conventions of Subsection 6. Let $w = w_1 w_2 \cdots w_n \in \Sigma_\mathcal{M}^*$ be an input of length n. We give a logarithmic space computable BPA $\mathcal{X} = \mathcal{X}(\mathcal{M}, w)$ and a test-free PDL$^\cap$ formula $\varphi = \varphi(\mathcal{M})$ both over $\Sigma = \Sigma(\mathcal{M})$ such that we have:

$$w \in L(\mathcal{M}) \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi.$$

Let Moves be the set of moves of $\mathcal{M}$, let $\mathrm{Dir} = \mathrm{Dir}_\exists \cup \mathrm{Dir}_\forall$ be the set of direction markers of $\mathcal{M}$, let $N = p(n)$ and $\Omega = Q \cup \Gamma_\mathcal{M}$. Moreover, define $\Omega_N = \{a(i) \mid a \in \Omega, 0 \leq i \leq N\}$. A configuration $c = \gamma_0 \gamma_1 \cdots \gamma_{i-1} q \gamma_{i+1} \cdots \gamma_N$ of $\mathcal{M}$ is a word of the language $\bigcup_{0 \leq i \leq N-1} \Gamma_\mathcal{M}^i Q \Gamma_\mathcal{M}^{N-i}$. We will encode c by the string

$$\gamma_0(0)\gamma_1(1) \cdots \gamma_{i-1}(i-1) q(i) \gamma_{i+1}(i+1) \cdots \gamma_N(N)$$

on the pushdown. In the following, we will confuse the notion of a configuration and its encoding – what we mean should be clear by from the context. We set $\Gamma = \Omega_N \cup \mathrm{Dir}$. Let us define the transitions Δ of our BPA $\mathcal{X} = (\Gamma, \Delta)$ and implicitly give the set Σ. First, we want $\mathcal{X}$ to be able to push the initial configuration $q_0(0)w_1(1)w_2(2) \cdots w_n(n)\Box(n+1) \cdots \Box(N)$ on the stack:

$$\varepsilon \xrightarrow{\text{input}}_\mathcal{X} q_0(0) w_1(1) \cdots w_n(n) \Box(n+1) \cdots \Box(N)$$

Moreover Δ consists of the following transitions:

$$
\begin{aligned}
\varepsilon &\xrightarrow{a}_{\mathcal{X}} a && \text{for every } a \in \Omega\\
d &\xrightarrow{\text{push}_N}_{\mathcal{X}} a(N)d && \text{for every } d \in \text{Dir and } a \in \Omega\\
a(i) &\xrightarrow{\text{push}}_{\mathcal{X}} a'(i-1)a(i) && \text{for every } 0 < i \leq N \text{ and } a, a' \in \Omega\\
a(0) &\xrightarrow{\text{push}_0}_{\mathcal{X}} a(0) && \text{for every } a \in \Omega\\
a(0) &\xrightarrow{\text{pop}_0}_{\mathcal{X}} \varepsilon && \text{for every } a \in \Omega\\
a(N) &\xrightarrow{\text{pop}_N}_{\mathcal{X}} \varepsilon && \text{for every } a \in \Omega\\
a(i) &\xrightarrow{\bar{a}}_{\mathcal{X}} \varepsilon && \text{for every } 0 \leq i \leq N \text{ and } a \in \Omega\\
d &\xrightarrow{\bar{d}}_{\mathcal{X}} \varepsilon && \text{for every } d \in \text{Dir}\\
\varepsilon &\xrightarrow{\text{loop}}_{\mathcal{X}} \varepsilon && \\
a(i) &\xrightarrow{\text{copy}}_{\mathcal{X}} a(i)a(i) && \text{for every } 0 \leq i \leq N \text{ and } a \in \Omega\\
a(i) &\xrightarrow{\text{shift}}_{\mathcal{X}} a(i)x && \text{for every } 0 \leq i \leq N, x \in \Gamma, \text{ and } a \in \Omega\\
a(i) &\xrightarrow{a/b}_{\mathcal{X}} b(i) && \text{for every } 0 \leq i \leq N \text{ and } a, b \in \Omega
\end{aligned}
$$

In the rest of the proof, we define the formula φ. We will need several auxiliary programs. For any subset $X \subseteq \Omega$ define $\text{pop}_X = \bigcup_{x \in X} \overline{x}$ and $\text{push}_X = \bigcup_{x \in X} x$. The program $\overline{\text{conf}}$ pops the top configuration from the stack:

$$\overline{\text{conf}} \quad = \quad \text{pop}_0 \circ \text{pop}_\Omega^* \circ \text{pop}_N$$

On the other hand, the program conf pushs a potential configuration, more precisely a string of the form $\omega_0(0)\omega_1 \cdots \omega_N(N)$, where each $\omega_i \in \Omega$, on the stack:

$$\text{conf} \quad = \quad \text{push}_N \circ \text{push}^* \circ \text{push}_0$$

Let $\mu \in$ Moves. Our next goal is to define a program check_μ which is executable only if cdc' is on top of the stack, where (i) c and c' are configurations of $\mathcal{M}$, (ii) $d \in$ Dir, and (iii) $c' \vdash^\mu_{\mathcal{M}} c$. We restrict ourselves to the case where $\mu = (q', \gamma', \leftarrow)$. In order to define check_μ, we will need, for every $a, b \in \Omega$, a program $\text{test}_{a,b}$. Assume that the top of the stack is a certain suffix of a configuration of $\mathcal{M}$ followed bt a direction marker $d \in$ Dir and a complete configuration of $\mathcal{M}$, i.e. the top of the stack has the following shape:

$$\omega_k(k)\omega_{k+1}(k+1) \cdots \omega_N(N) d \omega'_0(0) \cdots \omega'_N(N)$$

i.e. each $\omega_i \in \Omega$. We want the program $\mathrm{test}_{a,b}$ to be executable only if $\omega_k = a$ and $\omega'_k = b$. Furthermore, we want to pop $\omega_k(k)$. This can be achieved as follows:

$$\mathrm{test}_{a,b} \quad = \quad a/b \circ [(\mathrm{pop}_\Omega^* \circ \mathrm{pop}_{\mathrm{Dir}} \circ \mathrm{pop}_\Omega^* \circ \mathrm{copy} \circ \mathrm{shift}^*) \cap \mathrm{loop}] \circ \overline{b}$$

Now, we can give check_μ.

$$\begin{aligned}\mathrm{check}_\mu \quad = \quad & \mathrm{loop} \cap \\ & \left[\left(\bigcup_{a \in \Omega} \mathrm{test}_{a,a}\right)^* \circ \bigcup_{q \in Q, \gamma, \gamma'' \in \Gamma_{\mathcal{M}}} (\mathrm{test}_{q',\gamma''} \circ \mathrm{test}_{\gamma'',q} \circ \mathrm{test}_{\gamma',\gamma}) \circ \right. \\ & \left. \left(\bigcup_{a \in \Omega} \mathrm{test}_{a,a}\right)^* \circ \mathrm{pop}_{\mathrm{Dir}} \circ \mathrm{push}_\Omega^* \right]\end{aligned}$$

Let $q \in Q$ and $\gamma \in \Gamma_{\mathcal{M}}$. The program $\mathrm{test}_{q,\gamma}$ tests if the top configuration is in current state q and scans the symbol γ:

$$\mathrm{test}_{q,\gamma} \quad = \quad \mathrm{loop} \cap (\mathrm{pop}_\Omega^* \circ \overline{q} \circ \overline{\gamma} \circ \mathrm{push}_\Omega^*)$$

The program traverse will be only executable if it is possible to go along one edge of the computation tree of $\mathcal{M}$.

$$\begin{aligned}\mathrm{traverse} \quad = \quad & \bigcup_{\substack{q \in Q_{\mathrm{acc}} \\ \gamma \in \Gamma_{\mathcal{M}}}} \mathrm{test}_{q,\gamma} \circ \overline{\mathrm{conf}} \\ \cup \quad & \bigcup_{R(\mu_1,\mu_2) \in \mathrm{Dir}_\forall} \overline{R(\mu_1,\mu_2)} \circ \overline{\mathrm{conf}} \\ \cup \quad & \bigcup_{E(\mu) \in \mathrm{Dir}_\exists} \overline{E(\mu)} \circ \overline{\mathrm{conf}} \\ \cup \quad & \bigcup_{(\mu_1,\mu_2) \in \delta(Q_\forall, \Gamma_{\mathcal{M}})} \overline{L(\mu_1,\mu_2)} \circ R(\mu_1,\mu_2) \circ \mathrm{conf} \circ \mathrm{check}_{\mu_2} \\ \cup \quad & \bigcup_{\substack{q \in Q_\forall, \gamma \in \Gamma_{\mathcal{M}} \\ \delta(q,\gamma) = (\mu_1,\mu_2)}} \mathrm{test}_{q,\gamma} \circ L(\mu_1,\mu_2) \circ \mathrm{conf} \circ \mathrm{check}_{\mu_1} \\ \cup \quad & \bigcup_{\substack{q \in Q_\exists, \gamma \in \Gamma_{\mathcal{M}} \\ \delta(q,\gamma) = (\mu_1,\mu_2), \mu \in \{\mu_1,\mu_2\}}} \mathrm{test}_{q,\gamma} \circ E(\mu) \circ \mathrm{conf} \circ \mathrm{check}_\mu\end{aligned}$$

Our final formula φ is defined as follows:

$$\varphi \quad = \quad \langle \mathrm{input} \circ \mathrm{traverse}^* \rangle \neg \langle \mathrm{pop}_\Omega \rangle \mathtt{true}$$

Eventually, we have:

$$w \in L(\mathcal{M}) \quad \Leftrightarrow \quad (\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$$

$\square$

Hence, we get an EXP lower bound for the data complexity of test-free $\mathrm{PDL}^{\cap}$ over basic parallel processes. This lower bound increases to 2EXP for some fixed BPA if a (test-free) $\mathrm{PDL}^{\cap}$ formula is now the input.

Theorem 9.2. *There exists a fixed BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ such that the following problem is 2EXP-hard:*
INPUT: A test-free $PDL^{\cap}$-formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$?

Proof. Let $\mathcal{M} = (Q, \Sigma_{\mathcal{M}}, \Gamma_{\mathcal{M}}, q_0, \delta, \Box)$ be a fixed $2^{p(m)} - 1$ space bounded ATM for some polynomial p with a 2EXP-hard acceptance problem. Moreover, we assume that $\mathcal{M}$ fulfills the conventions of Subsection 6. Let $w = w_1 w_2 \cdots w_n \in \Sigma_{\mathcal{M}}^*$ be an input of length n. We give a BPA $\mathcal{X} = \mathcal{X}(\mathcal{M}) = (\Gamma, \Delta)$ over $\Sigma = \Sigma(\mathcal{M})$ and a logarithmic space computable test-free $\mathrm{PDL}^{\cap}$-formula $\varphi = \varphi(w, \mathcal{M})$ over Σ such that we have:

$$w \in L(\mathcal{M}) \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$$

Let $\mathrm{Moves} = Q \times \Gamma_{\mathcal{M}} \times \{\leftarrow, \rightarrow\}$ be the set of moves of $\mathcal{M}$, $\mathrm{Dir} = \mathrm{Dir}_{\exists} \cup \mathrm{Dir}_{\forall}$ be the set of universal and existential direction markers of $\mathcal{M}$, 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 the configuration $c = \gamma_0 \gamma_1 \cdots \gamma_{i-1} q \gamma_{i+1} \cdots \gamma_{2^N - 1}$ by:

$$\gamma_0[0]\gamma_1[1] \cdots \gamma_{i-1}[i-1] q[i] \gamma_{i+1}[i+1] \cdots \gamma_{2^N-1}[2^N - 1], \tag{1}$$

where $[k]$ denotes the binary representation of k ($0 \le k \le 2^N - 1$) with N bits, i.e., $[k] = \beta_0 \beta_1 \cdots \beta_{N-1}$ with $\beta_j \in \{0,1\}$ and $k = \sum_{j=0}^{N-1} 2^j \cdot \beta_j$. When we speak of a *cell* we mean $\omega[i]$ for an $\omega \in \Omega$ and a $0 \le i \le 2^N - 1$. We put $\Gamma = \Omega \cup \{0,1\} \cup \mathrm{Dir}$ and $\Sigma = \Gamma \cup \overline{\Gamma} \cup \{\mathrm{loop}\}$. We define the transition relation Δ of $\mathcal{X}$ to be the same as the one of Tree_{Γ} (see Example 4.1) and additionally the rule $\varepsilon \xrightarrow{\mathrm{loop}} \varepsilon$. Thus, $\mathcal{K}(\mathcal{X})$ is nothing else than $\mathcal{K}(\mathrm{Tree}_{\Gamma})$ but where in addition each node has a loop transition labeled by "loop". Now we give test-free $\mathrm{PDL}^{\cap}$-programs which are only executable if certain conditions are true.

- The following little auxiliary programs are only executable from nodes $w \in \Gamma^*$ where there is a factorization $w = uv \in \Gamma^*$ such that u is of a certain kind. After execution of these programs, we are in v.
  - $\overline{X} = \bigcup_{x \in X} \overline{x}$ for any $X \subseteq \Gamma$: Pops a single symbol $x \in X$ from the stack for any subset $X \subseteq \Gamma$.
  - $\mathrm{pop}_i = \overline{\{0,1\}}^i$ for all $0 \le i \le N$: Pops i bits from the stack.
  - $\overline{\mathrm{cell}} = \overline{\Omega} \circ \mathrm{pop}_N$: Pops a cell $\omega[i]$ from the stack, where $\omega \in \Omega$.
  - $\overline{\mathrm{cell}_0} = \overline{\Omega} \circ \overline{0}^N$: Pops a cell $\omega[0]$ from the stack, where $\omega \in \Omega$.
  - $\overline{\mathrm{cell}_1} = \overline{\Gamma_{\mathcal{M}}} \circ \overline{1}^N$: Pops a cell $\gamma[2^N - 1]$ from the stack, where $\gamma \in \Gamma_{\mathcal{M}}$.

- For each subset $X \subseteq \Gamma$ we define $X = \bigcup_{x \in X} x$ which pushs a symbol $x \in X$ on top of the stack.

- We give a program $\overline{\text{inc}}$ which is executable only if on top of the stack there is a word of the form $\omega[i]\omega'[i+1]$ for some $\omega, \omega' \in \Omega$ and some $0 \leq i < N-1$. The program $\overline{\text{inc}}$ pops $\omega[i]$ during its execution. In order to define $\overline{\text{inc}}$ we will use the programs $\chi_{j,\beta}$ for all $0 \leq j \leq N-1$ and $\beta \in \{0,1\}$ 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:

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

We define $\overline{\text{inc}}$ as follows:

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

- We give a program $\overline{\text{conf}}$ that is only executable if the top of the stack is a legal configuration in the sense of (1), i.e., a word of the form $\omega_0[0]\omega_1[1] \cdots \omega_{2^N-1}[2^N-1]$ must be on top of the stack, 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:

$$\begin{aligned}
\overline{\text{conf}} \quad = \quad & (\overline{\text{cell}_0} \circ \overline{\text{cell}}^*) \cap (\overline{\text{inc}}^* \circ \overline{\text{cell}_1}) \\
& \cap \ (\overline{\Gamma_{\mathcal{M}} \cup \{0,1\}}^* \circ \overline{Q} \circ \overline{\Gamma_{\mathcal{M}} \cup \{0,1\}}^*)
\end{aligned}$$

- We give the program cells that pushes finitely many cells $\omega[i]$ $(0 \leq i \leq 2^N-1,\ \omega \in \Omega)$ on top of the stack:

$$\text{cells} \quad = \quad (\{0,1\}^N \circ \Omega)^*$$

- For all $\mu \in \text{Moves}$ we give programs check_μ which are only executable if cdc' is on top of the stack where: (i) c and c' are configurations of $\mathcal{M}$ in the sense of (1), (ii) $d \in \text{Dir}$ and (iii) $c' \vdash^{\mu}_{\mathcal{M}} c$. We restrict ourselves to the case where $\mu = (q', \gamma', \leftarrow)$. First we define some auxiliary programs:

– For all $0 \leq j \leq N-1$ and $\beta \in \{0,1\}$ let $\sigma_{j,\beta} = \mathrm{pop}_j \circ \overline{\beta} \circ \overline{\{0,1\}}^*$. For all $\omega, \omega' \in \Omega$ the program $\pi_{\omega,\omega'}$ assumes that the top of the stack 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

$$\omega_k[k]\omega_{k+1}[k+1]\cdots\omega_{2^N-1}[2^N-1]\ d\ \omega'_0[0]\omega'_1[1]\cdots\omega'_{2^N-1}[2^N-1]$$

is assumed to be on top of the stack. The program $\pi_{\omega,\omega'}$ simply tests if $\omega_k = \omega$ and $\omega'_k = \omega'$ and during execution of it $\omega_k[k]$ will be popped from the stack:

$$\pi_{\omega,\omega'} \quad = \quad \left(\bigcap_{i=0}^{N-1} \bigcup_{\beta \in \{0,1\}} \overline{\omega} \circ \sigma_{i,\beta} \circ \overline{\mathrm{cell}}^* \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{cell}}^* \circ \overline{\omega'} \circ \sigma_{i,\beta} \right) \circ \Gamma^* \cap \overline{\mathrm{cell}}$$

– The program $\pi_=$ has the same assumptions as $\pi_{\omega,\omega'}$ but checks whether the content of the top cell $\omega[k]$ equals the content of the k-th cell of the subsequent configuration:

$$\pi_= \quad = \quad \bigcup_{\omega \in \Omega} \pi_{\omega,\omega}$$

Finally we give check_μ:

$$\begin{aligned} \mathrm{check}_\mu \quad = \quad \mathrm{loop} \cap \Bigg[& \overline{\mathrm{conf}} \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \ \cap \\ & \left(\pi_=^* \circ \bigcup_{q \in Q, \gamma, \gamma'' \in \Gamma_{\mathcal{M}}} (\pi_{q',\gamma''} \circ \pi_{\gamma'',q} \circ \pi_{\gamma',\gamma}) \circ \pi_=^* \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \right) \Bigg] \circ \Gamma^* \end{aligned}$$

- The program $\mathrm{test}_{q,\gamma}$ tests if the top configuration is in current state $q \in Q_\exists \cup Q_\forall$ and scans the symbol $\gamma \in \Gamma_{\mathcal{M}}$:

$$\mathrm{test}_{q,\gamma} \quad = \quad \overline{\mathrm{cell}}^* \circ \overline{q} \circ \mathrm{pop}_N \circ \overline{\gamma} \circ \Gamma^* \cap \mathrm{loop}$$

- The program traverse is only executable if we can go along one edge of the computation tree of $\mathcal{M}$ (Q_{acc} is the set of accepting states of $\mathcal{M}$):

$$\begin{aligned}
\text{traverse} \;=\; & \bigcup_{\substack{q\in Q_{\text{acc}}\\ \gamma\in\Gamma_{\mathcal{M}}}} \text{test}_{q,\gamma} \circ \overline{\text{conf}} \\
\cup & \bigcup_{R(\mu_1,\mu_2)\in \text{Dir}_\forall} \overline{R(\mu_1,\mu_2)} \circ \overline{\text{conf}} \\
\cup & \bigcup_{E(\mu)\in \text{Dir}_\exists} \overline{E(\mu)} \circ \overline{\text{conf}} \\
\cup & \bigcup_{(\mu_1,\mu_2)\in\delta(Q_\forall,\Gamma_{\mathcal{M}})} \overline{L(\mu_1,\mu_2)} \circ R(\mu_1,\mu_2) \circ \text{cells} \circ \text{check}_{\mu_2} \\
\cup & \bigcup_{\substack{q\in Q_\forall,\gamma\in\Gamma_{\mathcal{M}}\\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \text{test}_{q,\gamma} \circ L(\mu_1,\mu_2) \circ \text{cells} \circ \text{check}_{\mu_1} \\
\cup & \bigcup_{\substack{q\in Q_\exists,\gamma\in\Gamma_{\mathcal{M}}\\ \delta(q,\gamma)=(\mu_1,\mu_2),\mu\in\{\mu_1,\mu_2\}}} \text{test}_{q,\gamma} \circ E(\mu) \circ \text{cells} \circ \text{check}_{\mu}
\end{aligned}$$

- The program check_w checks whether the initial configuration $q_0[0]w_1[1]w_2[2]\cdots w_n[n]\Box[n+1]\cdots\Box[2^N-1]$ is a prefix of the content of the stack:

$$\text{check}_w \;=\; \left(\overline{\text{conf}} \cap \left(\overline{q_0} \circ \text{pop}_N \circ \bigcirc_{i=1}^{n}(\overline{w_i} \circ \text{pop}_N) \circ (\overline{\Box} \circ \text{pop}_N)^*\right)\right) \circ \Gamma^* \cap \text{loop}$$

- We give the program final:

$$\text{final} \;=\; \text{cells} \circ \text{check}_w \circ \text{traverse}^*$$

Our final formula φ, for which we have $\varepsilon \in [\![\varphi]\!]_{\mathcal{K}(\mathcal{X})} \Leftrightarrow w \in L(\mathcal{M})$, is:

$$\varphi = \langle\text{final}\rangle\neg\langle\overline{\Gamma}\rangle\texttt{true}$$

Eventually, we have:

$$w \in L(\mathcal{M}) \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{X}),\varepsilon) \models \varphi$$

□

9.2 Upper bounds

9.2.1 Two-way alternating tree automata

Walukiewicz first proved that model-checking the modal μ-calculus over pushdown systems is EXP-complete [31]. Cachat extended this result to prefix-recognizable systems [4]. A different approach is the usage of two-way alternating tree automata introduced by Vardi [29], as for example in [16, 15]. The idea is to reduce the model-checking problem for a pushdown system or a prefix-recognizable system to the emptiness problem of two-way alternating tree automata.

Let Γ be a finite *alphabet* and $u \in \Gamma^*$ a word. A Γ-*tree* is a suffix-closed subset $T \subseteq \Gamma^*$, i.e., for all $w \in \Gamma^*, a \in \Gamma$ with $aw \in T$ we have $w \in T$. We call the elements of T *nodes* and $\varepsilon \in T$ is the *root* of T. A node $u \in T$ is called *leaf* if and only if $au \notin T$ for all $a \in \Gamma$. An *infinite path* of a tree T is an infinite sequence $u_0 u_1 \cdots \in T^\omega$ such that $u_0 = \varepsilon$ is the root of T and for all $i \geq 0$ we have $u_{i+1} = au_i$ for some $a \in \Gamma$. Let Σ and Γ be finite alphabets. We call a pair (T, λ) a Σ-*labeled* Γ-*tree* if T is a Γ-tree and $\lambda : T \to \Sigma$ labels every node of T by an element of Σ. The complete Γ-tree is the $\Gamma \uplus \{\perp\}$-labeled Γ-tree $(\Gamma^*, \lambda_\Gamma)$ where $\lambda_\Gamma(\varepsilon) = \perp$ and $\lambda_\Gamma(aw) = a$ for all $a \in \Gamma$ and $w \in \Gamma^*$. For a finite set X we denote by $\mathcal{B}^+(X)$ the set of all *positive boolean formulas over* X. Note that `true` and `false` are positive boolean formulas. A subset $Y \subseteq X$ *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if and only if by assigning `true` to all elements in Y the formula θ is evaluated to `true`. Let $\mathrm{ext}(\Gamma) = \Gamma \uplus \{\varepsilon, \downarrow\}$ denote the *extension of* Γ. Furthermore, we 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 for short) over Γ is a tuple $\mathcal{T} = (Q, \delta, \mathrm{Acc})$, where (i) Q is a finite non-empty *set of states*, (ii) $\delta : Q \times (\Gamma \cup \{\perp\}) \to \mathcal{B}^+(Q \times \mathrm{ext}(\Gamma))$ is the *transition function*, and (iii) $\mathrm{Acc} : Q \to \{0, \ldots, m\}$ is the *priority function* (where $m \in \mathbb{N}$) which assigns to each state an integer between 0 and m. Let $\mathcal{T} = (Q, \delta, \mathrm{Acc})$ be a TWATA over Γ, $u \in \Gamma^*$ a word, and let $q_0 \in Q$ be a state. A (q_0, u)-*run* of $\mathcal{T}$ (over the complete Γ-tree $(\Gamma^*, \lambda_\Gamma)$) is a $(Q \times \Gamma^*)$-labeled Ω-tree $R = (T_R, \lambda_R)$ for some finite set Ω such that the following two conditions are satisfied: (i) $\varepsilon \in T_R$ and $\lambda_R(\varepsilon) = (q_0, u)$ and (ii) if $\alpha \in T_R$ with $\lambda_R(\alpha) = (q, v)$ and $\delta(q, \lambda_\Gamma(v)) = \theta$, then there exists a subset $Y \subseteq Q \times \mathrm{ext}(\Gamma)$ that satisfies the formula θ and for all $(q', e) \in Y$ there exists an $\omega \in \Omega$ with $\omega\alpha \in T_R$ and $\lambda_R(\omega\alpha) = (q', ev)$. We say a (q_0, u)-run $R = (T_R, \lambda_R)$ of $\mathcal{T}$ is *successful* if and only if for every infinite path $w_1, w_2, \ldots$ of T_R, $\min(\{\mathrm{Acc}(q) \mid \lambda_R(w_i) \in \{q\} \times \Gamma^* \text{ for infinitely many } i\})$ is even. This is precisely the parity acceptance condition as in [29]. We also write $[\![\mathcal{T}, q_0]\!]$ for the set $\{u \in \Gamma^* \mid \text{ there exists a successful } (q_0, u)\text{-run of } \mathcal{T}\}$. Let $\mathcal{T} = (Q, \delta, \mathrm{Acc})$ be a TWATA over Γ. The *size* $|\mathrm{Acc}|$ of Acc is defined to be $\max\{\mathrm{Acc}(q) \mid q \in Q\}$. The *size* $|\mathcal{T}|$ of $\mathcal{T}$ is defined as $|\Gamma| + |Q| + \sum_{\theta \in \mathrm{ran}(\delta)} |\theta| + |\mathrm{Acc}|$. Let $\mathcal{T}_1 = (Q_1, \delta_1, \mathrm{Acc}_1)$ and $\mathcal{T}_2 = (Q_2, \delta_2, \mathrm{Acc}_2)$ each be a TWATA over Γ. Then, we define the *disjoint union* $\mathcal{T}_1 \uplus \mathcal{T}_2$ of $\mathcal{T}_1$ and $\mathcal{T}_2$ to be the TWATA $(Q_1 \uplus Q_2, \delta_1 \uplus \delta_2, \mathrm{Acc}_1 \uplus \mathrm{Acc}_2)$.

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 [29, 16, 15], where also runs of TWATAs on arbitrarily labeled trees are considered. The following result follows from [29]:

Theorem 9.3 ([29])**.** *For a given TWATA $\mathcal{T} = (Q, \delta, \mathrm{Acc})$ and a state $q \in Q$, it can be checked in time exponential in $|Q| \cdot |\mathrm{Acc}|$ whether $\varepsilon \in [\![\mathcal{T}, q]\!]$.*

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 the alphabet Γ. A finite automaton A *over* $\mathcal{T}$ is a pair $(Q, \rightarrow_A)$ where Q is a finite state set and $\rightarrow_A$ is a set of transitions of the following form:

- $p \xrightarrow{a}_A q$ for $p, q \in Q, a \in \Gamma \cup \overline{\Gamma} \cup \{\varepsilon\}$, or
- $p \xrightarrow{\mathcal{T},s}_A q$ for $p, q \in Q, s \in S$.

We call the latter transitions the *test-transitions* of A. Note that w.l.o.g. we can assume that ε-transitions $p \xrightarrow{\varepsilon}_A q$ do not occur in A since we can introduce a dummy state s_ε where $\delta(s_\varepsilon, \gamma) = \mathtt{true}$ for all $\gamma \in \Gamma \cup \{\perp\}$ and an arbitrary value for $\mathrm{Acc}(s_\varepsilon)$ and replace $p \xrightarrow{\varepsilon}_A q$ by $p \xrightarrow{\mathcal{T},s_\varepsilon}_A q$. We denote with $A^\downarrow$ (resp. $A^\uparrow$) the finite automaton 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 $\overline{\Gamma} \cup \{\varepsilon\}$ (resp. $\Gamma \cup \{\varepsilon\}$). Let $\mathcal{T}$ be a TWATA over Γ and let $A = (Q, \rightarrow_A)$ be a finite automaton over $\mathcal{T}$. Define the relation $\Rightarrow_A \subseteq (\Gamma^* \times Q) \times (\Gamma^* \times Q)$ as the smallest relation such that:

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

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

We will use a TWATA $\mathcal{T}$ to simulate a formula φ (i.e. $[\![\mathcal{T}, q]\!] = [\![\varphi]\!]$ for some state q of $\mathcal{T}$) and use an NFA A over a TWATA $\mathcal{T}$ to simulate a program π (i.e. $[\![A, p, q]\!] = [\![\pi]\!]$ for some states p, q of A) that uses tests that are simulated by $\mathcal{T}$.

9.2.2 The main construction

We now describe an automata theoretic construction which will be crucial for handling the intersection of programs in our upper bound of $\mathrm{PDL}^\cap$ over prefix-recognizable systems.

Let $\mathcal{T} = (S, \delta, \mathrm{Acc})$ be a TWATA over Γ and let $A = (Q, \rightarrow_A)$ be a finite automaton over $\mathcal{T}$. Define the set $\mathrm{hop}_A \subseteq \Gamma^* \times Q \times Q$ as 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 $u \in [\![\mathcal{T}, s]\!]$ and $p \xrightarrow{\mathcal{T},s}_A q$ for $s \in S$, then $(u, p, q) \in \mathrm{hop}_A$

The idea of the relation hop_A is that $(u, p, q) \in \mathrm{hop}_A$ if and only if A (initially being in state p) can walk through the complete Γ-tree from the node u back to u (and finally being in state q) where on this walk A never visits a node that is a strict suffix of u. Each time we move in the complete Γ-tree from v to av (resp. va to v), we have to read a (resp. $\bar{a}$) in the automaton A. Formally we have:

Lemma 9.4. *We have $(u,p,q) \in \mathrm{hop}_A$ if and only if there exist $n \geq 1$, $u_1,\dots,u_n \in \Gamma^*$, and $q_1,\dots,q_n \in Q$ such that*

(1) $u_1 = u_n = \varepsilon$,

(2) $q_1 = p$, $q_n = q$, *and*

(3) $(u_1u, q_1) \Rightarrow_A (u_2u, q_2) \cdots \Rightarrow_A (u_nu, q_n)$.

Proof. If $(u,p,q) \in \mathrm{hop}_A$, then an induction over a proof tree for the fact $(u,p,q) \in \mathrm{hop}_A$ shows that there exist $n \geq 1$, $u_1,\dots,u_n \in \Gamma^*$, and $q_1,\dots,q_n \in Q$ such that (1)–(3) from the lemma hold. Now assume that there exist $n \geq 1$, $u_1,\dots,u_n \in \Gamma^*$, and $q_1,\dots,q_n \in Q$ such that $u_1 = u_n = \varepsilon$, $q_1 = p$, $q_n = q$, and $(u_1u,q_1) \Rightarrow_A (u_2u,q_2) \cdots \Rightarrow_A (u_nu,q_n)$. We prove by induction over n that $(u,p,q) \in \mathrm{hop}_A$. If $n=1$ then $p=q$ and hence $(u,p,q) \in \mathrm{hop}_A$. If $n=2$, i.e., $(u,p) \Rightarrow_A (u,q)$, then there must exist a test transition $p \xrightarrow{\mathcal{T},s}_A q$ such that $u \in [\![\mathcal{T},s]\!]$. Thus, $(u,p,q) \in \mathrm{hop}_A$. Now assume that $n \geq 3$. We can distinguish the following two cases:

Case 1. There exists $1 < i < n$ such that $u_i = \varepsilon$. Then, by induction we get $(u,p,q_i),(u,q_i,q) \in \mathrm{hop}_A$. Hence, also $(u,p,q) \in \mathrm{hop}_A$.

Case 2. There does not exist $1 < i < n$ such that $u_i = \varepsilon$. Then there must exist $a \in \Gamma$ such that $p \xrightarrow{a}_A q_2$, $q_{n-1} \xrightarrow{\bar{a}}_A q$, and $u_i = v_ia$ for some $v_i \in \Gamma^*$ $(1 < i < n)$. Inductively, we get $(au, q_2, q_{n-1}) \in \mathrm{hop}_A$. Thus, $(u,p,q) \in \mathrm{hop}_A$. ☐

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

Lemma 9.5. *There exists a TWATA $\mathcal{U} = (S', \delta', \mathrm{Acc}')$ over Γ with state set $S' = S \uplus (Q \times Q)$ such that:*

(i) *for every $s \in S$ we have* $[\![\mathcal{U},s]\!] = [\![\mathcal{T},s]\!]$,

(ii) *for every $(p,q) \in Q \times Q$ we have* $[\![\mathcal{U},(p,q)]\!] = \{u \in \Gamma^* \mid (u,p,q) \in \mathrm{hop}_A\}$, *and*

(iii) $|\mathrm{Acc}'| = |\mathrm{Acc}|$.

Proof. For states in S, the transitions of $\mathcal{U}$ are the same as for $\mathcal{T}$. If $q \in Q$ and $\omega \in \Gamma \cup \{\bot\}$, then we introduce the transition

$$\delta'((q,q),\omega) = \mathtt{true}.$$

If $p \neq q$ and $\omega \in \Gamma \cup \{\bot\}$, then we introduce the transition

$$\delta'((p,q),\omega) = \bigvee_{\substack{p',q' \in Q, a \in \Gamma \\ p \xrightarrow{a}_A p', q' \xrightarrow{\bar{a}}_A q}} \langle (p',q'), a\rangle \vee \bigvee_{r \in Q} (\langle (p,r),\varepsilon\rangle \wedge \langle (r,q),\varepsilon\rangle) \vee \bigvee_{\substack{s \in S \\ p \xrightarrow{\mathcal{T},s}_A q}} \langle s, \varepsilon\rangle.$$

We define the priority function Acc' as follows:

$$\mathrm{Acc}'(s') = \begin{cases} \mathrm{Acc}(s') & \text{if } s' \in S \\ 1 & \text{if } s' \in Q \times Q \end{cases}$$

Trivially (iii) holds. We put $\mathrm{Acc}'((p,q)) = 1$ for all $p,q \in Q$ since $\mathcal{U}$ should spend only a finite amount of time on verifying whether $u \in \mathrm{hop}_A$ for a node $u \in \Gamma^*$. From the definition of δ' it is clear that (i) holds. From the definition of hop_A, the construction of δ', and Acc' it follows that $[\![\mathcal{U},(p,q)]\!] = \{u \in \Gamma^* \mid (u,p,q) \in \mathrm{hop}_A\}$, hence (ii) holds. $\square$

Now define a new finite automaton $B = (Q, \rightarrow_B)$ over the TWATA $\mathcal{U}$, that results from A by adding for every pair $(p,q) \in Q \times Q$ the test transition $p \xrightarrow{\mathcal{U},(p,q)} q$.

Lemma 9.6. *Let $u,v \in \Gamma^*$ and $p,q \in Q$. Then the following three statements are equivalent:*

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

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

(3) 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 $(w,v) \in [\![B^{\uparrow},r,q]\!]$

Proof. The implication (3) $\Rightarrow$ (2) is trivial. For (2) $\Rightarrow$ (1) note that for every test-transition $p' \xrightarrow{\mathcal{U},(p',q')}_B q'$ of B and every $x \in [\![\mathcal{U},(p',q')]\!]$ we have $(x,p',q') \in \mathrm{hop}_A$ and hence $(x,p') \Rightarrow_A^* (x,q')$. It remains to prove the implication (1) $\Rightarrow$ (3). Assume $(u,v) \in [\![A,p,q]\!]$. Then there exist words $w_0,\ldots,w_n \in \Gamma^*$ and states $q_0,\ldots,q_n \in Q$ such that

$$(u,p) = (w_0,q_0) \Rightarrow_A (w_1,q_1) \Rightarrow_A \cdots \Rightarrow_A (w_n,q_n) = (v,q).$$

Let $\mu = \min\{|w_j| \mid 0 \le j \le n\}$ denote the length of the shortest word of all the w_j. Let $d = |u| - \mu$, $e = |v| - \mu$, $i_l = \min\{j : |w_j| = \mu + l\}$ (for all $0 \le l \le d$) and $j_l = \max\{j : |w_j| = \mu + l\}$ (for all $0 \le l \le e$). Note that i_l and j_l indeed exist. Let $x_l = w_{i_l}$ for $0 \le l \le d$. Clearly, for all $0 < l \le d$ we have $x_l = w_{i_l} = w_{i_{(l-1)}-1}$. By definining $r_l = q_{i_l}$ and $r'_l = q_{i_{l-1}-1}$ we have $(x_l, r_l, r'_l) \in \mathrm{hop}_A$, hence by Lemma 9.5 we have $x_l \in [\![\mathcal{U},(r_l,r'_l))]\!]$. Moreover, there exists $a_l \in \Gamma$ such that $(x_l, r'_l) \Rightarrow_A (x_{l-1}, r_{l-1})$ where $x_l = a_l x_{l-1}$, thus $(x_l, r_l) \Rightarrow_{B^{\downarrow}}^* (x_{l-1}, r_{l-1})$. Analogously, let $y_l = w_{j_l}$ for $0 \le l \le e$. We have $y_l = w_{j_{l-1}+1} = w_{j_l}$. By definining $s'_l = q_{j_{l-1}+1}$ and $s_l = q_{j_l}$ we have $(y_l, s'_l, s_l) \in \mathrm{hop}_A$. Hence by Lemma 9.5 we have $y_l \in [\![\mathcal{U},(s'_l,s_l)]\!]$. Moreover, there exists $b_l \in \Gamma$ such that $(y_{l-1}, s_{l-1}) \Rightarrow_A (y_l, s'_l)$ where $y_l = b_l y_{l-1}$. Thus $(y_{l-1}, s_{l-1}) \Rightarrow_{B^{\uparrow}}^* (y_l, s_l)$. Clearly, we have $x_0 = y_0$ and $(x_0, r_0, s_0) \in \mathrm{hop}_A$. Hence by Lemma 9.5 we have $x_0 \in [\![\mathcal{U},(r_0,s_0)]\!]$. Altogether, we get:

$$\begin{gathered}(u,p) = (w_0,q_0) = (x_d,r_d) \Rightarrow_{B^{\downarrow}}^* (x_{d-1},r_{d-1}) \\ \cdots \Rightarrow_{B^{\downarrow}}^* (x_0,r_0) \Rightarrow_{B^{\uparrow}} (y_0,s_0) \Rightarrow_{B^{\uparrow}}^* (y_1,s_1) \Rightarrow_{B^{\uparrow}}^* \\ \cdots (y_{e-1},s_{e-1}) \Rightarrow_{B^{\uparrow}}^* (y_e,s_e) = (w_n,q_n) = (v,q)\end{gathered}$$

Obviously, x_0 is a common suffix of u and v, since $u = a_d a_{d-1} \cdots a_1 x_0$ and $v = b_e b_{e-1} \cdots b_1 x_0$. Finally we have $(u,x_0) \in [\![B^{\downarrow},p,r_0]\!]$ and $(x_0,v) \in [\![B^{\uparrow},r_0,q]\!]$. This completes the proof of the lemma. $\square$

We define the set $\mathrm{drop}_B \subseteq \Gamma^* \times Q \times Q$ as the smallest set such that:

- for all $u \in \Gamma^*$ and $p \in Q$, $(u, p, p) \in \mathrm{drop}_B$

- 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 $u \in [\![\mathcal{U}, s']\!]$, $p \xrightarrow{\mathcal{U},s'}_B r$ for $s' \in S'$, and $(u, r, q) \in \mathrm{drop}_B$, then $(u, p, q) \in \mathrm{drop}_B$

- if $u \in [\![\mathcal{U}, s']\!]$, $r \xrightarrow{\mathcal{U},s'}_B q$ for $s' \in S'$, and $(u, p, r) \in \mathrm{drop}_B$, then $(u, p, q) \in \mathrm{drop}_B$

The idea of the relation drop_B is that $(u, p, q) \in \mathrm{drop}_B$ if and only if B (initially being in state p) can walk through the complete Γ-tree from the node u back to u (and finally being in state q). During this walk, B first makes only test-transitions and pops; in a second phase B only makes test-transitions and pushes. Each time we move in the complete Γ-tree from v to av (resp. from va to v), we have to read a (resp. $\bar{a}$) in the automaton B. Formally we have:

Lemma 9.7. *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 $(v, u) \in [\![B^{\uparrow}, r, q]\!]$.*

Proof. If $(u, p, q) \in \mathrm{drop}_B$, then an induction over a proof tree for the fact $(u, p, q) \in \mathrm{drop}_B$ shows that there exist $r \in Q$ and a suffix v of u such that $(u, v) \in [\![B^{\downarrow}, p, r]\!]$ and $(v, u) \in [\![B^{\uparrow}, r, q]\!]$. Now assume that $(u, v) \in [\![B^{\downarrow}, p, r]\!]$ and $(v, u) \in [\![B^{\uparrow}, r, q]\!]$ for some suffix v of u and some state $r \in Q$. There exist $m, n \geq 0$ such that $(u, p) \Rightarrow^m_{B\downarrow} (v, r) \Rightarrow^n_{B\uparrow} (u, q)$. By induction on $m + n$ we prove $(u, p, q) \in \mathrm{drop}_B$. If $m + n = 0$, then $p = q$ and hence, $(u, p, q) \in \mathrm{drop}_B$. Now assume that $m > 0$ or $n > 0$. We can distinguish the following three cases:

Case 1. $(u, p) \Rightarrow_{B\downarrow} (u, p') \Rightarrow^{m-1}_{B\downarrow} (v, r)$. Thus, there exists a test-transition $p \xrightarrow{\mathcal{U},s'}_B p'$ such that $u \in [\![\mathcal{U}, s']\!]$. By induction we get $(u, p', r) \in \mathrm{drop}_B$. Hence, $(u, p, q) \in \mathrm{drop}_B$.

Case 2. $(v, r) \Rightarrow^{n-1}_{B\uparrow} (u, q') \Rightarrow_{B\uparrow} (u, q)$. This case can be treated analogously to case 1.

Case 3. $u = au'$ for some $a \in \Gamma$ and $u' \in \Gamma^*$ such that $(u, p) \Rightarrow_{B\downarrow} (u', p') \Rightarrow^{m-1}_{B\downarrow} (v, r) \Rightarrow^{n-1}_{B\uparrow} (u', q') \Rightarrow_{B\uparrow} (u, q)$. Then we must have $p \xrightarrow{\bar{a}}_B p'$ and $q' \xrightarrow{a}_B q$. Moreover, by induction we have $(u', p', q') \in \mathrm{drop}_B$. Hence, we get $(u, p, q) \in \mathrm{drop}_B$. $\square$

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

Lemma 9.8. *There exists a TWATA $\mathcal{V} = (S'', \delta'', \mathrm{Acc}'')$ with state set $S'' = S' \uplus (Q \times Q)$ such that:* [2]

(i) for every state $s' \in S'$ of $\mathcal{U}$ we have $[\![\mathcal{V}, s']\!] = [\![\mathcal{U}, s']\!]$,

(ii) for every state $(p, q) \in Q \times Q$ we have $[\![\mathcal{V}, (p, q)]\!] = \{u \in \Gamma^* \mid (u, p, q) \in \mathrm{drop}_B\}$, and

(iii) $|\mathrm{Acc}''| = |\mathrm{Acc}|$.

[2]Note that S'' contains two disjoint copies of $Q \times Q$. In the rest of this subsection, when writing (p, q) for $p, q \in Q$, we implictly assume that $(p, q) \in S'' \setminus S'$.

Proof. For states of $\mathcal{U}$, the transitions of $\mathcal{V}$ are the same as for $\mathcal{U}$. Now let $(p,q) \in Q \times Q$ be a state of $\mathcal{V}$, which does not belong to S'. If $p = q$ and $\omega \in \Gamma \cup \{\bot\}$, then we introduce the transition

$$\delta''((q,q),\omega) = \texttt{true}.$$

If $p \neq q$ and $a \in \Gamma$, then we introduce the transition

$$\delta''((p,q),a) = \bigvee_{\substack{p',q' \in Q \\ p \xrightarrow{\bar{a}}_B p', q' \xrightarrow{a}_B q}} \langle (p',q'), \downarrow \rangle \vee \bigvee_{\substack{s' \in S', r \in Q \\ p \xrightarrow{\mathcal{U},s'}_B r}} (\langle s', \varepsilon\rangle \wedge \langle (r,q), \varepsilon \rangle) \vee \bigvee_{\substack{s' \in S', r \in Q \\ r \xrightarrow{\mathcal{U},s'}_B q}} (\langle s', \varepsilon\rangle \wedge \langle (p,r), \varepsilon \rangle).$$

If $p \neq q$, then we introduce the transition

$$\delta''((p,q),\bot) = \bigvee_{\substack{s' \in S', r \in Q \\ p \xrightarrow{\mathcal{U},s'}_B r}} (\langle s', \varepsilon\rangle \wedge \langle (r,q), \varepsilon \rangle) \vee \bigvee_{\substack{s' \in S', r \in Q \\ r \xrightarrow{\mathcal{U},s'}_B q}} (\langle s', \varepsilon\rangle \wedge \langle (p,r), \varepsilon \rangle).$$

The priority function Acc'' is defined as follows:

$$\mathrm{Acc}''(s'') = \begin{cases} \mathrm{Acc}'(s'') & \text{if } s'' \in S' \\ 1 & \text{if } s'' \in Q \times Q \end{cases}$$

Trivially (iii) holds. From the definition of δ'' it is clear that (i) holds. From the definition of drop_B and the construction of δ'' and Acc'' it follows that $[\![\mathcal{V},(p,q)]\!] = \{u \in \Gamma^* \mid (u,p,q) \in \mathrm{drop}_B\}$, hence (ii) holds. $\square$

Let $C = (Q, \to_C)$ be the finite automaton over the TWATA $\mathcal{V}$, that results from B by adding for every pair $(p,q) \in Q \times Q$ the test transition $p \xrightarrow{\mathcal{V},(p,q)} q$. Note that the state set of A, B and C is Q. For two words $u, v \in \Gamma^*$ we denote with $\inf(u,v)$ the longest common suffix of u and v.

Lemma 9.9. *Let $u, v \in \Gamma^*$ and $p, q \in Q$. Then the following three statements are equivalent:*

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

(2) $(u,v) \in [\![C,p,q]\!]$

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

Proof. The implications (1) $\Rightarrow$ (2) and (3) $\Rightarrow$ (2) are trivial. For (2) $\Rightarrow$ (1) note that for every test-transition $p' \xrightarrow{\mathcal{V},(p',q')} q'$ of C and every $x \in [\![\mathcal{V},(p',q')]\!]$ we have $(x,p',q') \in \mathrm{drop}_B$ and hence $(x,p') \Rightarrow_B^* (x,q')$. Thus, $(u,p) \Rightarrow_C^* (v,q)$ implies $(u,p) \Rightarrow_B^* (v,q)$, and hence (by Lemma 9.6) $(u,p) \Rightarrow_A^* (v,q)$. It remains to show (1) $\Rightarrow$ (3). Thus, assume that $(u,v) \in [\![A,p,q]\!]$. By Lemma 9.6 we have $(u,p) \Rightarrow_{B^{\downarrow}}^* (w,r) \Rightarrow_{B^{\uparrow}}^* (u,q)$ for some state $r \in Q$ and some common suffix w of u

and v. Since $\inf(u,v)$ is the longest common suffix of u and v, there must exist $p', q' \in Q$ such that

$$(u,p) \Rightarrow^*_{B\downarrow} (\inf(u,v), p') \Rightarrow^*_{B\downarrow} (w,r) \Rightarrow^*_{B\uparrow} (\inf(u,v), q') \Rightarrow^*_{B\uparrow} (u,q).$$

By Lemma 9.7 we have $(\inf(u,v), p', q') \in \mathrm{drop}_B$, i.e., $\inf(u,v) \in [\![\mathcal{V}, (p',q')]\!]$ by Lemma 9.8. Hence, we get

$$(u,p) \Rightarrow^*_{C\downarrow} (\inf(u,v), p') \Rightarrow_C (\inf(u,v), q') \Rightarrow^*_{C\uparrow} (u,q).$$

This implies (3). $\square$

9.2.3 A 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 will 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 $\mathcal{T}$ grows exponentially in the size of the formula φ and polynomially in the size of $\mathcal{Z}$. The size of the priority function Acc will be linear in the size of φ. This proves a 2EXP upper bound by Theorem 9.3. From now on any occurring TWATA will implicitly be over Γ and the size of the priority function will at least be 1. The construction of $\mathcal{T}$ will be done inductively over the structure of the formula φ. More precisely, for every subformula ψ of φ we will construct a TWATA $\mathcal{T}(\psi)$ together with a state s of $\mathcal{T}(\psi)$ such that $[\![\psi]\!] = [\![\mathcal{T}(\psi), s]\!]$. For every program π that occurs in φ we construct a TWATA $\mathcal{T}(\pi)$ and a finite automaton $A(\pi)$ over $\mathcal{T}(\pi)$ such that $[\![\pi]\!] = [\![A(\pi), p, q]\!]$ for two states p and q of $A(\pi)$.

The case $\psi = \mathtt{true}$ is clear, the case $\psi = \psi_1 \vee \psi_2$ can be skipped since $[\![\psi_1 \vee \psi_2]\!] = [\![\neg(\neg\psi_1 \wedge \neg\psi_2)]\!] = [\![\neg\langle(\neg\psi_1)?\rangle\neg\psi_2]\!]$. If $\psi = \neg\theta$, then we apply the standard complementation procedure, see e.g. [24], 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. When our subformula ψ is of the form $\langle\pi\rangle\theta$ for an PDL$^\cap$ program π and a subformula θ we have inductively already constructed $A = A(\pi)$ with state set Q and $\mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$ such that $[\![\pi]\!] = [\![A(\pi), p, q]\!]$ for two states $p, q \in Q$. Too, we have inductively already constructed $\mathcal{T}(\theta) = (S_2, \delta_2, \mathrm{Acc}_2)$ such that $[\![\theta]\!] = [\![\mathcal{T}(\theta), s_2]\!]$ for some state $s_2 \in S_2$. We define the TWATA $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$ with $S = Q \uplus S_1 \uplus S_2$. For states in S_1 or in S_2 the transitions of $\mathcal{T}(\psi)$ are the same as for $\mathcal{T}(\pi)$ or $\mathcal{T}(\theta)$, respectively. For states $q' \in Q$ and $a \in \Gamma$ we define:

$$\begin{aligned} \delta(q', a) \quad = \quad & \bigvee_{\substack{q'' \in Q \\ q' \xrightarrow{\overline{a}}_A q''}} \langle q'', \downarrow\rangle \vee \bigvee_{\substack{q'' \in Q, b \in \Gamma \\ q' \xrightarrow{b}_A q''}} \langle q'', b\rangle \vee \bigvee_{\substack{q'' \in Q \\ q' \xrightarrow{\varepsilon}_A q''}} \langle q'', \varepsilon\rangle \vee \\ & \bigvee_{\substack{q'' \in Q, r \in S_1 \\ q' \xrightarrow{\mathcal{T}, r}_A q''}} (\langle r, \varepsilon\rangle \wedge \langle q'', \varepsilon\rangle) \vee ((q' = q) \wedge \langle s_2, \varepsilon\rangle). \end{aligned}$$

and for $q' \in Q$ we define

$$\delta(q', \bot) = \bigvee_{\substack{q'' \in Q, b \in \Gamma \\ q' \xrightarrow{b}_A q''}} \langle q'', b\rangle \vee \bigvee_{\substack{q'' \in Q \\ q' \xrightarrow{\varepsilon}_A q''}} \langle q'', \varepsilon\rangle \vee \bigvee_{\substack{q'' \in Q, r \in S_1 \\ q' \xrightarrow{\mathcal{T}, r}_A q''}} (\langle r, \varepsilon\rangle \wedge \langle q'', \varepsilon\rangle) \vee ((q' = q) \wedge \langle s_2, \varepsilon\rangle).$$

The priority function Acc is defined as follows:

$$\mathrm{Acc}(s) = \begin{cases} 1 & \text{if } s \in Q \\ \mathrm{Acc}_1(s) & \text{if } s \in S_1 \\ \mathrm{Acc}_2(s) & \text{if } s \in S_2 \end{cases}$$

We put $\mathrm{Acc}(s) = 1$ for all $s \in Q$ since we want to assure that the automaton $A(\pi)$ is simulated for a finite time only, since ψ is a box formula with existential modality. We obtain $[\![\psi]\!] = [\![\mathcal{T}(\psi), p]\!]$.

Let us now describe the construction of $A(\pi)$ and $\mathcal{T}(\pi)$ for each occurring PDL$^\cap$ program π.

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 $p \xrightarrow{\mathcal{T},r} q$. Hence, we have $[\![\pi]\!] = [\![A(\pi), p, q]\!] = \{(u, u) \mid u \in [\![\mathcal{T}(\psi), r]\!]\}$.

Case $\pi = \sigma \in \Sigma$: Assume that $\alpha(\sigma) = \bigcup_{i=1}^{n_\sigma}(L(\mathcal{A}_i^\sigma) \times L(\mathcal{B}_i^\sigma))L(\mathcal{C}_i^\sigma)$. Let the homomorphism $h : \Gamma \to \overline{\Gamma}$ with $h(a) = \overline{a}$ for all $a \in \Gamma$ be given. From the representation of $\alpha(\sigma)$ we can easily construct finite automata A_i, B_i, C_i such that $L(A_i) = h(L(\mathcal{A}_i^\sigma))$, $L(B_i) = L(\mathcal{B}_i^\sigma)^{\mathrm{rev}}$, and $L(C_i) = h(L(\mathcal{C}_i^\sigma))$. Let $C_i = (Q_i, \overline{\Gamma}, q_i, F_i, \to_{C_i})$. First, we give the TWATA $\mathcal{T}(\pi) = (S, \delta, \mathrm{Acc})$ over Γ where $S = \biguplus_{i=1}^{n_\sigma} Q_i$ and $\mathrm{Acc}(s) = 1$ for all $s \in S$. The transition function δ is defined as follows, where $a \in \Gamma$, $s_i \in S_i$ $(1 \leq i \leq n_\sigma)$:

$$\delta(s_i, a) = \bigvee_{\substack{s_i' \in S_i \\ s_i' \in \delta_i(s_i, \overline{a})}} (s_i', \downarrow) \vee \bigvee_{\substack{s_i' \in S_i \\ s_i' \in \delta_i(s_i, \varepsilon)}} (s_i', \varepsilon)$$

Moreover, for $s_i \in S_i$ $(1 \leq i \leq n_\sigma)$, we define:

$$\delta(s_i, \bot) = \bigvee_{\substack{s_i' \in S_i \\ s_i' \in \delta_i(s_i, \varepsilon)}} (s_i', \varepsilon) \vee (s_i \in F_i)$$

The automaton $A(\pi)$ results from the disjoint union of all automata A_i and B_i, by adding two fresh states p, q. There is an ε-transition from p to all initial states of all A_i and an ε-transition from all final states of all B_i to q. There is a test transition labeled by $\mathcal{T}(\pi), q_i$

between any final state of A_i and any initial state of B_i in $A(\pi)$ for all $1 \leq i \leq n_\sigma$. Clearly $[\![\pi]\!] = [\![A(\pi), p, q]\!]$.

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

It remains to construct $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$. For this, we use the construction of Section 9.2.2:

Constructing $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$

Assume that the finite automata $A(\pi_i) = (Q_i, \rightarrow_{A(\pi_i)})$, where $A(\pi_i)$ is over the TWATA $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$, are already constructed ($i \in \{1, 2\}$). Thus, $[\![A(\pi_i), p_i, q_i]\!] = [\![\pi_i]\!]$ for some states $p_i, q_i \in Q_i$. We first construct the finite automaton $C(\pi_i)$ over the TWATA $\mathcal{V}(\pi_i) = (S''_i, \delta''_i, \mathrm{Acc}''_i)$ as described in Section 9.2.2. Note that $|S''_i| = |S_i| + 2 \cdot |Q_i|^2$. We take $\mathcal{T}(\pi_1 \cap \pi_2) = \mathcal{V}(\pi_1) \uplus \mathcal{V}(\pi_2)$. The finite automaton $A(\pi_1 \cap \pi_2)$ is the product automaton of $C(\pi_1) = (Q_1, \rightarrow_{C(\pi_1)})$ and $C(\pi_2) = (Q_2, \rightarrow_{C(\pi_2)})$, where test transitions can be done asynchronously:

- The state set of $A(\pi_1 \cap \pi_2)$ is $Q_1 \times Q_2$.
- For $a \in \Gamma \cup \overline{\Gamma}$ we have $(r_1, r_2) \xrightarrow{a}_{A(\pi_1 \cap \pi_2)} (r'_1, r'_2)$ if and only if $r_1 \xrightarrow{a}_{C(\pi_1)} r'_1$ and $r_2 \xrightarrow{a}_{C(\pi_2)} r'_2$.
- For a state s_1 of $\mathcal{V}(\pi_1)$ we have the test transition
$$(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2), s_1}_{A(\pi_1 \cap \pi_2)} (r'_1, r_2)$$
if and only if $r_1 \xrightarrow{\mathcal{V}(\pi_1), s_1}_{C(\pi_1)} r'_1$.
- For a state s_2 of $\mathcal{V}(\pi_2)$ we have the test transition
$$(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2), s_2}_{A(\pi_1 \cap \pi_2)} (r_1, r'_2)$$
if and only if $r_2 \xrightarrow{\mathcal{V}(\pi_2), s_2}_{C(\pi_2)} r'_2$.

Lemma 9.10. *We have* $[\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$.

Proof. Since $(u, v) \in [\![\pi_1 \cap \pi_2]\!]$ if and only if $(u, v) \in [\![\pi_1]\!]$ and $(u, v) \in [\![\pi_2]\!]$, we know by induction that it suffices to prove: $(u, v) \in [\![A(\pi_i), p_i, q_i]\!]$ for all $i \in \{1, 2\}$ if and only if $(u, v) \in [\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!]$. So let $(u, v) \in [\![A(\pi_i), p_i, q_i]\!]$ for all $i \in \{1, 2\}$. Then Lemma 9.9 implies the existence of a state $r_i \in Q_i$ such that $(u, \inf(u, v)) \in [\![C(\pi_i)^\downarrow, p_i, r_i]\!]$ and $(\inf(u, v), v) \in [\![C(\pi_i)^\uparrow, r_i, q_i]\!]$ for all $i \in \{1, 2\}$. This implies $(u, \inf(u, v)) \in [\![A(\pi_1 \cap \pi_2)^\downarrow, (p_1, p_2), (r_1, r_2)]\!]$ and $(\inf(u, v), v) \in [\![A(\pi_1 \cap \pi_2)^\uparrow, (r_1, r_2), (q_1, q_2)]\!]$. Thus, we have $(u, v) \in [\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!]$. On the other hand, any run witnessing $(u, v) \in [\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!]$ is a witness for $(u, v) \in [\![C(\pi_i), p_i, q_i]\!]$ for all $i \in \{1, 2\}$. By Lemma 9.9 we obtain $(u, v) \in [\![A(\pi_i), p_i, q_i]\!]$ for all $i \in \{1, 2\}$. ☐

From the construction of $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$ we immediately get:

Remark 9.11. *Let $A(\pi_i) = (Q_i, \rightarrow_{A(\pi_i)})$, $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$, $A(\pi_1 \cap \pi_2) = (Q, \rightarrow_{A(\pi_1 \cap \pi_2)})$ and $\mathcal{T}(\pi_1 \cap \pi_2) = (S, \delta, \mathrm{Acc})$. Then we have (i) $|Q| = |Q_1| \cdot |Q_2|$, (ii) $|S| = |S_1| + |S_2| + 2 \cdot |Q_1|^2 + 2 \cdot |Q_2|^2$, and $|\mathrm{Acc}| = \max\{|\mathrm{Acc}_1|, |\mathrm{Acc}_2|\}$.*

Recall that $\mathcal{Z}$ is the prefix-recognizable system and φ is the PDL$^\cap$ formula of our input and we want to check $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$. A careful analysis of the constructions outlined above, allows us to prove inductively:

Lemma 9.12. *If $|\mathcal{Z}|$ and $|\varphi|$ are sufficiently large, then the following two statements hold:*

- *For any 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 any subprogram π of φ such that $A(\pi) = (Q, \rightarrow_{A(\pi)})$ and $\mathcal{T}(\pi) = (S, \delta, \mathrm{Acc})$ we have (i) $|Q| \leq |\mathcal{Z}|^{|\pi|}$, (ii) $|S| \leq |\mathcal{Z}|^{2 \cdot |\pi|^2}$, and (iii) $|\mathrm{Acc}| \leq |\pi|$.*

Proof. We show the lemma via mutual structural induction over ψ and π. The case $\psi = \mathtt{true}$ is trivial. Let $\pi = \sigma \in \Sigma$ be an atomic program and $\alpha(\sigma) = \bigcup_{i=1}^{n_\sigma} (L(\mathcal{A}_i^\sigma) \times L(\mathcal{B}_i^\sigma)) L(\mathcal{C}_i^\sigma)$. We can assume $\sum_{i=1}^{n_\sigma} |\mathcal{C}_i^\sigma| \geq 2$. Hence, we have $|Q| = \sum_{i=1}^{n_\sigma} (|Q(\mathcal{A}_i^\sigma)| + |Q(\mathcal{B}_i^\sigma)|) + 2$ and therefor $|Q| \leq |\mathcal{Z}|$. Moreover, we have $|S| = \sum_{i=1}^{n_\sigma} |Q(\mathcal{C}_i^\sigma)|$, hence $|S| \leq |\mathcal{Z}|$. By construction we have $|\mathrm{Acc}| = 1 = |\pi|$. Let us continue with the structural induction over ψ. In case $\psi = \neg\theta$ and $\mathcal{T}(\theta) = (S', \delta', \mathrm{Acc}')$ then by the standard complementation of $\mathcal{T}(\theta)$ yielding $\mathcal{T}(\psi)$ we have $S = S'$. Moreover, by induction we have $|\mathrm{Acc}'| \leq |\theta|$, hence $|\mathrm{Acc}| = |\mathrm{Acc}'| + 1 \leq |\theta| + 1 = |\psi|$. Now assume $\psi = \langle \pi \rangle \theta$. Let $A(\pi) = (Q, \rightarrow_{A(\pi)})$, $\mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$, and $\mathcal{T}(\theta) = (S_2, \delta_2, \mathrm{Acc}_2)$. Then by construction we have:

$$\begin{array}{rcl} |S| & = & |Q| + |S_1| + |S_2| \\ & \overset{\text{induction}}{\leq} & |\mathcal{Z}|^{|\pi|} + |\mathcal{Z}|^{2 \cdot |\pi|^2} + |\mathcal{Z}|^{2 \cdot |\theta|^2} \\ & \leq & |\mathcal{Z}|^{4 \cdot |\pi| \cdot |\theta| + 2 \cdot |\pi^2| + 2 \cdot |\theta|^2} \\ & = & |\mathcal{Z}|^{2 \cdot |\psi|^2} \end{array}$$

Let us continue with structural induction of π. The cases $\pi = \chi^*$ and $\pi = \psi?$ are easy to see. Assume $\pi = \pi_1 \cup \pi_2$ or $\pi = \pi_1 \circ \pi_2$ and let Q_i to be the state set of $A(\pi_i)$ and $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$ ($i \in \{1, 2\}$). By the the standard construction we get:

$$\begin{array}{rcl} |Q| & = & |Q_1| + |Q_2| \\ & \overset{\text{induction}}{\leq} & |\mathcal{Z}|^{|\pi_1|} + |\mathcal{Z}|^{|\pi_2|} \\ & \leq & |\mathcal{Z}|^{|\pi_1| + |\pi_2|} \\ & = & |\mathcal{Z}|^{|\pi|} \end{array}$$

The estimation of $|S|$ and $|\mathrm{Acc}|$ is straightforward again.
Now assume $\pi = \pi_1 \cap \pi_2$. Let Q_i to be the state set of $A(\pi_i)$ and $\mathcal{T}(\pi_i) = (S_i, \delta_i, \mathrm{Acc}_i)$

Table 4: PDL$^\cap$ and test-free PDL$^\cap$ over BPA,PDS and PRS.

		BPA	**PDS**	**PRS**
PDL$^\cap$, PDL$^\cap\backslash ?$	**data**	EXP		
	expression	2EXP		
	combined			

($i \in \{1, 2\}$). By Remark 9.11 we have $|Q| = |Q_1| \cdot |Q_2|$ and $|S| = |S_1| + |S_2| + 2 \cdot |Q_1|^2 + 2 \cdot |Q_2|^2$. Hence, we get:

$$\begin{array}{rcl}
|Q| & = & |Q_1| \cdot |Q_2| \\
& \overset{\text{induction}}{\leq} & |\mathcal{Z}|^{|\pi_1|} \cdot |\mathcal{Z}|^{|\pi_2|} \\
& = & |\mathcal{Z}|^{|\pi|} \\
|S| & = & |S_1| + |S_2| + 2 \cdot |Q_1|^2 + 2 \cdot |Q_2|^2 \\
& \overset{\text{induction}}{\leq} & |\mathcal{Z}|^{2 \cdot |\pi_1|^2} + |\mathcal{Z}|^{2 \cdot |\pi_2|^2} + 2 \cdot |\mathcal{Z}|^{2 \cdot |\pi_1|} + 2 \cdot |\mathcal{Z}|^{2 \cdot |\pi_2|} \\
& \leq & |\mathcal{Z}|^{2 \cdot (|\pi_1|^2 + |\pi_2|^2 + |\pi_1| + 1 + |\pi_2| + 1)} \\
& \leq & |\mathcal{Z}|^{2 \cdot (|\pi_1| + |\pi_2|)^2} \\
& = & |\mathcal{Z}|^{2 \cdot |\pi|^2}
\end{array}$$

By induction we have $|\text{Acc}_i| \leq |\pi_i|$ for all $i \in \{1, 2\}$. Thus, $|\text{Acc}| = \max\{|\text{Acc}_1|, |\text{Acc}_2|\} \leq |\text{Acc}_1| + |\text{Acc}_2| \leq |\pi_1| + |\pi_2| + 1 = |\pi|$. □

Hence, by construction of $\mathcal{T}(\varphi)$, Lemma 9.12, and Theorem 9.3 we get a 2EXP upper bound for the combined complexity of PDL$^\cap$ over prefix-recognizable systems.

Theorem 9.13. *The following problem is in* 2EXP*:*
INPUT: A PRS $\mathcal{Z} = (\Gamma, \alpha)$ and a PDL$^\cap$ formula φ each over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$?

When the PDL$^\cap$ formula is not part of the input, the complexity goes down to EXP by Lemma 9.12.

Corollary 9.14. *For any fixed PDL$^\cap$ formula φ over Σ the following problem is in* EXP*:*
INPUT: A PRS $\mathcal{Z} = (\Gamma, \alpha)$ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{Z}), \varepsilon) \models \varphi$?

Table 4 summarizes the results for model-checking (test-free) PDL$^\cap$ over BPA, PDS, and PRS.

10 Further considerations

We can now ask what other kinds of interesting extensions of PDL there are and how the complexity of the model-checking problem over BPA, PDS and PRS increases. Instead of subjoining PDL with the intersection of programs receiving PDL$^{\cap}$, we can as well more generally add an operator for the complementation of programs receiving *PDL*$^{\neg}$. PDL$^{\neg}$ formulas φ and PDL$^{\neg}$ programs π over Σ are given by the following grammar, where $\sigma \in \Sigma$:

$$\begin{array}{lcl} \varphi & ::= & \texttt{true} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle\pi\rangle\varphi \\ \pi & ::= & \sigma \mid \pi_1 \cup \pi_2 \mid \pi_1 \circ \pi_2 \mid \pi^* \mid \neg\pi \mid \varphi? \end{array}$$

The semantic of PDL$^{\neg}$ over a Kripke structure $\mathcal{K} = (S, \{\to_\sigma \mid \sigma \in \Sigma\})$ is defined as for PDL — additionally with the semantic of the complementation of a PDL$^{\neg}$ program defined as follows:

$$[\![\neg\pi]\!]_{\mathcal{K}} = (S \times S) \setminus [\![\pi]\!]_{\mathcal{K}}$$

Clearly we can represent the intersection $\pi_1 \cap \pi_2$ as $\neg(\neg\pi_1 \cup \neg\pi_2)$. The function Tower : $\mathbb{N} \to \mathbb{N}$ is defined as Tower$(0) = 1$ and Tower$(n) = 2^{\mathrm{Tower}(n-1)}$ for all $n \geq 1$. We want to give a simple reduction proving model-checking PDL$^{\neg}$ to be hard for $\mathsf{DTIME}\left(\mathrm{Tower}\left(\frac{n}{\log^*(n)^2}\right)\right)$ already over a fixed BPA. Therefor we introduce star-free expressions. Let Σ be a finite alphabet. Then *star-free expressions* α over Σ are given by the following grammar, where $a \in \Sigma$:

$$\alpha ::= \emptyset \mid \varepsilon \mid a \mid \alpha_1 \cup \alpha_2 \mid \alpha_1; \alpha_2 \mid \widehat{\alpha}$$

The *language* $L(\alpha)$ defined by a star-free expression α is inductively defined as $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(\alpha_1 \cup \alpha_2) = L(\alpha_1) \cup L(\alpha_2)$, $L(\alpha_1; \alpha_2) = L(\alpha_1)L(\alpha_2)$ and finally $L(\widehat{\alpha}) = \Sigma^* \setminus L(\alpha)$. The *size* of a star-free expression α is inductively defined as $|\emptyset| = |\varepsilon| = 1$, $|\alpha_1 \cup \alpha_2| = |\alpha_1; \alpha_2| = |\alpha_1| + |\alpha_2| + 1$ and finally $|\widehat{\alpha}| = |\alpha| + 1$.

Theorem 10.1 ([11])**.** *For a fixed alphabet Γ the following problem is hard for* $\mathsf{DTIME}\left(\mathrm{Tower}\left(\frac{\mathrm{n}}{\log^*(\mathrm{n})^2}\right)\right)$*:*
INPUT: A star-free expression α over Γ.
QUESTION: $L(\alpha) = \Gamma^*$?

Corollary 10.2. *There exists a fixed BPA $\mathcal{X} = (\Gamma, \Delta)$ over Σ such that the following problem is hard for* $\mathsf{DTIME}\left(\mathrm{Tower}\left(\frac{\mathrm{n}}{\log^*(\mathrm{n})^2}\right)\right)$*:*
INPUT: A PDL$^{\neg}$ formula φ over Σ.
QUESTION: $(\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi$?

Proof. Given a star-free expression α over some fixed alphabet Γ we give a fixed BPA $\mathcal{X} = (\Gamma, \Delta)$ and a (polynomial time computable) PDL$^{\neg}$ formula φ each over Σ such that

$$L(\alpha) = \Gamma^* \qquad \Leftrightarrow \qquad (\mathcal{K}(\mathcal{X}), \varepsilon) \models \varphi.$$

We put $\Sigma = \Gamma \cup \overline{\Gamma} \cup \{\text{empty}\}$ and define Δ to be the transition relation of Tree_Γ (see Example 4.1). Note that the program empty $\in \Sigma$ does not occur as a labeling in Δ. We give a test-free $\text{PDL}^\neg$ program $\|\alpha\|$ over $\overline{\Gamma}$ such that $[\![\|\alpha\|]\!] = \{(wx, x) \mid x \in \Gamma^*, w \in L(\alpha)\}$ inductively follows:

$$\begin{aligned}
\|\emptyset\| &= \text{empty} \\
\|a\| &= \overline{a} \\
\|\alpha_1 \cup \alpha_2\| &= \|\alpha_1\| \cup \|\alpha_2\| \\
\|\alpha_1; \alpha_2\| &= \|\alpha_1\| \circ \|\alpha_2\| \\
\|\widehat{\alpha}\| &= \overline{\Gamma}^* \cap \neg\|\alpha\|
\end{aligned}$$

for all $a \in \Gamma$. Finally we have:

$$L(\alpha) = \Gamma^* \quad \Leftrightarrow \quad \forall w \in \Gamma^* : (w, \varepsilon) \in [\![\|\alpha\|]\!] \quad \Leftrightarrow \quad (\mathcal{K}(\mathcal{X}), \varepsilon) \models [\Gamma^*]\langle\|\alpha\|\rangle\neg\langle\overline{\Gamma}\rangle\texttt{true}$$

$\square$

11 Open problems

A fruitful research direction might be to extend $\text{PDL}^\cap$ by a unary fixpoint operator. The resulting logic is strictly more expressive than $\text{PDL}^\cap$ and the modal μ-calculus. We are confident that our upper bounds for $\text{PDL}^\cap$ from Theorem 9.13 and Corollary 9.14 can be extended to this logic.

References

[1] O. M. Ahmed Bouajjani, Javier Esparza. Reachability analysis of pushdown automata: Application to model-checking. In A. W. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97), Warsaw (Poland)*, number 1243 in Lecture Notes in Computer Science, pages 135–150. Springer, 1997.

[2] 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.

[3] A. Blumensath. Prefix-Recognisable Graphs and Monadic Second-Order Logic. Technical Report 2001-06, RWTH Aachen, Department of Computer Science, 2001.

[4] T. Cachat. Uniform Solution of Parity Games on Prefix-Recognizable Graphs. *Electronic Notes in Theoretical Computer Science*, 68(6), 2002.

[5] D. Caucal. On Infinite Transition Graphs Having a Decidable Monadic Theory. *Theoretical Computer Science*, 290(1):79–115, 2002.

[6] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[7] R. Danecki. Nondeterministic Propositional Dynamic Logic with intersection is decidable. In *Proceedings of the 5th Symposium on Computation Theory (Zaborw, Poland)*, number 208 in Lecture Notes in Computer Science, pages 34–53, 1984.

[8] J. Esparza. On the Decidabilty of Model Checking for Several mu-calculi and Petri Nets. In S. Tison, editor, *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP '94), Edinburgh (U.K.)*, number 787 in Lecture Notes in Computer Science, pages 115–129. Springer, 1994.

[9] J. Esparza, A. Kucera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Information and Computation*, 186(2):355–376, 2003.

[10] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[11] M. Fürer. *Nicht-elementare untere Schranken in der Automatentheorie*. PhD thesis, ETH Zürich, 1978. In German.

[12] D. Harel. Recurring dominoes: making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

[13] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of computing. The MIT Press, 2000.

[14] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 40–46, New York, NY, USA, 1974. ACM Press.

[15] O. Kupferman, N. Piterman, and M. Vardi. Model Checking Linear Properties of Prefix-Recognizable Systems. In E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14. International Conference on Computer Aided Verification CAV 2002, Copenhagen (Denmark)*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385, 2002.

[16] O. Kupferman and M. Y. Vardi. An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000), Chiacago (USA)*, number 1855 in Lecture Notes in Computer Science, pages 36–52. Springer, 2000.

[17] M. Lange. Model Checking Propositional Dynamic Logic with All Extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[18] M. Lange and C. Lutz. 2-ExpTime Lower Bounds for Propositional Dynamic Logics with Intersection. *Journal of Symbolic Logic*, 70(4):1072–1086, 2005.

[19] C. Löding and O. Serre. Propositional Dynamic Logic with Recursive Programs. In *Proceedings of Foundations of Software Science and Computation Structures 2006*. Springer, 2006.

[20] D. Lugiez and P. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 274(1–2):89–115, 2002.

[21] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[22] R. Mayr. Process Rewrite Systems. *Information and Computation*, 156(1):264–286, 2000.

[23] R. Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 256(1-2):31–62, 2001.

[24] D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987.

[25] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[26] C. Stirling. Decidability of Bisimulation Equivalence for Pushdown Processes. *unpublished*, 2000.

[27] W. Thomas. A Short Introduction to Infinite Automata. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Proceedings of the 5th International Conference on Developments in Language Theory (DLT 2001), Vienna (Austria)*, number 2295 in Lecture Notes in Computer Science, pages 130–144. Springer, 2001.

[28] M. Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC 1982)*, pages 137–146. ACM Press, 1982.

[29] M. Y. Vardi. Reasoning about the past with two-way automata. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP '98), Aalborg (Denmark)*, number 1443 in Lecture Notes in Computer Science, pages 628–641. Springer, 1998.

[30] I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In S. Kapoor and S. Prasad, editors, *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000), New Delhi (India)*, number 1974 in Lecture Notes in Computer Science, pages 127–138. Springer, 2000.

[31] I. Walukiewicz. Pushdown Processes: Games and Model-Checking. *Information and Computation*, 164(2):234–263, 2001.

[32] S. Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products*. Dissertation, RWTH Aachen, 2005.