Computational Complexity of Propositional Dynamic Logics

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

Doctor rerum naturalium
(Dr. rer. nat.)

im Fachgebiet

Informatik

vorgelegt von

Dipl.-Inf. Stefan Göller
geboren am 5. November 1981 in Stuttgart.

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Markus Lohrey (Universität Leipzig)
2. Prof. Dr. Damian Niwiński (Universität Warschau)
3. Prof. Dr. Colin Stirling (Universität Edinburgh)

Die Verleihung des akademischen Grades erfolgt auf Beschluss des Rates der Fakultät für Mathematik und Informatik vom 15.12.2008.

Acknowledgments

First and foremost, I want to thank my advisor Markus Lohrey for his excellent supervision, for his patience, and for his constant and honest interest in the research I have been doing. I am very glad to have had such an extraordinarily capable and friendly advisor and I hope that our collaboration will continue.

Next, I feel honored to have Damian Niwińksi and Colin Stirling as reviewers of this thesis. I thank them for reviewing this thesis within the short given time.

I want to thank the members of the graduation commitee Manfred Droste, Martin Bogdan, Gerhard Brewka, Dietrich Kuske and Martin Middendorf for a fast procedure.

I yield a lot of thanks to the Deutsche Forschungsgemeinschaft (DFG) for support through the project GELO over the years.

I want to thank my co-authors Carsten Lutz and Dirk Nowotka for the interesting discussions we had. I hope that the collaboration will continue.

I want to thank all my former colleagues of the Department of Theoretical Computer Science in Stuttgart Volker Diekert, Ulrich Hertrampf, Benjamin Hoffmann, Manfred Kufleitner, Nicole Ondrusch, Holger Petersen, Heike Photien and Horst Prote for a really nice time. Moreover, I am grateful to my friends and former colleagues from the Computer Science faculty at the University of Stuttgart for the numerous creative ways of distracting ourselves from working hard.

I thank my colleagues at the University of Leipzig for a really nice time. Especially, I would like to thank Niko Haubold, Daniel Kirsten, Dietrich Kuske, Christian Mathissen, and Ingmar Meinecke for always having an open ear and Andrea Hesse and Karin Quaas for enjoyable coffee sessions.

Finally I thank my family and my friends for their love and support.

To my parents and my brother

Contents

1 Introduction

First-order Dynamic Logic (DL), originally introduced by Pratt [91] and carried on by Harel et al. [54], is a formal system that allows to reason about programs. Unlike first-order logic, where free variables are statically evaluated, there are special constructs in DL that allow to describe the dynamic change of valuations of variables, namely *programs*. In DL each formula defines a set of valuations of variables (over some domain of computation as for example the natural numbers). Programs, on the other hand, define a binary relation on the set of valuations: Intuitively a program π relates valuation f with valuation g whenever g can be reached after execution of the program π under f . Thus, DL allows to model the input-output behaviour of programs. Several extensions of DL that comprise more complex data structures than the mere usage of variables, as for example stacks, have been introduced. Some of them increase the expressive power of DL, some do not; see [53] for an intensive treatment.

Propositional Dynamic Logic (PDL), introduced by Fischer and Ladner [36], can be seen to relate to DL as propositional logic relates to first-order logic. More precisely, primitive PDL formulas (resp. programs) are abstracted and simply range over a countable set of *atomic propositions* $p, q, \ldots$ (resp. *atomic programs* $a, b, \ldots$). Complex formulas and programs are mutually defined from atomic propositions and programs. Formulas can be build using the boolean connectives and the diamond modality, i.e. $\langle\pi\rangle\varphi$ is a formula whenever π is program and φ is a formula. Programs can be seen as a regular language over atomic programs and test programs $\varphi?$ (where φ is a formula) and are represented using the operators union, composition, and Kleene star. As indicated above, since the domain of computation is abstracted in PDL, we no longer speak of valuations. Instead, formulas and programs are interpreted on state and edge labeled transition systems, also known as *Kripke structures*. Although being strictly less expressive than DL, there are fundamental properties of computation which are expressible in PDL.

An important algorithmic problem of a logic is satisfiability. The *satisfiability problem* asks, given some formula φ as input, whether there exists a Kripke structure K such that φ holds in some state of K. In contrast to the high undecidability status of DL, satisfiability in PDL is decidable and complete for deterministic exponential time [36, 94]. The reason that PDL behaves well in a complexity theoretical sense is its quite simple model theory. It is well-known that PDL is bisimulation-invariant. This implies that PDL has the tree model property, i.e. every satisfiable PDL formula has a model that is a tree. Furthermore PDL has the finite model property, i.e. every satisfiable PDL formula has a finite model [36].

In the early days of PDL, researchers mainly concentrated on satisfiability 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. The *model checking problem* asks, given some formula φ, some Kripke structure K and some state x of K, whether φ holds in x. Model checking is the prime reasoning problem in program verification [20], and has traditionally been studied for finite Kripke structures. Lange studies this original problem in a PDL context in [71], proving P-completeness for PDL and various extensions of it.

In this thesis, we study the computational complexity of the satisfiability problem and of the *infinite state* model checking problem of PDL and some of its extensions. As expected, in the infinite state model checking problem the input Kripke structures can be infinite but there is some formalism for finitely representing them. In recent years, verification of infinite state systems became a major topic in the model checking community. 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 [77, 85, 129, 128] in the context of infinite state systems. For instance, consider the property that a process can reach a state, where a condition φ holds, via a path on which the action sequence $a_1 a_2 \cdots a_n$ is repeated cyclically. Clearly, this can be expressed in Kozen's μ-calculus [65] (if φ can be expressed in the modal μ-calculus), but clearly the PDL formula $\langle (a_1; a_2; \ \cdots \ ; a_n)^* \rangle \varphi$ is a more readable specification; consult the textbook [5] for a thorough treatment of the modal μ-calculus. Secondly, and more important, IPDL (i.e. the extension of PDL with the intersection on programs) allows to formulate natural system properties that cannot be expressed in the modal μ-calculus (since they are not bisimulation invariant), like for instance that a system can be reset to the current state (Example 2.23) or that two forking pushdown processes may synchronize in the future (Example 2.35). When model checking infinite state systems, we need a formalism for finitely representing them. A number of different formalisms are available, which vary considerably in expressive power and give rise to different versions of the infinite state model checking problem, see [111] for a survey. In the rest of this discussion, we follow Mayr's uniform classification in terms of parallel and sequential composition [85]. When parallel composition is admitted to describe infinite state systems, model checking is undecidable already for test-free PDL as we will show. Because of this negative result, we concentrate on sequential composition. Mayr's classification thus leads us to pushdown systems (PDSs) and basic process algebras (BPAs). Pushdown systems were used to model the state space of programs with nested procedure calls [31], and model checking PDSs was studied for various temporal logics such as LTL, CTL, and the modal μ-calculus [3, 31, 68, 124, 125]. For a thorough study of temporal logics we refer to [104, 20]. BPAs are a special case of pushdown systems, namely those with a single internal state and were studied e.g. in [83, 50, 102]. In addition to pushdown systems and basic process algebras, we consider prefix-recognizable systems. Prefix-recognizable systems were introduced by Caucal

[18] and strictly extend pushdown systems. There exist numerous natural characterizations of prefix-recognizable systems, as e.g. the class of graphs (i.e. systems) that are MSO-interpretable in the complete binary tree [18], see also [73] for a comprehensive study.

Since its invention, PDL has become a classic of logic in computer science [53], and many extensions and variations have been proposed. PDL extensions are often inspired by the original application of reasoning about programs, while others aim at the numerous novel applications that PDL has found since its invention. Notable examples of such applications include reasoning about agent-based systems [86], regular path constraints for querying semi-structured data [2], and XML-querying [1, 110, 109]. In artifical intelligence, PDL received attention due to its close relationship to description logics [40] and epistemic logics [116, 118]. In the following, we survey different PDL extensions and summarize to what extend this thesis contributes to them.

Succinct representations: Since each PDL program can be seen as a regular language over the set of atomic programs and test programs, it is natural to ask whether the complexities of reasoning problems increase when programs are represented succinctly. A result of Mayer and Stockmeyer [81] states that adding interleaving (also known as shuffle) on programs leads to a PDL extension for which satisfiability is complete for deterministic doubly exponential time. Another (maybe more natural) way to gain succinctness would be, instead of representing programs by regular expressions, to use nondeterministic finite automata, yielding APDL. Despite the fact that nondeterministic finite automata are exponentially more succinct than regular expressions [27], satisfiability in APDL is still decidable in determininistic exponential time [53].

Extensions by program operators: A further prominent way to obtain extensions of PDL is to admit additional program operators besides union, composition and Kleene star. Some of the resulting logics are well-behaved in the sense that they do not alter the model theory and computational complexity of PDL in a dramatic way. For example, PDL extended with the converse program operator (CPDL) inherits the (two-way) tree model property from PDL; moreover, the above mentioned classical result of Fischer/Ladner and Pratt, stating that satisfiability in PDL is complete for EXP (deterministic exponential time) [36, 94], extends to CPDL [120]. Noteworthy in [35] it has been shown that several logics of knowledge can be interpreted in CPDL.

Other extensions are more challenging. For instance, it is well-known that satisfiability of PDL with negation of programs (NPDL) is complete for Σ_1^1, the first level of the analytic hierarchy [96]. However, restricting NPDL by allowing program negation only in front of atomic programs, yields an EXP-complete logic that has interesting applications in description logics and allows to express nice properties such as the universal modality or the window operator [80]. A maybe more prominent fragment of NPDL is IPDL, i.e. PDL extended with the intersection on programs[1], first introduced in [57]. IPDL has an intricate model theory that involves the loss of the tree model property and of the finite model property. As a consequence, it is not possible to easily transfer the standard decision procedures for PDL to IPDL, e.g. by using automata on infinite trees [122] or a translation into the alternation-free fragment of the modal μ-calculus.

[1]Note that, as for negation, the semantics of intersection is defined to be the intersection of the binary relations defined by the programs rather than intersection on a language level.

The history of this family of extensions of PDL started in 1984, when Danecki proved an upper bound of 2EXP (deterministic doubly exponential time) for satisfiability in IPDL using an intricate reduction to the non-emptiness problem of automata on infinite trees [22]. Alas, Danecki's proof is rather difficult and many details have been omitted in the published version. More than 20 years later, the lower bound for IPDL was improved to EXPSPACE-hardness [70] and finally tightened to 2EXP-hardness by Lange and Lutz [72]. PDL with intersection also has its applications in the context of XML querying. The XML query language XPath 2.0 allows to specify queries using an intersection operator on path expressions. In [110] the complexity of the query containment problem of (variants of) XPath 2.0 has been studied.

When adding both intersection and converse to PDL one obtains the logic ICPDL with a number of interesting applications in computer science. For example, the information logic DAL (for Data Analysis Logic) [33] and a number of epistemic logics for reasoning about distributed knowledge (which corresponds to program intersection [32]) can be embedded into ICPDL. This enables the transfer of results such as decidability and upper complexity bounds. Since programs in DAL are interpreted as equivalence relations, the presence of converse is important for embedding in this context – see [78] for more details. ICPDL also provides an important background theory for description logics with role intersection [6]. In a recent paper of Lutz satisfiability in ICPDL has been shown to be decidable using a reduction to the monadic second order theory of the infinite binary tree [78]. However, this only yields a non-elementary algorithm that does not match the 2EXP lower bound that ICPDL inherits from IPDL. In [45, 46], Lohrey, Lutz, and the author proved that ICPDL is in 2EXP, thus determining the precise complexity of ICPDL as 2EXP-complete. Is is worth observing that from Danecki's paper [22], in which it was shown that satisfiability in IPDL is in 2EXP, it is not at all clear how his techniques can be used to show that satisfiability in ICPDL is in 2EXP. Proving ICPDL's membership in 2EXP is also content of this thesis but we prove a more general result by extending ICPDL by allowing unary fixed point operators and additionally the usage of nominals. Nominals are atomic propositions that are true in precisely one state of Kripke structures and can hence be thought of as constants. We call the resulting logic μ-ICPDL$_{\mathtt{Nom}}$; it extends both the modal μ-calculus and ICPDL and allows nominals. With respect to expressiveness, our logic μ-ICPDL$_{\mathtt{Nom}}$ is incomparable to guarded least fixed point logic from [49]. Enriched μ-calculi and PDL with (some of) the operators converse, graded modalities and nominals have been investigated in [98, 67, 17, 38]. Graded modalities allow to express the existence of at least k (resp. at most k) successors satisfying some property. Numerous description logics, standardized as the ontology language OWL, demand for the above extensions and can be embedded into correspondingly enriched μ-calculi. Adding all of the operators converse, graded modalities and nominals, yields the fully enriched μ-calculus. It has been shown that satisfiability of the fully enriched μ-calculus is undecidabile [13]. Carrying on the work on enriched μ-calculi from [98, 67, 17, 38], Bonatti et al. [12] established EXP-completeness of the satisfiability problem for strict fragments of the fully enriched μ-calculus (i.e. fragments that are obtained by dropping at least one of the operators converse, graded modalities, or nominals) via the automata-theoretic approach. Although satisfiability of μ-ICPDL$_{\mathtt{Nom}}$ will turn out to be 2EXP-complete, we gain a better complexity upper bound when imposing a syntactic restriction on nestings of the intersection operator by a notion that we call *intersection width*. Despite its high expressiveness, satisfiability of μ-ICPDL$_{\mathtt{Nom}}$ formulas of constant intersection width is EXP-complete. It is interesting to remark that adding graded modalities already to IPDL leads to a highly undecidabile logic due to the result that deciding if a given IPDL formula has a *deterministic* model is Σ^1_1-complete [53].

Reasoning about dynamic policies: Numerous applications contain a set of policies that describe what is prohibited and what is permitted. Policies arise in many contexts. For one, they can comprise control policies, thus specifying which agents are permitted to access resources. They can as well be legal policies, describing actions that are legally permitted. In the course of time, the policies of an application may change e.g. due to interaction between the application and its user(s). When changing policies, the system has to guarantee that unintentional side-effects do not occur. Furthermore, it is often not straightforward to decide whether to modify the current policy set or not, not to mention the duty of creating it. Various examples of such typical practical scenarios are given in [95]. In order to allow comparison of different policies and reasoning about them, a variety of languages have been introduced, an overview is given in [126]. Van der Meyden's Dynamic Logic of Permission (DLP) is an example of an expressive logic that allows to reason about dynamic policies. Formally, it extends test-free PDL and allows to reason about a fixed policy set that describes the set of all permitted transitions of a system, which is in turn modeled as a Kripke structure. In addition to test-free PDL, the logic DLP allows to ask queries of the kind 'does there exist a sequence of solely permitted transitions to some state where the property φ holds' and 'does every sequence of transitions to states satisfying φ solely consist of permitted transitions'. Extending DLP, the logic $\mathrm{DLP_{dyn}}$ of Pucella and Weissman [95] additionally allows to update the policy set by removing and adding transitions. These removed or added transitions are defined as the direct product of two sets of states, each specified by boolean combinations of atomic propositions. In [95], several applications are stated that demand for the possibility to update the policy set within the logic. The even more general logic $\mathrm{DLP^+_{dyn}}$, introduced by Demri [24], allows to update the policy set by adding (via the $\mathbf{grant}$-operator) and removing (via the $\mathbf{revoke}$-operator) a direct product of state sets, but each specified by an *arbitrary* formula of the logic itself. Concerning the complexity of the satisfiability problem, gaps both for $\mathrm{DLP^+_{dyn}}$ and its fragment $\mathrm{DLP_{dyn}}$ existed. Satisfiability in $\mathrm{DLP_{dyn}}$ was known to lie in NEXP, whereas for $\mathrm{DLP^+_{dyn}}$ a $\mathsf{2EXP}$ upper bound was known; both $\mathrm{DLP_{dyn}}$ and $\mathrm{DLP^+_{dyn}}$ are EXP-hard due to EXP-hardness of PDL [36]. In this thesis, we close these complexity gaps by showing that already satisfiability in $\mathrm{DLP^+_{dyn}}$ can be decided in EXP, hence positively answering an open problem formulated in [24].

Non-regular extensions: A further way to gain expressiveness is to enrich PDL programs by allowing non-regular programs to appear in modalities of formulas. A simple but elegant argument that any non-regular extension leads to a logic that is strictly more expressive than PDL is due to Harel and Singerman [59]. In a general result of Harel and Raz [58] it was shown that extending PDL with programs that are defined by *simple-minded pushdown automata* leads to a logic with a decidable satisfiability problem. Even more general, Löding, Lutz and Serre [75] recently proved that satisfiability in *recursive PDL*, i.e. PDL extended by visibly pushdown languages, introduced in [4], is decidable and complete for deterministic doubly exponential time. The decisive property that is exploited for proving decidability of satisfiability in recursive PDL, is that visibly pushdown automata are determinizable and that the class of visibly pushdown languages has effective closure properties. Recursive PDL allows to express interesting behaviours of programs such as the set of executions of a recursive program [53]. Decidability of recursive PDL can be seen rather surprising when bearing in mind that PDL extended with the singleton deterministic context-free language $\{a^n b a^n \mid n \geq 0\}$ is highly undecidable [56]. Other extensions considered non-regular PDL extensions over a singleton alphabet [59, 55, 34, 56]. To mention a few results, it was shown that PDL extended with the non-regular language $\{a^{2^n} \mid n \geq 0\}$ is undecidable [55], the extension $\{a^{n^2} \mid n \geq 0\}$ is decidable [34], whereas decidability of PDL with $\{a^{n^3} \mid n \geq 0\}$ is open so far. Note however that the latter extension allows to define properties that are equivalent to long-standing open problems in number theory, as for instance the question whether every integer greater than 10^5 is the sum of five cubes. One can expect that proving decidability of this PDL extension is at least as hard as proving the mentioned number-theoretic open problem. Summarizing, the search for the precise borderline between decidable and undecidable non-regular PDL extensions has been going for quite some time and still is a very challenging research topic.

For further PDL extensions, the author refers to the textbook of Harel, Kozen, and Tiuryn [53].

Structure and main results of the thesis

In what follows, we summarize the content of each of the chapter of the thesis. Furthermore, the main results, according to the author, are emphasized.

Chapter 2 begins with recalling basic notions from computational complexity. Then, we formally introduce the syntax and semantics of Propositional Dynamic Logic. Thereafter we discuss several PDL extensions and prove some minor new contributions. Concerning these PDL extensions, we glance at some model theoretic aspects, but mainly concentrate on the complexity of satisfiability and model checking. Finally, infinite state systems are introduced.

In **Chapter 3** we introduce the logic μ-ICPDL$_{\texttt{Nom}}$. This logic can thought of as the extension of PDL with intersection and converse that additionally allows the usage of fixed point constructs (à la modal μ-calculus [5]) and the usage of nominals, i.e. atomic propositions that hold in precisely one state of each Kripke structure. We define an operator on pointed Kripke structures that preserves μ-ICPDL$_{\texttt{Nom}}$ theories and results in Kripke structures of bounded tree width, a widespread notion from graph theory [26]. Next, we introduce a reasoning problem called ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$. The *ω-regular tree satisfiability problem* asks, given a two-way alternating parity tree automaton $\mathcal{T}$ over infinite trees and some μ-ICPDL$_{\texttt{Nom}}$ formula φ that is interpreted over infinite trees, whether there exists a tree accepted by $\mathcal{T}$ that satisfies φ in its root. We prove that ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$ is complete for 2EXP. This leads to two applications. Concerning μ-ICPDL$_{\texttt{Nom}}$, we give reductions both from the satisfiability problem and from the model checking of prefix-recognizable systems to ω-regular tree satisfiability. Hence, we obtain that both satisfiability and model checking prefix-recognizable systems in μ-ICPDL$_{\texttt{Nom}}$ is in 2EXP. Next, we fully investigate the complexity of infinite state model checking μ-ICPDL$_{\texttt{Nom}}$ and certain fragments thereof. We provide two lower bound proofs. The first one shows the data complexity of model checking both loop-PDL and test-free IPDL over basic process algebras is hard for EXP. This is rather surprising since the data complexity of the modal μ-calculus over basic process algebras is in P [125]. Our second lower bound shows that already over some fixed basic process algebra model checking test-free IPDL is hard for 2EXP. Since satisfiability in PDL with atomic program negation is decidable and EXP-complete [80], we ask if the extension of μ-ICPDL$_{\texttt{Nom}}$ with negation of atomic programs is decidable. The latter question is answered negatively: We prove that already IPDL with atomic program negation is complete for Σ^1_1. Finally, we conclude with open problems.

The main results of Chapter 3 can be summarized as follows:

- Every satisfiable formula of μ-ICPDL$_{\texttt{Nom}}$ that uses at most k different nominals has a countable model of tree width at most $k+2$ (Corollary 3.6).
- Satisfiability in μ-ICPDL$_{\texttt{Nom}}$ is 2EXP-complete (Theorem 3.28) and satisfiability of μ-ICPDL$_{\texttt{Nom}}$ formulas of constant intersection width is EXP-complete (Corollary 3.29).
- Infinite state model checking of μ-ICPDL$_{\texttt{Nom}}$ is in 2EXP on prefix-recognizable systems (Theorem 3.31) and is already 2EXP-hard over some fixed basic process algebra (Theorem 3.35).

The results of Chapter 3 are based on joint work with Markus Lohrey and Carsten Lutz. With μ-ICPDL$_{\texttt{Nom}}$ replaced by ICPDL, Chapter 3 has been published in [43, 45, 46].

Chapter 4 investigates the complexity of satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$. Via a natural satisfiability preserving translation to PDL, Demri proved that satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is in 2EXP [24]. For $\mathrm{DLP}^+_{\mathrm{dyn}}$'s fragment $\mathrm{DLP}_{\mathrm{dyn}}$ of Pucella and Weissman, the best known upper bound was NEXP [95]. We close this complexity gap by showing that satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is complete for EXP. The starting point for proving our EXP upper bound is a proposal by Demri from [24] that suggests to translate $\mathrm{DLP}^+_{\mathrm{dyn}}$ to an extension of PDL, called PDL$\oplus$, that allows a certain operator $\oplus$ on programs. Hence, for proving an EXP upper bound for $\mathrm{DLP}^+_{\mathrm{dyn}}$, the hope would be to prove an EXP upper bound for PDL$\oplus$. We show that this approach may not be successful by providing a 2EXP lower bound for satisfiability in PDL$\oplus$. However, we syntactically restrict the $\oplus$ operator, which leads to the logic that we call PDL$\oplus[\mathbb{A}]$. It is worth mentioning that Demri's translation from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to PDL$\oplus$ does not yield formulas of PDL$\oplus[\mathbb{A}]$. Summarizing, we prove EXP membership of satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ in two steps. In a first step, we give a satisfiability preserving translation from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to PDL$\oplus[\mathbb{A}]$. In a second step, we prove that satisfiability in PDL$\oplus[\mathbb{A}]$ is in EXP. We continue with a discussion of a 2EXP-hard variant of $\mathrm{DLP}^+_{\mathrm{dyn}}$.

The main result of Chapter 4 can be stated as follows:

- Satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is EXP-complete (Theorem 4.5).

The results of chapter 4 have been published in [41].

In **Chapter 5** we systematically determine the complexity of infinite state model checking of PDL and fragments of it. As indicated above, aside from prefix-recognizable systems, we follow Mayr's classification of infinite state systems. Recall that the reason for blinding out systems of Mayr's process rewrite systems [85] that involve parallel composition is that already model checking test-free PDL over basic parallel processes (also known as communication-free nets) is undecidable. We show the latter via a reduction from the undecidable model checking problem of CTL's fragment EF over Petri nets [29, 30].

Following Vardi [119], we distinguish three ways of measuring the complexity of the model checking problem: Data complexity (the formula is fixed, the system is the input), expression complexity (the system is fixed, the formula is the input) and combined complexity (both the formula and the system are part of the input). We prove that over prefix-recognizable systems already reachability is EXP-hard and thus matches the EXP upper bound that is known for even solving parity games [15]. Our EXP lower bound is somewhat surprising, since the complexity of *alternating* reachability over prefix-recognizable systems is EXP-complete accompanied by the fact that for most graph classes there is a complexity jump from the reachability problem to the alternating reachability problem. Furthermore EXP-completeness of reachability on prefix-recognizable systems stands in contrast to the rather low complexity (P-completeness) of reachability on pushdown systems [14]. For the sake of completeness, we investigate the complexity of infinite state model checking of test-free and full PDL via reductions from known results of infinite state model checking of EF, CTL and the modal μ-calculus respectively. We conclude Chapter 5 with a discussion of the open problem of model checking EF and PDL over one-counter systems.

The described results appeared as a technical report [44] and were published in [43, 42]. The main result of Chapter 5 can be stated as follows:

- Reachability on prefix-recognizable systems is EXP-complete (Theorem 5.5).

Chapter 5 is based on joint work with Markus Lohrey.

We give a conclusion in **Chapter 6**.

The author recommends to read Chapter 2 first, since basic notions are introduced there. Chapter 3, 4, and 5 can be read independently expect that Section 4.4 uses the notion of two-way alternating parity tree automata on infinite trees, which are introduced in Definition 3.10 of Section 3.3. Except for Chapter 2, we discuss open problems at the end of each chapter.

2 Preliminaries

2.1 General notation

Let A and B be sets. If additionally $A \cap B = \emptyset$, then $A \uplus B$ denotes $A \cup B$ and recalls the fact that A and B are disjoint. By $\overline{A} = \{\overline{a} \mid a \in A\}$ we denote a *disjoint copy of* A. Let $f : A \to B$ be a function. Then for every subset $C \subseteq A$ we define $f(C) = \{f(c) \mid c \in C\}$. For every $b \in B$ let $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. For each partial function $f : A \to B$, let $\mathrm{dom}(f) = \{a \in A \mid \exists b \in B : f(a) = b\}$ denote the *domain* of f. By $\mathcal{B}^+(A)$ denote set of *positive boolean formulas* with variables over A. Formally $\mathcal{B}^+(A)$ is the smallest set that satisfies the following conditions: (i) $\mathtt{true}, \mathtt{false} \in \mathcal{B}^+(A)$, (ii) $a \in \mathcal{B}^+(A)$ for each $a \in A$, (iii) if $\varphi_1, \varphi_2 \in \mathcal{B}^+(A)$, then $\varphi_1 \vee \varphi_2 \in \mathcal{B}^+(A)$, and (iv) if $\varphi_1, \varphi_2 \in \mathcal{B}^+(A)$, then $\varphi_1 \wedge \varphi_2 \in \mathcal{B}^+(A)$. We say a subset $C \subseteq A$ *satisfies* a positive boolean formula $\varphi \in \mathcal{B}^+(A)$ if the characteristic function $f : A \to \{\mathtt{true}, \mathtt{false}\}$ that corresponds to C is a satisfying truth assignment of φ.

By $\mathbb{N} = \{0, 1, \ldots\}$ we denote the *natural numbers* and by $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ the *positive natural numbers*. For each $i, j \in \mathbb{N}$ let $[i, j] = \{i, i+1, \ldots, j\}$ denote the interval between i and j and let $[j] = [1, j]$ denote the first j positive natural numbers. For a function $f : \mathbb{N} \to \mathbb{N}$ let

$$\mathrm{poly}(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists \text{ polynomial } p : \mathbb{N} \to \mathbb{N}\ \forall n \in \mathbb{N} : g(n) \leq p(f(n))\}$$

denote the class of functions that grow at most polynomially in f and let

$$\exp(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists \text{ polynomial } p : \mathbb{N} \to \mathbb{N}\ \forall n \in \mathbb{N} : g(n) \leq 2^{p(f(n))}\}$$

denote the class of functions that grow at most exponentially in f.

A *nondeterministic finite automaton (NFA)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \to 2^Q$ is a *transition* *function*, and $F \subseteq Q$ is a set of *final states*. Let us inductively define $\xrightarrow{w}_{\mathcal{A}} \subseteq Q \times Q$ for each word $w \in \Sigma^*$ as the smallest relation such that for all $q \in Q$ we have $q \xrightarrow{\varepsilon}_{\mathcal{A}} q$ and for all $q, q', q'' \in Q$, all $w \in \Sigma^*$, and all $a \in \Sigma$ we have: if $q \xrightarrow{w}_{\mathcal{A}} q'$ and $q'' \in \delta(q', a)$, then $q \xrightarrow{wa}_{\mathcal{A}} q''$. By $L(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w}_{\mathcal{A}} q\}$ we denote the *language* of $\mathcal{A}$. For each state $q \in Q$ we denote by $\mathcal{A}_q = (Q, \Sigma, q, \delta, F)$ the same NFA as $\mathcal{A}$ but with initial state q. The *size* of $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q| + |\Sigma| + \sum_{q \in Q, a \in \Sigma} |\delta(q, a)|$. If the NFA $\mathcal{A}$ is not explicitly given then we denote by $Q(\mathcal{A})$ its state set, by $\Sigma(\mathcal{A})$ its alphabet, by $q_0(\mathcal{A})$ its initial state, by $\delta(\mathcal{A})$ its transition function, and by $F(\mathcal{A})$ its set of final states. For a word $w = a_1 \cdots a_n \in \Sigma^*$, where $a_i \in \Sigma$ for each $i \in [n]$, we denote by $\overleftarrow{w} = a_n \cdots a_1$ the *reversal* of w.

2.2 Complexity theory

In this section we recall notions from complexity theory. We assume the reader has some basic background. For a more detailed treatment, we refer to the textbook of Papadimitriou [90].

The following complexity classes are relevant in this thesis:

- L: The class of languages that can be accepted by some logarithmically space bounded deterministic Turing machine.
- NL: The class of languages that can be accepted by some logarithmically space bounded nondeterministic Turing machine.
- P: The class of languages that can be accepted by some polynomially time bounded deterministic Turing machine.
- NP: The class of languages that can be accepted by some polynomially time bounded nondeterministic Turing machine $\mathcal{M}$, i.e. there exists some polynomial $p(n)$ such that $\mathcal{M}$ halts on every input of length n after at most $p(n)$ steps.
- PSPACE: The class of languages that can be accepted by some polynomially space bounded deterministic Turing machine.[1]
- EXP: The class of languages that can be accepted by some exponentially time bounded deterministic Turing machine $\mathcal{M}$, i.e. there exists some polynomial $p(n)$ such that $\mathcal{M}$ halts on every input of length n after at most $2^{p(n)}$ steps.
- NEXP: The class of languages that can be accepted by some exponentially time bounded nondeterministic Turing machine.
- $\mathsf{EXPSPACE}$: The class of languages that can be accepted by some exponentially space bounded deterministic Turing machine.[1]
- $\mathsf{2EXP}$: The class of languages that can be accepted by some doubly exponentially time bounded deterministic Turing machine $\mathcal{M}$, i.e. there exists some polynomial $p(n)$ such that $\mathcal{M}$ halts on every input of length n after at most $2^{2^{p(n)}}$ steps.

[1] From Savitch's theorem [99] it directly follows that PSPACE (resp. $\mathsf{EXPSPACE}$) equals the class of languages accepted by some polynomially (resp. exponentially) space bounded *nondeterministic* Turing machine.

- mEXP: The class of languages that can be accepted by some m-fold exponentially time bounded deterministic Turing machine $\mathcal{M}$, i.e. there exists some polynomial $p(n)$ such that $\mathcal{M}$ halts on every input of length n after at most TOWER(m) steps, where TOWER$(0) = p(n)$ and TOWER$(k) = 2^{\text{TOWER}(k-1)}$ for each $k \geq 1$.

Let $\mathcal{C}$ and $\mathcal{D}$ be complexity classes. By $\mathcal{C}^{\mathcal{D}}$ we denote the class of all problems that can be decided by some Turing machine of the same sort and resource bound as from $\mathcal{C}$ that additionally has access to an oracle from $\mathcal{D}$. Note that $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$.

Let L_i be a language over some alphabet Σ_i for each $i \in \{1, 2\}$. We say that L_1 is *logspace reducible* (resp. *polytime reducible*) to L_2 whenever there exists function $f : \Sigma_1^* \to \Sigma_2^*$ that can be computed by a logarithmically space bounded (resp. polynomially time bounded) Turing machine such that additionally for each $w \in \Sigma_1^*$ we have $w \in L_1$ if and only if $f(w) \in L_2$. Let $\mathcal{C}$ be a complexity class and L a language. We say that L is *$\mathcal{C}$-hard with respect to logspace reductions (resp. polytime reductions)* whenever each language from $\mathcal{C}$ is logspace reducible (resp. polytime reducible) to L. A language L is *$\mathcal{C}$-complete* whenever L is in $\mathcal{C}$ and L is $\mathcal{C}$-hard. Whenever we do not explictly name the reduction type, we mean polytime reductions.

Beyond decidability, we denote by Σ_1^1 the class of languages definable by some existential second-order formula of arithmetic over the naturals, see also [96].

Alternation

For proving lower bounds alternating Turing machines will turn out to be useful. An *alternating Turing machine (ATM)* is a tuple $\mathcal{M} = (Q, \Gamma, \Sigma, q_0, \delta, \Box)$, where

- $Q = Q_\forall \uplus Q_\exists$ is a finite set of *states*, which is partitioned into *universal states* $Q_\forall$ and *existential states* $Q_\exists$,
- Γ is a finite *tape alphabet*,
- $\Sigma \subseteq \Gamma$ is a finite *input alphabet*,
- $q_0 \in Q$ is the *initial state*,
- the function $\delta : Q \times \Gamma \to (\text{Moves} \times \text{Moves}) \uplus \{\bot\}$ with $\text{Moves} = Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ assigns to every pair $(q, a) \in Q \times \Gamma$ either a pair of *moves* or $\bot$, and
- $\Box \in \Gamma \setminus \Sigma$ is a distinguished *blank symbol*.

Configurations of $\mathcal{M}$ are elements from $\Gamma^* Q \Gamma^+$. Hence, each configuration $wqaw'$ (with $w, w' \in \Gamma^*, q \in Q$, and $a \in \Gamma$) either has two successor configurations (whenever $\delta(q, a) \in \text{Moves} \times \text{Moves}$) or none (whenever $\delta(q, a) = \bot$).

Definition 2.1 *A* computation tree *T of a configuration u of $\mathcal{M}$ that is in current state $q \in Q$ and scans a symbol $a \in \Gamma$ is a tree, where each node is labeled with a configuration of $\mathcal{M}$, such that the root is labeled with u and exactly one of the following three conditions holds:*

(CT a) $\delta(q, a) = \bot$ *and T is a singleton, i.e. T consists of the root only.*

(CT b) $q \in Q_\forall$, $\delta(q,a) \neq \bot$, *i.e.* u *has two successor configurations* u_1 *and* u_2*, and* T *consists of the root* u *together with two subtrees* T_1 *and* T_2*, where* T_1 *(resp.* T_2*) is some computation tree of* u_1 *(resp.* u_2*).*

(CT c) $q \in Q_\exists$, $\delta(q,a) \neq \bot$ *and* T *consists of the root* u *together with one subtree* T'*, where* T' *is a computation tree of one of the two successor configurations of* u*.*

The height $h(T) \in \mathbb{N} \cup \{\infty\}$ *of a computation tree* T *is the supremum of all distances from leaves of* T *to the root of* T*.* □

An *acceptance tree* is a finite computation tree where every leaf is labeled by some universal configuration. We call a configuration of $\mathcal{M}$ *accepting* if it has an acceptance tree. The *language* of $\mathcal{M}$ is defined as

$$L(\mathcal{M}) \quad = \quad \{w \in \Sigma^* \mid \text{the initial configuration of } \mathcal{M} \text{ on } w \text{ is accepting}\}.$$

For a function $f : \mathbb{N} \to \mathbb{N}$ we say that $\mathcal{M}$ is f*-time bounded* (resp. f*-space bounded*) if for every $w \in \Sigma^*$ and every computation tree T on $\mathcal{M}$'s initial configuration on w we have $h(T) \leq f(|w|)$ (resp. $|u| \leq f(|w|)$ for every configuration u that appears in T).

Let AL, AP, APSPACE, AEXP, and AEXPSPACE be the alternating analogs of the complexity classes L, P, PSPACE, EXP, and EXPSPACE respectively, i.e. AL denotes the class of languages that can be accepted by some logarithmically space bounded ATM, and so on. By [19] it is well-known that the following equalities hold,

- AL = P,
- AP = PSPACE,
- APSPACE = EXP,
- AEXP = EXPSPACE, and
- AEXPSPACE = 2EXP.

2.3 Propositional Dynamic Logic (PDL)

In this section we introduce Propositional Dynamic Logic (PDL) of Fischer and Ladner [36]. After defining the syntax and semantics of PDL, we restate some well-known results concerning the model theory and the complexity of problems related to PDL.

Syntax

In the rest of this thesis, let us fix a countable set of *atomic propositions* $\mathbb{P}$ and a countable set of *atomic programs* $\mathbb{A}$.

Definition 2.2 Propositional Dynamic Logic (PDL) *consists of two syntactic entities that are mutually connected: The set of* formulas $\mathcal{F}$ *and the set of* programs $\mathcal{P}$ *of PDL are defined to be the smallest sets that satisfy the following conditions:*

(1) $\mathbb{P} \subseteq \mathcal{F}$,

(2) $\mathbb{A} \subseteq \mathcal{P}$,

(3) *if* $\varphi \in \mathcal{F}$, *then* $\neg\varphi \in \mathcal{F}$,

(4) *if* $\varphi_1, \varphi_2 \in \mathcal{F}$, *then* $\varphi_1 \vee \varphi_2 \in \mathcal{F}$,

(5) *if* $\pi \in \mathcal{P}$ *and* $\varphi \in \mathcal{F}$, *then* $\langle\pi\rangle\varphi \in \mathcal{F}$,

(6) *if* $\varphi \in \mathcal{F}$, *then* $\varphi? \in \mathcal{P}$,

(7) *if* $\pi_1, \pi_2 \in \mathcal{P}$, *then* $\pi_1 \cup \pi_2 \in \mathcal{P}$,

(8) *if* $\pi_1, \pi_2 \in \mathcal{P}$, *then* $\pi_1 \circ \pi_2 \in \mathcal{P}$, [2] *and*

(9) *if* $\pi \in \mathcal{P}$, *then* $\pi^* \in \mathcal{P}$. □

We introduce the following abbreviations,

$$\begin{array}{rcl}
\texttt{true} & = & p \vee \neg p \quad \text{for some } p \in \mathbb{P}, \\
\texttt{false} & = & \neg\texttt{true}, \\
\varphi_1 \wedge \varphi_2 & = & \neg(\neg\varphi_1 \vee \neg\varphi_2), \\
\varphi_1 \rightarrow \varphi_2 & = & \neg\varphi_1 \vee \varphi_2, \\
\varphi_1 \leftrightarrow \varphi_2 & = & (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1), \\
[\pi]\varphi & = & \neg\langle\pi\rangle\neg\varphi, \\
\langle \mathsf{A} \rangle\varphi & = & \bigcup_{a \in \mathsf{A}} a \quad \text{for any finite } \mathsf{A} \subseteq \mathbb{A}, \\
\pi^+ & = & \pi \circ \pi^*, \text{ and} \\
\bigcirc_{i=1}^{n}\pi_i & = & \pi_1 \circ \cdots \circ \pi_n \quad \text{for programs } \pi_1, \ldots, \pi_n.
\end{array}$$

Formulas of the kind $\langle\pi\rangle\varphi$ are called *diamond formulas*, whereas formulas of the kind $[\pi]\varphi$ are called *box formulas*. *Test programs* are programs of the kind $\varphi?$, where $\varphi \in \mathcal{F}$ is a formula. *Test-free PDL* is the fragment of PDL that disallows rule (6) of Definition 2.2. *Modal logic* is the fragment of PDL that disallows rules (6) to (9) of Definition 2.2.

Semantics

Formulas and programs are interpreted on Kripke structures, which we introduce next. A *Kripke structure* is a tuple $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\})$, where

- W is a set of *worlds*, [3]

[2] Originally, the composition operator was denoted by ";". Instead we use the symbol "∘" since it bears its actual semantics in mind, namely the composition of relations.

[3] Worlds of Kripke structures are also known as *states* but we do not use this notion in order to avoid confusion with states of automata.

- $\to_a \subseteq W \times W$ is a *transition relation* for each $a \in \mathbb{A}$, and
- $W_p \subseteq W$ is a set of worlds for each $p \in \mathbb{P}$.

We call K *deterministic* if additionally $\to_a$ is a partial function for each $a \in \mathbb{A}$, i.e. $x \to_a y$ and $x \to_a y'$ implies $y = y'$ for each $x, y, y' \in W$ and for each $a \in \mathbb{A}$. We call K a *tree* if

- $W \subseteq S^*$ is a non-empty prefix-closed subset of some set S,
- $\bigcup_{a\in\mathbb{A}} \to_a = \{(x, xs) \in W \times W \mid x \in S^*, s \in S\}$,
- $\to_a \cap \to_b = \emptyset$ for each $a, b \in \mathbb{A}$ with $a \neq b$.

We often write Kripke structures as tuples $(W, \{\to_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$, where $\mathsf{A} \subseteq \mathbb{A}$ and $\mathsf{P} \subseteq \mathbb{P}$ are finite subsets, with the meaning that $\to_a = \emptyset$ for each $a \in \mathbb{A} \setminus \mathsf{A}$ and $W_p = \emptyset$ for each $p \in \mathbb{P} \setminus \mathsf{P}$.

For each PDL formula $\varphi \in \mathcal{F}$ and each PDL program $\pi \in \mathcal{P}$, we inductively define the semantics $[\![\varphi]\!]_K \subseteq W$ and $[\![\pi]\!]_K \subseteq W \times W$ as follows,

$$\begin{aligned}
[\![p]\!]_K &= W_p \quad \text{for each } p \in \mathbb{P},\\
[\![a]\!]_K &= \to_a \quad \text{for each } a \in \mathbb{A},\\
[\![\neg\varphi]\!]_K &= W \setminus [\![\varphi]\!]_K,\\
[\![\varphi_1 \vee \varphi_2]\!]_K &= [\![\varphi_1]\!]_K \cup [\![\varphi_2]\!]_K,\\
[\![\langle\pi\rangle\varphi]\!]_K &= \{x \in W \mid \exists y \in [\![\varphi]\!]_K : (x,y) \in [\![\pi]\!]_K\},\\
[\![\varphi?]\!]_K &= \{(x,x) \mid x \in [\![\varphi]\!]_K\},\\
[\![\pi_1 \cup \pi_2]\!]_K &= [\![\pi_1]\!]_K \cup [\![\pi_2]\!]_K,\\
[\![\pi_1 \circ \pi_2]\!]_K &= [\![\pi_1]\!]_K \circ [\![\pi_2]\!]_K, \text{ and}\\
[\![\pi^*]\!]_K &= [\![\pi]\!]_K^*.
\end{aligned}$$

Example 2.3 *The following equalities hold for all PDL formulas $\varphi, \varphi_1, \varphi_2$, all PDL programs π, π_1, π_2, and all Kripke structures K,*

$$\begin{aligned}
[\![\varphi_1 \wedge \varphi_2]\!]_K &= [\![\langle\varphi_1?\rangle\varphi_2]\!]_K,\\
[\![\langle\pi_1 \cup \pi_2\rangle\varphi]\!]_K &= [\![\langle\pi_1\rangle\varphi \vee \langle\pi_2\rangle\varphi]\!]_K, \textit{ and}\\
[\![[\pi^*]\varphi]\!]_K &= [\![\varphi \wedge [\pi^+]\varphi]\!]_K.
\end{aligned}$$ □

Since we study the computational complexity of problems related to PDL, we have to define the size of formulas and programs.

Definition 2.4 *The* size $|\varphi|$ *of each formula φ and the size $|\pi|$ of each program π is inductively defined as follows,*

- $|p| = |a| = 1$ *for each $p \in \mathbb{P}$ and each $a \in \mathbb{A}$,*

- $|\neg\varphi| = |\varphi| + 1$,
- $|\varphi_1 \vee \varphi_2| = |\varphi_1| + |\varphi_2| + 1$,
- $|\langle\pi\rangle\varphi| = |\pi| + |\varphi| + 1$,
- $|\varphi?| = |\varphi| + 1$,
- $|\pi_1 \cup \pi_2| = |\pi_1 \circ \pi_2| = |\pi_1| + |\pi_2| + 1$, *and*
- $|\pi^*| = |\pi| + 1$. □

Satisfiability and model checking

We say that a Kripke structure K with world set W is a *model* of a formula φ, if $x \in [\![\varphi]\!]_K$ for some $x \in W$. A formula is *satisfiable* if it has a model. The following two algorithmic problems are discussed in this thesis.

- The *satisfiability problem* is the question, given a formula φ, whether φ is satisfiable.
- The *model checking problem* is the question, given a Kripke structure K, some world x of K, and a formula φ, whether $x \in [\![\varphi]\!]_K$.

Concerning the model checking problem, we have to be precise how to measure the size of the Kripke structure. If we do not comment on the representation of the Kripke structure, we assume that the Kripke structure is finite and given explicitly, say by an adjacency matrix.

Our investigation of the complexity of the model checking problem follows Vardi's methodology from [119]. For a class of (finitely presented but potentially infinite) Kripke structures $\mathcal{C}$, we distinguish three different ways of measuring the complexity of the model checking problem for $\mathcal{C}$:

(1) One may fix a formula φ and consider the complexity of verifying for a given Kripke structure $K \in \mathcal{C}$ and some world x of K whether $x \in [\![\varphi]\!]_K$; thus, only the Kripke structure and the world belong to the input (*data complexity*).

(2) One may fix a Kripke structure $K \in \mathcal{C}$ and a world x of K and consider the complexity of verifying for a given formula φ, whether $x \in [\![\varphi]\!]_K$ holds; thus, only the formula belongs to the input (*expression complexity*).

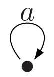
(3) The Kripke structure, the world, and the formula all belong to the input (*combined complexity*).

Let us recall some well-known results on the model theory of PDL and the complexity of the satisfiability and the model checking problem of PDL.

Using filtration, Fischer and Ladner were able to prove the following theorem.

Theorem 2.5 ([36]) *Every satisfiable PDL formula has a finite model.* □

Furthermore, it is well-known that every satisfiable PDL formula has a tree model since every Kripke structure is bisimilar to its *tree unraveling* and PDL cannot distinguish bisimilar Kripke structures. However, note that the formula $[a^*]\langle a\rangle\mathtt{true}$ does not have a *finite* tree model but, among others, the following two models:



and

The upper bound of the following theorem is due to Pratt and the lower bound due to Fischer and Ladner.

Theorem 2.6 ([36, 92]) *Satisfiability in PDL is* EXP*-complete.* □

The following is known about the complexity of the model checking problem of PDL.

Theorem 2.7 ([71]) *The combined complexity of the model checking problem of PDL is* P*-complete.* □

Note that in [71], Lange proved membership in P of the model checking problem for numerous extensions of PDL such as converse, complementation, context-free programs and many more. Several of these extensions are discussed in the next section.

2.4 PDL extensions

The aim of this section is to acquaint the reader with different extensions of Propositional Dynamic Logic. We give a résumé on different PDL extensions, discuss their model theory and investigate the complexity of the satisfiability and the model checking problem.

Succinct representations

Note that each PDL program π can be seen as a regular language $L(\pi)$ when interpreting π as a regular expression over the finite set $\{a \in \mathbb{A} \mid a \text{ occurs in } \pi\} \cup \{\varphi? \mid \varphi? \text{ occurs in } \pi\}$. The corresponding PDL semantics $[\![\pi]\!]_K$ is defined as follows: $(x, y) \in [\![\pi]\!]_K$ if and only if there exists some word $w = a_1 \cdots a_n \in L(\pi)$ and worlds $z_0, z_1, \ldots, z_n$ such that $z_0 = x$, $z_n = y$, and the following holds for each $i \in [n]$,

- if $a_i = \psi?$, then $z_{i-1} = z_i$ and $z_i \in [\![\psi]\!]_K$, and
- if $a_i \in \mathbb{A}$, then $z_{i-1} \rightarrow_{a_i} z_i$.

In *APDL*, each program is given by a nondeterministic finite automaton (NFA). Since NFAs are exponentially more succinct than regular expressions [27] the complexity of satisfiability in APDL may increase. However, the following theorem states that the latter is not the case.

Theorem 2.8 ([93]) *Satisfiability in APDL is* EXP*-complete.* □

Another means of extending PDL is by adding to $\cup$, $\circ$ and $*$, the *interleaving operator* $|$ (also known as *shuffle product*) on regular expressions, where the language $L(\pi_1|\pi_2)$ is defined as

$$L(\pi_1|\pi_2) \quad = \quad \{v_1 w_1 \cdots v_n w_n \mid v_1 \cdots v_n \in L(\pi_1) \wedge w_1 \cdots w_n \in L(\pi_2)\}.$$

Mayer and Stockmeyer investigated *PDL with interleaving (PDL$^{|}$)*, i.e. where programs are succinctly represented by regular expressions with the interleaving operator.

Theorem 2.9 ([82]) *Satisfiability in PDL$^{|}$ is* 2EXP*-complete.* $\square$

Remarkably, 2EXP-hardness already holds for PDL$^{|}$ formulas where each occurring program has the form $\pi_1 \mid \cdots \mid \pi_k$ for programs π_i that each only contain the operators union, composition and Kleene star. Recall that by Theorem 2.7 the combined complexity of the model checking problem for numerous PDL extensions was proven to be in P. In contrast to these tractable PDL extensions, the following proposition states that model checking PDL$^{|}$ is harder, namely complete for PSPACE.

Proposition 2.10 *The combined complexity of PDL$^{|}$ model checking is* PSPACE*-complete.* $\square$

PROOF For the upper bound, let φ be a PDL$^{|}$ formula over the atomic programs A and over the atomic propositions P and let $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$ be a Kripke structure. Recall that $\mathsf{PSPACE} = \mathsf{P}^{\mathsf{PSPACE}}$. We give a $\mathsf{P}^{\mathsf{PSPACE}}$ procedure that computes in a bottom-up manner

(1) for each subformula ψ of φ the set $[\![\psi]\!]_K \subseteq W$ and

(2) for each program π such that $\langle\pi\rangle\psi$ is a subformula of φ the set $[\![\pi]\!]_K \subseteq W \times W$.

Point (1): Computing $[\![\psi]\!]_K$ can be done efficiently when we have inductively already computed $[\![\theta]\!]_K$ for each subformula θ of ψ and, in case $\psi = \langle\pi\rangle\theta$ is a diamond formula, the set $[\![\pi]\!]_K$.

Point (2): Inductively, we have already computed $[\![\theta]\!]_K$ for each test program $\theta?$ that appears in π. Computing $[\![\pi]\!]_K$ boils down to deciding $(x_1, x_2) \in [\![\pi]\!]_K$ in PSPACE for each $x_1, x_2 \in W$. Firstly, we can associate with each regular expression with interleaving ρ the NFA $\mathcal{A}(\rho)$ inductively by the syntactic structure of ρ as follows: If the outermost operator of ρ is $\cup, \circ$, or $*$, then $\mathcal{A}(\rho)$ is defined by the standard automata-theoretic construction for union, composition or Kleene star, respectively; if $\rho = \rho_1|\rho_2$, then $\mathcal{A}(\rho)$ is the *asynchronous* product automaton of $\mathcal{A}(\rho_1)$ and $\mathcal{A}(\rho_2)$. Let $\mathcal{A}(\pi) = (Q, \Sigma, q_0, \delta, F)$ be the NFA for π when π is treated as a regular expression with interleaving over the alphabet $\Sigma = \mathsf{A} \cup \{\theta? \mid \theta? \text{ occurs in } \pi\}$. Without loss of generality let us assume that $F = \{q_f\}$ is a singleton. Next, observe that we have $(x_1, x_2) \in [\![\pi]\!]_K$ if and only if there exists a path from (q_0, x_1) to (q_f, x_2) in the directed graph $G = (V, E)$ with vertex set V and edge set E, where $V = Q \times W$ and

$$E \quad = \quad \{((q,y),(p,z)) \in (Q \times W)^2 \mid \exists b \in \Sigma : p \in \delta(q,b) \wedge (y,z) \in [\![b]\!]_K\}.$$

Note that the size of $\mathcal{A}(\pi)$ may be exponential in the size of π; consequently the size of G may be exponential in the size of π. However, since we only need space linear in $|\pi| + |W|$ for representing each node of G, we can decide reachability in G in space quadratic in $|\pi| + |W|$.

Thus, we can decide $(x_1, x_2) \in [\![\pi]\!]_K$ for each $x_1, x_2 \in W$ in PSPACE and hence compute $[\![\pi]\!]_K$ in $\mathsf{P}^{\mathsf{PSPACE}}$.

Let us prove the PSPACE lower bound. In [64], the following problem was proven to be complete for PSPACE:

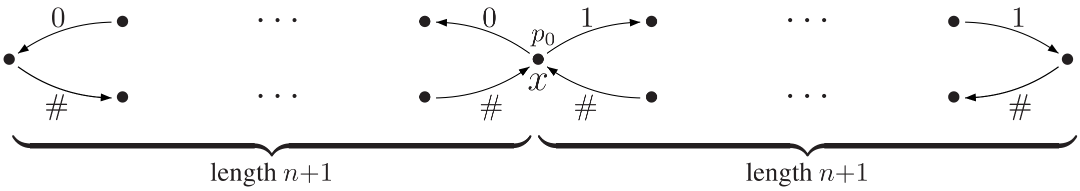
INPUT: Regular expressions without interleaving $(\rho_i)_{i\in[n]}$ over the alphabet $\{0,1\}$ for $n \geq 1$.

QUESTION: $\bigcap_{i\in[n]} L(\rho_i) \neq \emptyset$?

We give a polytime reduction from the latter problem to the model checking problem for $\mathrm{PDL}^{|}$. Let $(\rho_i)_{i\in[n]}$ be regular expressions. Define the homomorphism $h : \{0,1\} \to \{0,1,\#\}^*$ as $h(a) = a^{n+1}\#^{n+1}$ for each $a \in \{0,1\}$. Define $h(\rho_i)$ to be the regular expression that is obtained from ρ_i by replacing each occurrence of a letter $a \in \{0,1\}$ in ρ_i by $h(a)$ for each $i \in [n]$. By Point (4) of Lemma 7.2 in [82] the following equivalence holds for each $w \in \{0,1\}^*$,

$$w \in \bigcap_{i\in[n]} L(\rho_i) \quad \Leftrightarrow \quad h(w) \in L\Big(h(\rho_1) \mid h(\rho_2) \mid \cdots \mid h(\rho_n)\Big).$$

Let K be the following Kripke structure, where only the world x satisfies the atomic proposition p_0:



To complete our reduction, it directly follows

$$\bigcap_{i\in[n]} L(\rho_i) \neq \emptyset \quad \Leftrightarrow \quad x \in [\![\, \langle h(\rho_1) \mid h(\rho_2) \mid \cdots \mid h(\rho_n)\rangle p_0 \,]\!]_K.$$

■

Converse

The logic *CPDL* extends PDL by additionally allowing the converse $\overline{\pi}$ of programs π. The semantics is defined as

$$[\![\overline{\pi}]\!]_K \quad = \quad \{(y,x) \mid (x,y) \in [\![\pi]\!]_K\}.$$

Vakarelov proved via filtration that CPDL has the finite model property.

Theorem 2.11 ([113]) *Every satisfiable CPDL formula has a finite model.* □

But one the other hand, when we restrict ourselves to deterministic models, one can enforce infinite models in CPDL.

Proposition 2.12 *There exists a CPDL formula that has a deterministic model but not any deterministic model that is finite.* □

PROOF Every deterministic model of the formula

$$\neg\langle a\rangle\texttt{true} \wedge [\overline{a}^*]\langle\overline{a}\rangle\texttt{true}$$

contains an infinite backward ray:

$$\bullet \xleftarrow[a]{} \bullet \xleftarrow[a]{} \bullet \xleftarrow[a]{} \bullet \quad \cdots$$

■

Using automata-theoretic techniques, Vardi identified the complexity of satisfiability in CPDL.

Theorem 2.13 ([120]) *Satisfiability in CPDL is* EXP*-complete.* □

The previous theorem also follows from a natural translation from CPDL to PDL found by De Giacomo.

Theorem 2.14 ([39]) *There exists a satisfiability preserving translation from CPDL to PDL computable in polynomial time.* □

Negation and Intersection

The logic *NPDL* extends PDL by allowing programs of the kind $\neg\pi$. The semantics is defined as

$$[\![\neg\pi]\!]_K \quad = \quad W \times W \setminus [\![\pi]\!]_K.$$

Recall that Σ_1^1 denotes the class of languages definable by some existential second-order formula of arithmetic over the naturals.

Theorem 2.15 ([53]) *Satisfiability in NPDL is Σ_1^1-complete.* □

Restricting NPDL by allowing only negation of *atomic programs* yields PDL$^{(\neg)}$. The following is known about the complexity of satisfiability in PDL$^{(\neg)}$.

Theorem 2.16 ([80]) *Satisfiability in PDL$^{(\neg)}$ is* EXP*-complete.* □

Since adding full negation of programs leads to undecidability, one can consider the fragment IPDL of NPDL that allows the intersection of programs, i.e. IPDL extends PDL by allowing programs of the kind $\pi_1 \cap \pi_2$, where

$$[\![\pi_1 \cap \pi_2]\!]_K \quad = \quad [\![\pi_1]\!]_K \cap [\![\pi_2]\!]_K.$$

The following two propositions show that the model theory of IPDL is more involved.

Proposition 2.17 ([53]) *There exists a satisfiable IPDL formula that does not have a finite model.* □

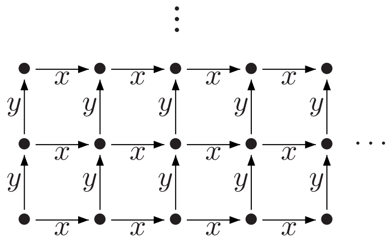


Figure 2.1: The infinite grid over deterministic programs x and y.

PROOF Every model of the formula

$$[a^*]\left(\langle a\rangle\texttt{true} \wedge \neg\langle a^+ \cap \texttt{true?}\rangle\texttt{true}\right).$$

contains the infinite ray:

■

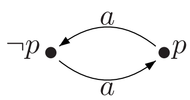
The following proposition is taken from [78].

Proposition 2.18 ([78]) *There exists a satisfiable IPDL formula that does not have a tree model.* □

PROOF The formula

$$\neg p \wedge \neg\langle b\rangle\texttt{true} \wedge \langle (a \circ p? \circ a) \cap b^*\rangle\texttt{true}$$

enforces a cycle of length 2:



■

Via a reduction from the recurring tiling problem, Harel proved the following theorem.

Theorem 2.19 ([52]) *It is Σ^1_1-complete to decide whether a given IPDL formula has a deterministic model.* □

Note that restriction to deterministic Kripke structures is decisive for the high undecidability status of IPDL: Over any two atomic deterministic programs x and y the formula

$$[(x \cup y)^*]\ \Big(\neg(\langle (x \cup y)^+ \cap \texttt{true?}\rangle\texttt{true}) \wedge \neg\langle x \cap y\rangle\texttt{true} \wedge \langle x \circ y \cap y \circ x\rangle\texttt{true}\Big)$$

enforces the infinite grid as substructure, see also Figure 2.1. Hence it can be seen how to encode computations of a Turing machine with an undecidable membership problem.

However, when we do not restrict ourselves to deterministic Kripke structures, the following upper bound for satisfiability in IPDL was proven by Danecki.

Theorem 2.20 ([22]) *Satisfiability in IPDL is in* $2\mathsf{EXP}$. □

A matching lower bound for IPDL was found over twenty years later by Lange and Lutz.

Theorem 2.21 ([72]) *Satisfiability in IPDL is* $2\mathsf{EXP}$*-hard. Moreover, deciding whether an IPDL formula has a tree model is* $2\mathsf{EXP}$*-hard.* □

Although Danecki's paper contains admireable ideas [22], one must admit that the proofs in his conference version are rather hard to verify fully and numerous details are missing. Moreover it is unclear how to extend his techniques to *ICPDL*, the extension of IPDL with the converse of programs. For ICPDL, the following upper bound was known so far.

Theorem 2.22 ([78]) *Satisfiability in ICPDL is decidable.* □

Theorem 2.22 is shown in [78] via a reduction to the monadic second order theory of the infinte binary tree and hence yields a non-elementary complexity upper bound. In Chapter 3 we prove that satisfiability of μ-ICPDL$_{\mathtt{Nom}}$ is $2\mathsf{EXP}$-complete (Theorem 3.28) . Besides intersection and converse, the logic μ-ICPDL$_{\mathtt{Nom}}$ allows the usage of fixed points and nominals. Nominals are distinguished atomic propositions that hold in exactly one world of a Kripke structure. Thus, μ-ICPDL$_{\mathtt{Nom}}$ is the extension of ICPDL and the modal μ-calculus that additionally allows the usage of nominals. The model theory that we employ for proving a $2\mathsf{EXP}$ upper bound for satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is that every satisfiable μ-ICPDL$_{\mathtt{Nom}}$ formula that uses k different nominals has a countable model of *tree width* at most $k+2$ (Corollary 3.6). Tree width is an established measure from graph theory [26] that roughly describes how tree-like a graph is.

The logic loop-PDL (resp. loop-CPDL) is the extension of PDL (resp. CPDL) that allows the usage of formulas of the kind $\mathrm{loop}(\pi)$, where π is a program, with the semantics

$$[\![\mathrm{loop}(\pi)]\!]_K \quad = \quad \{x \in W \mid (x,x) \in [\![\pi]\!]_K\}.$$

Example 2.23 *Let* $\mathsf{A} \subseteq \mathbb{A}$ *be a finite set of atomic programs and let* $\mathsf{P} \subseteq \mathbb{P}$ *be a finite set of atomic propositions. Moreover, let* $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$ *be a* deterministic *Kripke structure, i.e.* $\rightarrow_a$ *is a partial function for each* $a \in \mathsf{A}$.

We call a world $x \in W$ *a* recovery world *in* K *if for each* $y \in W$ *that is reachable from* x *we have that* x *is also reachable from* y*. Formally, we require that* $(x,y) \in [\![\mathsf{A}]\!]_K^*$ *implies* $(y,x) \in [\![\mathsf{A}]\!]_K^*$ *for all* $y \in W$*. In program verification, the existence of a recovery world can express that the program can always return to the initial state.*

The set of recovery worlds can be defined in loop*-PDL. More precisely,* $x \in W$ *is a recovery world in* K *if and only if* $x \in [\![\varphi]\!]_K$*, where*

$$\varphi \quad = \quad [\mathsf{A}^*] \bigwedge_{a \in \mathsf{A}} \Big(\langle a \rangle \mathtt{true} \;\rightarrow\; \mathrm{loop}(a \circ \mathsf{A}^*) \Big).$$

PROOF "only-if": Assume x is a recovery world. Let $\rightarrow_K = \bigcup_{a \in \mathsf{A}} \rightarrow_a$. We show that every world $y \in W$, with $x \rightarrow_K^* y$, satisfies the formula

$$\bigwedge_{a \in \mathsf{A}} \Big(\langle a \rangle \mathtt{true} \;\rightarrow\; \mathrm{loop}(a \circ \mathsf{A}^*) \Big).$$

Assume $x \to_K^* y$ and $y \in [\![\langle a\rangle\texttt{true}]\!]_K$ for some $a \in \mathsf{A}$. This implies $y \to_a z$ for some world $z \in W$. Since x is a recovery world and $x \to_K^* z$, we have $z \to_K^* x$. Altogether we have $y \to_a z \to_K^* x \to_K^* y$, hence $y \in [\![\mathrm{loop}(a \circ \mathsf{A}^*)]\!]_K$.

"if": Assume $x \in [\![[\mathsf{A}^*] \bigwedge_{a\in\mathsf{A}} (\langle a\rangle\texttt{true} \to \mathrm{loop}(a \circ \mathsf{A}^*))]\!]_K$. We prove that $y \to_K^* x$ holds for each $y \in W$ for which $x \to_K^* y$. We prove that $y \to_K^* x$ holds by induction on the minimal $n \in \mathbb{N}$ such that $x \to_K^n y$. The case $n = 0$ is trivial. Now assume that $n > 0$ and let $x \to_K^{n-1} z \to_a y$ be an arbitrary path from x to y of minimal length n for some intermediate world $z \in W$ and some $a \in \mathsf{A}$. Since $z \in [\![\mathrm{loop}(a \circ \mathsf{A}^*)]\!]_K$ and K is deterministic we have $y \to_K^* z$. By induction hypothesis we have $z \to_K^* x$, hence $y \to_K^* z \to_K^* x$. ■

Note that loop-PDL (resp. loop-CPDL) is a fragment of IPDL (resp. ICPDL) since $\mathrm{loop}(\pi)$ is semantically equivalent to $\langle \pi \cap \texttt{true}?\rangle\texttt{true}$. We will prove that the recovery world property from Example 2.23 is not even expressible in μ-ICPDL$_{\texttt{Nom}}$ (Proposition 3.38) assuming that we do *not* impose any restrictions on the underlying Kripke structures such as e.g. determinism.

Danecki investigated that every satisfiable loop-PDL formula has a so called *cactus* model. Via a suitable encoding of cactus models into an ω-regular tree language, he established the following complexity result for satisfiability in loop-PDL.

Theorem 2.24 ([23]) *Satisfiability in* loop-*PDL is* EXP*-complete.* □

Recall that loop-PDL is a fragment of IPDL and hence of μ-ICPDL$_{\texttt{Nom}}$. In Chapter 3, we generalize Theorem 2.24 in such a way that we introduce a syntactic measure of nestings of the intersection operator for μ-ICPDL$_{\texttt{Nom}}$ formulas – that we call *intersection width* – and prove that satisfiability of μ-ICPDL$_{\texttt{Nom}}$ formulas of constant intersection width is in EXP (Corollary 3.29). Since each loop-PDL (resp. loop-CPDL) formula can easily be translated to some equivalent IPDL (resp. ICPDL) formula of intersection width at most two, it follows an EXP upper bound for satisfiability in loop-PDL (resp. loop-CPDL), see also Corollary 3.30.

Since satisfiability in PDL$^{(\neg)}$ (the extension of PDL with negation of atomic programs) is EXP-complete and satisfiability in μ-ICPDL$_{\texttt{Nom}}$ is 2EXP-complete, the natural question arises how hard satisfiability in μ-ICPDL$_{\texttt{Nom}}$ with negation of atomic programs is. However, via a reduction from the recurring tiling problem, we prove that already satisfiability in IPDL with negation of atomic programs is hard for Σ_1^1 (Theorem 3.37).

Extensions by model manipulation

Another mode of extension is model manipulation: Whether a formula φ holds in a world of a Kripke structure K can depend on whether some modified Kripke structure K' satisfies a certain subformula of φ. We state two such extensions that are related to PDL. A prominent example of such a logic is van Benthem's Sabotage Modal Logic (SML) [115]. Recall that modal logic is the fragment of PDL where programs are restricted to be atomic, i.e. by dropping rules (6) to (9) of Definition 2.2. SML extends modal logic by adding a sabotage operator $\langle - \rangle_a$ via the following formula building rule

if φ is an SML formula, then $\langle - \rangle_a \varphi$ is an SML formula for each $a \in \mathbb{A}$.

For each Kripke structure $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\})$, the semantics of $\langle - \rangle_a \varphi$ is defined as

$$[\![\langle - \rangle_a \varphi]\!]_K \quad = \quad \{x \in W \mid \exists t \in \to_a \text{ such that } x \in [\![\varphi]\!]_{K \setminus (t,a)}\},$$

where for each $a \in \mathbb{A}$ and each $t \in \to_a$ the Kripke structure $K \setminus (t, a)$ is obtained from K by removing the transition t from $\to_a$.

Lemma 6 in [76] states that SML does not have the tree model property. Moreover in [76] it is shown that SML does not have the finite model property. Satisfiability in modal logic is PSPACE-complete, but adding sabotage modalities makes the satisfiability problem dramatically harder.

Theorem 2.25 ([76]) *Satisfiability in SML is undecidable.* □

It is well-known that the combined complexity of model checking in Modal Logic is P-complete, whereas for SML it jumps to PSPACE-completeness.

Theorem 2.26 ([76]) *The combined complexity of model checking in SML is* PSPACE*-complete. However, both expression complexity and data complexity of SML model checking is in* P. □

For a comprehensive investigation of sabotage modal logics, we refer to [97].

Let us introduce Demri's $\mathrm{DLP}^+_{\mathrm{dyn}}$ – an extension of PDL that allows model manipulation. Syntactically, formulas of $\mathrm{DLP}^+_{\mathrm{dyn}}$ are obtained by adding to Definition 2.2 the following two formula building rules:

(10) if $\pi \in \mathcal{P}$ and $\varphi \in \mathcal{F}$, then $\mathbf{perm}(\pi)\varphi, \mathbf{fperm}(\pi)\varphi \in \mathcal{F}$ and

(11) if $\varphi_1, \varphi_2, \varphi_3 \in \mathcal{F}$, then $\mathbf{grant}(\varphi_1, \varphi_2)\varphi_3, \mathbf{revoke}(\varphi_1, \varphi_2)\varphi_3 \in \mathcal{F}$.

The semantics of $\mathrm{DLP}^+_{\mathrm{dyn}}$ is defined over extended Kripke structures. An *extended Kripke structure* is a tuple $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ such that the first three components of K yield a Kripke structure and $P \subseteq W \times W$ is a binary relation that we call *policy set*. For sets of transitions $A, B \subseteq W \times W$ and each operation $\mathrm{op} \in \{\cup, \setminus\}$, we define

$$K\!\upharpoonright (A, B, \mathrm{op}) \quad = \quad (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P \;\mathrm{op}\; (A \times B)).$$

The semantics for $\mathbf{grant}$ and $\mathbf{revoke}$ formulas is defined as follows:

$$\begin{aligned} [\![\mathbf{grant}(\varphi_1, \varphi_2)\varphi_3]\!]_K &= [\![\varphi_3]\!]_{K\upharpoonright([\![\varphi_1]\!]_K, [\![\varphi_2]\!]_K, \cup)} \\ [\![\mathbf{revoke}(\varphi_1, \varphi_2)\varphi_3]\!]_K &= [\![\varphi_3]\!]_{K\upharpoonright([\![\varphi_1]\!]_K, [\![\varphi_2]\!]_K, \setminus)} \end{aligned}$$

Hence $\mathbf{grant}$ (resp. $\mathbf{revoke}$) "modifies" the current policy set by adding (resp. removing) a set of transitions that is specified as the direct product of sets that are definable in $\mathrm{DLP}^+_{\mathrm{dyn}}$ itself. It remains to define the semantics of $\mathbf{perm}$ and $\mathbf{fperm}$ formulas. We do this informally.

- A world $x \in W$ satisfies the formula $\mathbf{perm}(\pi)\varphi$ whenever for some word w in the regular language defined by π there exists a w-labeled path from x to some world $y \in [\![\varphi]\!]_K$ such that each $\mathbb{A}$-labeled transition on this path is in the policy set P.

- A world $x \in W$ satisfies the formula $\mathbf{fperm}(\pi)\varphi$ whenever each path labeled with some word from the regular language defined by π from x to worlds $y \in [\![\varphi]\!]_K$ has the property that all $\mathbb{A}$-labeled transitions on this path are in the policy set P.

Recall that in sabotage formulas of SML the transition that is removed is not specified at all. In $\mathrm{DLP}^+_{\mathrm{dyn}}$, on the other hand, the model modifications *have to be* defined in the logic itself. It will turn out that precisely this fact makes $\mathrm{DLP}^+_{\mathrm{dyn}}$ more manageable with respect to satisfiability. Only until recently, the following complexity upper bound was known for $\mathrm{DLP}^+_{\mathrm{dyn}}$.

Theorem 2.27 ([24]) *Satisfiability in DLP^+_{dyn} is in* 2EXP. □

We will close the complexity gap between EXP-hardness that $\mathrm{DLP}^+_{\mathrm{dyn}}$ inherits from PDL and its membership in 2EXP from the previous theorem. We prove in Chapter 4 that satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is EXP-complete (Theorem 4.5).

Non-regular extensions

In contrast to extending PDL programs by operators whose semantics is defined via the underlying relations, the classical viewpoint is that PDL programs are seen as regular languages over a finite set of test programs and atomic programs. Consequently, a natural way of extending PDL is to take a class of languages $\mathcal{L}$ subsuming the regular languages and consider PDL+$\mathcal{L}$, which we define next.

Definition 2.28 *Let* $\mathsf{A} \subseteq \mathbb{A}$ *be finite and let $\mathcal{L}$ be a class of languages over* A. *The logic PDL+$\mathcal{L}$ extends PDL (Definition 2.2) with the following formula building rule:*

(11) if $L \in \mathcal{L}$ and $\varphi \in \mathcal{F}$, then $\langle L \rangle \varphi \in \mathcal{F}$. □

Clearly, as we are interested in the complexity/decidability status of the satisfiability or model checking problem of PDL+$\mathcal{L}$, we have to come up with a finite representation for each of the languages from $\mathcal{L}$. However, in the following, the concrete representations are not important and details are left to the reader.

A result from Harel and Singermann states that PDL+$\mathcal{L}$ is strictly more expressive than PDL already for an arbitrary singleton language class $\mathcal{L} = \{L\}$ provided L is non-regular.

Theorem 2.29 ([59]) *Let L be an arbitrary non-regular language over a finite subset of $\mathbb{A}$. Then there exists a PDL+$\{L\}$ formula φ such that there is no PDL formula ψ with $[\![\varphi]\!]_K = [\![\psi]\!]_K$ for all Kripke structures K.* □

It is remarkable that undecidability of the satisfiability problem of PDL+$\mathcal{L}$ can be obtained already for a very simple language class $\mathcal{L}$ consisting of one language only.

Theorem 2.30 ([56]) *Let $a^\# b a^\#$ denote the non-regular language $\{a^n b a^n \mid n \geq 0\}$. Satisfiability in PDL+$\{a^\# b a^\#\}$ is Σ^1_1-complete.* □

In [4] Alur and Madhusudan proposed the class of *visibly pushdown languages (VPL)*. A visibly pushdown language is a language accepted by some visibly pushdown automaton (VPA), which is an ε-free pushdown automaton, where the input alphabet is partitioned into three types of symbols: *Calls*, *returns* and *internal symbols*. When reading an input symbol its type determines whether a push, a pop, or an internal action has to be performed. Moreover, VPAs may push or pop at most one stack symbol at a time. Inter alia, it is shown in [4] that VPL is closed under all boolean operations and that universality and inclusion is EXP-complete. A very general result on non-regular PDL extensions is the following one, due to Löding, Lutz, and Serre.

Theorem 2.31 ([75]) *Satisfiability in PDL+VPL is* $2\mathsf{EXP}$*-complete.* □

Note however, that for the latter theorem to hold it is important that each of the occurring visibly pushdown languages that occur in formulas is defined over some common partition of the input alphabet.

Open problem 29 of Demri's habilitation thesis [25] asks what the complexity of satisfiability in PDL+VPL with the converse operator (let us denote it by CPDL+VPL) is. Let us describe CPDL+VPL in more detail. To each CPDL+VPL formula φ one can assign some visibly pushdown partition $\widetilde{\Sigma}$ of $\mathsf{A} \cup \overline{\mathsf{A}}$, where A is some finite alphabet and $\overline{\mathsf{A}} = \{\overline{a} \mid a \in \mathsf{A}\}$ is a disjoint copy of A; moreover every language that occurs in φ is accepted by some visibly pushdown automaton over $\widetilde{\Sigma}$. Such formulas φ are meant to be interpreted over Kripke structures $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$ such that for each $a \in \mathsf{A}$ we have

$$[\![\overline{a}]\!]_K \quad = \quad \{(y, x) \mid (x, y) \in [\![a]\!]_K\}.$$

Let us sketch an undecidability proof of satisfiability in CPDL+VPL by using techniques from [56].

Proposition 2.32 *Satisfiability in CPDL+VPL is undecidable.* □

Proof (sketch) Fix the alphabet $\mathsf{A} = \{a, b\}$ and the following partition $\widetilde{\Sigma}$ of $\Sigma = \mathsf{A} \cup \overline{\mathsf{A}}$:

- Calls: $\{a\}$
- Returns: $\{\overline{a}\}$
- Internal symbols: $\{b, \overline{b}\}$

Consider the following three formulas:

- $\varphi_1 = \neg\langle\overline{a}\rangle\mathtt{true} \wedge \langle ab\rangle\mathtt{true}$
- $\varphi_2 = [\Sigma^* b]\left(\langle b\rangle\mathtt{true} \wedge (\langle\overline{a}^+\rangle\neg\langle\overline{a}\rangle\mathtt{true})\right)$
- $\varphi_3 = [\Sigma^*]\left(\neg\langle\overline{a}\rangle\mathtt{true} \rightarrow \quad \left(\neg\left\langle\{a^i b\overline{a}^j \mid j \geq i+2\}\right\rangle\mathtt{true} \right.\right.$
$$\left.\left. \wedge\, \neg\left\langle\{a^i b\overline{a}^j \mid j \leq i\}\right\rangle\neg\langle\overline{a}\rangle\mathtt{true}\right)\right)$$

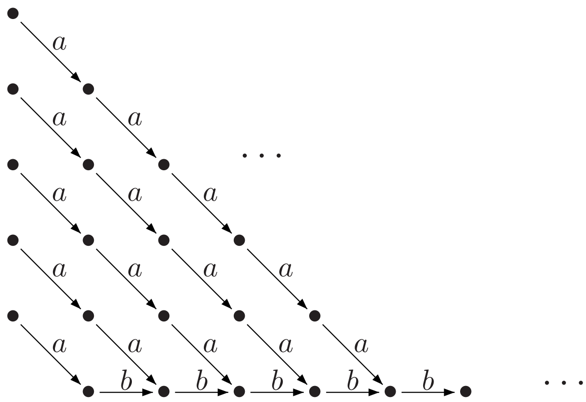


Figure 2.2: A substructure of any model of $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$.

Observe that each of the languages that occur in φ_1, φ_2, and φ_3 can be accepted by some visibly pushdown automaton over $\widetilde{\Sigma}$. It is not hard to see that in any model of $\varphi_1 \wedge \varphi_2$ there exists an infinite sequence of b-transitions such that on every world on this sequence there begins a non-empty finite sequence of inverted a-transitions. The formula φ_3 requires that for every (reachable) world x that has no a-predecessor we have that each path labeled by $a^i b \overline{a}^j$, that begins in x and that ends in some world that also does not have an a-predecessor, satisfies $j = i + 1$. Thus, any model of $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ contains the Kripke structure depicted in Figure 2.2. The arrangement of the worlds of Figure 2.2 is not chosen without reason. Think of worlds that are *not* situated on the lower most path as points of an infinite grid. It requires standard arguments to reduce the tiling problem of the first quadrant of the plane (which is undecidable by [7]) to satisfiability in CPDL+VPL. Tile types are represented by atomic propositions and it has to be guaranteed that each (reachable) "grid world" satisfies exactly one of these. Checking that the tile type of each grid world and its horizontal successor (resp. vertical successor) match the horizontal (resp. vertical) matching relation can easily be accomplished by programs defined by the VPL language $\{a^i b \overline{a}^i \mid i \geq 1\}$ (resp.$\{a^i b \overline{a}^{i+1} \mid i \geq 1\}$) over $\widetilde{\Sigma}$. ■

Recently, k-phase multi-stack visibly pushdown automata (k-MVPAs), a natural extension of VPAs, have been introduced in [112]. A k-MVPA is an automaton equipped with multiple stacks where, again, the action on each stack is determined by the input; more precisely, every input symbol specifies on which stack a push or pop operation or whether an internal operation is done. Moreover, a k-MVPA is restricted to accept only words that can be obtained by concatenating at most k *phases*, where a phase is a sequence of input symbols that invoke pop actions from at most one stack. Note that k-MVPAs with one stack coincide with VPAs. The language class k-MVPL that is described by k-MVPAs has various effective closure properties and a decidable non-emptiness problem. Therefore it is an interesting question to ask if the corresponding extension of PDL is still decidable. This question was raised in [112] and answered negatively by Nowotka and the author.

Theorem 2.33 ([47]) *Satisfiability in PDL+k-MVPL is Σ^1_1-complete and already Σ^1_1-hard for $k = 2$.* □

2.5 Infinite state systems

In this section, we introduce basic process algebras (BPAs), pushdown systems (PDSs) and prefix-recognizable systems (PRSs)[4] as formalisms for finitely representing infinite Kripke structures and, based on this, the infinite state model checking problem of (extensions of) PDL. Besides prefix-recognizable systems, we follow Mayr's classification of infinite state systems in terms of process rewrite systems [84]. The reason that we do not include infinite state systems from [84] that are obtained from rewriting of processes that contain parallel composition is that already model checking test-free PDL over basic parallel processes is undecidable (Corollary 5.2).

Due to technical simplicity, in the context of infinite state model checking, we generally consider Kripke structures without atomic propositions, i.e. Kripke structures are no longer triples but a pairs. Moreover, the only atomic formulas we consider are `true` and `false`. This simplifies notation and can be done without loss of generality since an atomic proposition p can easily be simulated by introducing a reflexive loop via a new atomic program a_p at precisely those worlds that satisfy p. Additionally, we modify any input formula by replacing each occurrence of an atomic proposition p by the formula $\langle a_p\rangle\mathtt{true}$. In the following $\mathsf{A} \subseteq \mathbb{A}$ always denotes a finite set of atomic programs.

Basic process algebras

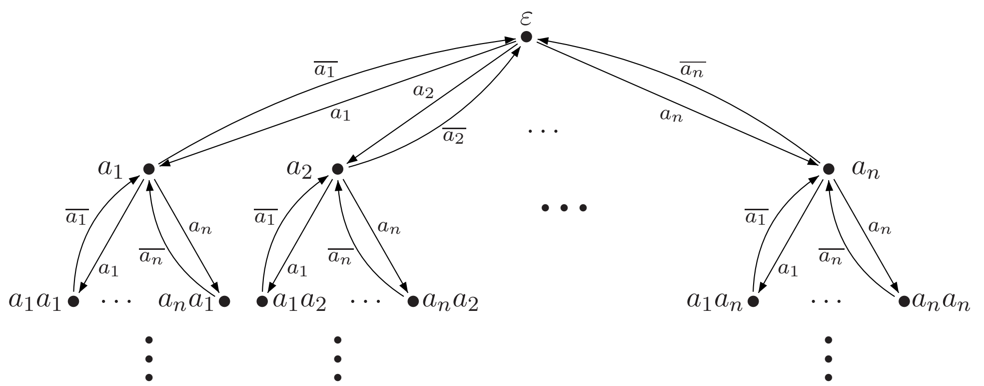
A *basic process algebra (BPA)* is a pair $\mathcal{X} = (\Gamma, \{\xrightarrow{a}_{\mathcal{X}} \mid a \in \mathsf{A}\})$, where Γ is a finite *stack alphabet* and $\xrightarrow{a}_{\mathcal{X}} \subseteq (\Gamma \cup \{\varepsilon\}) \times \Gamma^*$ is a finite set of *rewriting rules* for each $a \in \mathsf{A}$. For better readability, we write $b \xrightarrow{a}_{\mathcal{X}} v$ rather than $(b, v) \in \xrightarrow{a}_{\mathcal{X}}$. The worlds of the Kripke structure defined by $\mathcal{X}$ are the set of words over Γ. For each $a \in \mathsf{A}$ there is an a-labeled transition from a world w to a world w' whenever the rule $\rightarrow_a$ contains a pair (b, v) such that w begins with b and w' is obtained from w by replacing this b by v. Formally, the Kripke structure defined by a BPA $\mathcal{X}$ is $K(\mathcal{X}) = (\Gamma^*, \{\rightarrow_a \mid a \in \mathsf{A}\})$, where

$$\rightarrow_a \quad = \quad \{(bw, vw) \mid w \in \Gamma^*, b \xrightarrow{a}_{\mathcal{X}} v\} \qquad \text{for each } a \in \mathsf{A}.$$

The *size* of a BPA $\mathcal{X}$ is $|\mathcal{X}| = |\Gamma| + \sum_{a\in\mathsf{A}} \sum_{b \xrightarrow{a}_{\mathcal{X}} v} |bv|$.

Example 2.34 *For a finite alphabet $\Gamma = \{a_1, \ldots, a_n\}$ the complete Γ-tree with backward edges is given by the BPA $\mathcal{X} = (\Gamma, \{\xrightarrow{a}_{\mathcal{X}} \mid a \in \mathsf{A}\})$ over $\mathsf{A} = \Gamma \cup \overline{\Gamma}$, where $\xrightarrow{a}_{\mathcal{X}} = \{(\varepsilon, a)\}$ and $\xrightarrow{\overline{a}}_{\mathcal{X}} = \{(a, \varepsilon)\}$ for each $a \in \Gamma$. Figure 2.3 shows $K(\mathcal{X})$.*

[4] PRS, i.e. prefix-recognizable systems, should not be confused with Mayr's process rewrite systems hierarchy from [84].



□

Figure 2.3: The complete $\{a_1, \ldots, a_n\}$-tree with backward edges.

Pushdown systems

A *pushdown system (PDS)* is a tuple $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$ where Q is a finite set of *states*, Γ is a finite *stack alphabet*, and $\xrightarrow{a}_{\mathcal{Y}} \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ is a finite set of *rewriting rules* for each $a \in \mathsf{A}$. Again, instead of writing $(q, b, q', v) \in \xrightarrow{a}_{\mathcal{Y}}$ we use infix notation and write $qb \xrightarrow{a}_{\mathcal{Y}} q'v$. Intuitively, a PDS describes a Kripke structure in which each world is a *configuration* qv of the system, with $q \in Q$ the state and $v \in \Gamma^*$ the stack content. In v, the left-most symbol corresponds to the top of the stack. A rewriting rule $qb \xrightarrow{a}_{\mathcal{Y}} q'v$ says that, travelling a, we can move from any configuration qbw to $q'vw$, i.e. replace b with v on top of the stack and switch the state from q to q'. Formally, the Kripke structure defined by a PDS $\mathcal{Y}$ is $K(\mathcal{Y}) = (Q\Gamma^*, \{\rightarrow_a \mid a \in \mathsf{A}\})$, where

$$\rightarrow_a \quad = \quad \{(qbw, q'vw) \mid w \in \Gamma^*, qb \xrightarrow{a}_{\mathcal{Y}} q'v\} \qquad \text{for each } a \in \mathsf{A}.$$

The *size* of a PDS is $|\mathcal{Y}| = |Q| + |\Gamma| + \sum_{a \in \mathsf{A}} \sum_{qb \xrightarrow{a}_{\mathcal{Y}} q'v} |qbq'v|$.

It is obvious that every BPA $\mathcal{X}$ can be translated into a single-state PDS $\mathcal{Y}$ such that $K(\mathcal{X})$ and $K(\mathcal{Y})$ are isomorphic. Thus, every BPA can be seen as a PDS over a singleton state set.

The following example shows an application of IPDL to the verification of two cooperating pushdown systems.

Example 2.35 *Consider two pushdown systems $\mathcal{Y}_1$ and $\mathcal{Y}_2$ with a common stack alphabet Γ and let $\mathcal{Y}_i = (Q_i, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}_i} \mid a \in \mathsf{A}_i\})$ for each $i \in \{1, 2\}$. It is possible for $\mathcal{Y}_1$ and $\mathcal{Y}_2$ to synchronize via a common configuration, so we assume that $Q_1 \cap Q_2 \neq \emptyset$. To distinguish actions of $\mathcal{Y}_1$ from those of $\mathcal{Y}_2$, we assume that $\mathsf{A}_1 \cap \mathsf{A}_2 = \emptyset$. Finally, let $K(\mathcal{Y}_i)$ be deterministic for $i \in \{1, 2\}$. The two systems can be modeled by a single pushdown system*

$$\mathcal{Y} \quad = \quad (Q_1 \cup Q_2, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}_1} \mid a \in \mathsf{A}_1\} \cup \{\xrightarrow{a}_{\mathcal{Y}_2} \mid a \in \mathsf{A}_2)\}).$$

We want to ensure that, whenever $\mathcal{Y}_1$ and $\mathcal{Y}_2$ are in the same configuration w and $\mathcal{Y}_i$ can reach from w a configuration w_i by an atomic program (from A_i), for $i \in \{1, 2\}$, then the two systems *can again reach a common configuration from w_1 and w_2. Interpreted over $K(\mathcal{Y})$ the latter property can be expressed by the IPDL formula*

$$[\mathsf{A}_1^* \cap \mathsf{A}_2^*] \left(\bigwedge_{a \in \mathsf{A}_1, b \in \mathsf{A}_2} \langle a \rangle \mathtt{true} \wedge \langle b \rangle \mathtt{true} \quad \rightarrow \quad \langle a \circ \mathsf{A}_1^* \cap b \circ \mathsf{A}_2^* \rangle \mathtt{true} \right).$$

Observe that, in the context of model checking PDAs, requiring $K(\mathcal{Y}_i)$ to be determinstic is not a restriction. It is always possible to modify $\mathcal{Y}_i$ such that $K(\mathcal{Y}_i)$ is deterministic by introducing additional atomic programs. □

Prefix-recognizable systems

Let Γ be a finite alphabet. 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 $R_i = \{(uw, vw) \mid u \in U_i, v \in V_i, w \in W_i\}$, where $U_i, V_i, W_i \subseteq \Gamma^*$ are regular languages for each $i \in [n]$. 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)* is a tuple $\mathcal{Z} = (\Gamma, \phi)$ where $\phi : \mathsf{A} \to \mathrm{PRR}(\Gamma)$ assigns to every atomic program $a \in \mathsf{A}$ a prefix-recognizable relation $\phi(a)$ such that $\phi(a) = \bigcup_{i=1}^{n_a} (L(\alpha_i^a) \times L(\beta_i^a))L(\gamma_i^a)$ over Γ which is given by a collection of triples of regular expressions $(\alpha_i^a, \beta_i^a, \gamma_i^a)_{i \in [n_a]}$. A PRS $\mathcal{Z} = (\Gamma, \phi)$ defines the Kripke structure $K(\mathcal{Z}) = (\Gamma^*, \{\phi(a) \mid a \in \mathsf{A}\})$. The *size* $|\mathcal{Z}|$ of a PRS $\mathcal{Z} = (\Gamma, \phi)$ is defined as $|\Gamma| + |\mathsf{A}| + \sum_{a \in \mathsf{A}} \sum_{i=1}^{n_a} |\alpha_i^a| + |\beta_i^a| + |\gamma_i^a|$, where the size of regular expressions is defined in analogy to the size of PDL programs.

Example 2.36 *Let $\mathcal{Z} = (\{0,1\}, \phi)$ over $\mathsf{A} = \{0, 1, \preceq\}$ be the prefix-recognizable system defined as $\phi(a) = (\{\varepsilon\} \times \{a\}) \{0,1\}^*$ for each $a \in \{0,1\}$ and $\phi(\preceq) = (\{\varepsilon\} \times \{0,1\}^*) \{0,1\}^*$. The Kripke structure $K(\mathcal{Z})$ is the complete binary tree with the prefix relation $\preceq$.* □

Note that every PDS $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$ can be translated into an equivalent PRS $\mathcal{Z}(\mathcal{Y}) = (Q \cup \Gamma, \phi)$, where

$$\phi(a) \quad = \bigcup_{qb \xrightarrow{a}_{\mathcal{Y}} q'v} \left(\{qb\} \times \{q'v\}\right) \Gamma^* \qquad \text{for each } a \in \mathsf{A}.$$

Also note that every world of the PRS from Example 2.36 has infinitely many outgoing transitions, whereas every world of every PDS has both finitely many incoming and outgoing transitions.

Comparison to other definitions Note that our definition of basic process algebras allows transitions of the form $\varepsilon \xrightarrow{a}_{\mathcal{X}} v$ and our definition of pushdown systems allows transitions of the form $q \xrightarrow{a}_{\mathcal{Y}} q'v$ where q and q' are states. It is easy to see that our definitions describe exactly the same class of basic process algebras as defined in [83, 84] and the same class of pushdown systems as defined in [125, 14, 124, 84]. Moreover, there are logspace translations between the two formalisms. When defining pushdown systems $\mathcal{Y}$ with state set Q, over atomic programs A *and over* a set of atomic propositions P it is well-established to introduce a mapping

$\lambda : Q \to 2^{\mathsf{P}}$ which assings to every state $q \in Q$ a subset of P. If $p \in \lambda(q)$ for a state q of $\mathcal{Y}$ and $p \in \mathsf{P}$, then all configurations of $\mathcal{Y}$ in state q satisfy p. Given such a mapping $\lambda : Q \to 2^{\mathsf{P}}$ our definition from above can imitate it by introducing fresh transition rules in $\mathcal{Y}$ which describe correspondingly labeled self-loops at exactly those worlds of the Kripke structure $K(\mathcal{Y})$ where the atomic propositions are satisfied. More precisely, we extend the set of atomic programs A by P (assuming P and A to be disjoint) and introduce for all states $q \in Q$ the transition rule $q \xrightarrow{p}_{\mathcal{Y}} q$ if and only if $p \in \lambda(q)$. Similar remarks apply to basic process algebras and prefix-recognizable systems.

3

ICPDL with fixed points and nominals

In this chapter we study μ-ICPDL$_{\mathtt{Nom}}$ which extends ICPDL by fixed point operators and nominals. Our main result is that satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is 2EXP-complete which matches the 2EXP lower bound for IPDL (Theorem 3.37). For the fragment ICPDL of μ-ICPDL$_{\mathtt{Nom}}$ the previously best known upper bound was non-elementary (Theorem 2.22).

Section 3.1 introduces μ-ICPDL$_{\mathtt{Nom}}$ and an infinitary logic ICPDL$^{\infty}_{\mathtt{Nom}}$ that extends ICPDL by infinitary disjunction and allows the usage of nominals. It is shown that for each cardinal κ and each μ-ICPDL$_{\mathtt{Nom}}$ formula φ there exists an ICPDL$^{\infty}_{\mathtt{Nom}}$ formula φ_κ such that φ and φ_κ are equivalent with respect to all Kripke structures of cardinality at most κ.

In Section 3.2 we show via ICPDL$^{\infty}_{\mathtt{Nom}}$ that every satisfiable μ-ICPDL$_{\mathtt{Nom}}$ formula that uses at most k different nominals has a countable model of tree width at most $k+2$. To obtain this result we define for each finite set of nominals M an operator $\# = \#(M)$ on pointed Kripke structures and prove that for any pointed Kripke structure[1] K the pointed Kripke structure $K^{\#}$ satisfies the same μ-ICPDL$_{\mathtt{Nom}}$ formulas with nominals from M and moreover $K^{\#}$ has tree width at most $|M| + 2$.

In Section 3.3 we introduce a special reasoning problem called *ω-regular tree satisfiability* in μ-ICPDL$_{\mathtt{Nom}}$. The ω-regular tree satisfiability problem asks, given some two-way alternating parity tree automaton $\mathcal{T}$ over infinite trees (that naturally defines a set of infinite tree Kripke structures) and some μ-ICPDL$_{\mathtt{Nom}}$ formula φ interpreted on infinite trees, whether there exists some tree $T \in L(\mathcal{T})$ such that φ holds in the root of T. We reduce general satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ to ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ as follows: For an input μ-ICPDL$_{\mathtt{Nom}}$ formula φ over a set of nominals M, we compute a two-way alternating parity tree automaton $\mathcal{T}$ that accepts encodings of all countable Kripke structures of tree width at most $|M| + 2$ and some μ-ICPDL$_{\mathtt{Nom}}$ formula φ' interpreted on infinite trees such that φ is satisfiable if and only if there exists some tree $T \in L(\mathcal{T})$ such that φ' holds in the root of T.

[1] A *pointed* Kripke structure is a Kripke structure K together with some world of K.

In Section 3.4 we prove that ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is 2EXP-complete. For the 2EXP upper bound we give a reduction to the non-emptiness problem of two-way alternating parity tree automata. Furthermore, we investigate a syntactic measure on μ-ICPDL$_{\mathtt{Nom}}$ formulas, namely *intersection width*, which the complexity of ω-regular tree satisfiability is doubly exponential in. As a corollary we obtain that both ω-regular tree satisfiability and general satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ of formulas of constant intersection width is in EXP.

Alongside 2EXP-completeness of general satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ (Section 3.5) via translation to ω-regular tree satisfiability, we prove that ω-regular tree satisfiability leads to a further application, namely infinite state model checking. In Section 3.6 it is shown that for μ-ICPDL$_{\mathtt{Nom}}$ the model checking problem over prefix-recognizable systems can easily be reduced to ω-regular tree satisfiability. We proceed by proving lower bounds for infinite state model checking (fragments of) μ-ICPDL$_{\mathtt{Nom}}$. More precisely, it is shown that the data complexity already both for loop-PDL and for test-free IPDL over basic processes algebras is EXP-hard. This is rather surprising since the data complexity of μ-calculus model checking over basic process algebras is in P [125]. Moreover, we show that there exists a fixed basic process algebra such that model checking test-free IPDL is hard for 2EXP.

Due to decidability of μ-ICPDL$_{\mathtt{Nom}}$ on the one hand and decidability of PDL with atomic program negation on the other, we discuss in Section 3.7 the question whether μ-ICPDL$_{\mathtt{Nom}}$ with atomic program negation is decidable. It turns out that already IPDL with atomic program negation is highly undecidable, namely hard for Σ_1^1.

3.1 ICPDL with fixed points and nominals: μ-ICPDL$_{\mathtt{Nom}}$

As for PDL, let $\mathbb{P}$ be a countable set of atomic propositions and let $\mathbb{A}$ be a countable set of atomic programs. Moreover, let $\overline{\mathbb{A}} = \{\overline{a} \mid a \in \mathbb{A}\}$ be a disjoint copy of $\mathbb{A}$. By $\mathbb{X}$ we denote a countable set of *fixed point variables* and by $\mathtt{Nom}$ a countable set of *nominals*. We assume that $\mathbb{P}$, $\mathbb{X}$, and $\mathtt{Nom}$ are pairwise disjoint. The set of μ-ICPDL$_{\mathtt{Nom}}$ *formulas* Φ and the set of μ-ICPDL$_{\mathtt{Nom}}$ *programs* Π are the smallest sets that satisfy the following conditions,

- $\mathbb{P} \cup \mathbb{X} \cup \mathtt{Nom} \subseteq \Phi$,
- if $\varphi \in \Phi$, then $\neg\varphi \in \Phi$,
- if $\pi \in \Pi$ and $\varphi \in \Phi$, then $\langle\pi\rangle\varphi \in \Phi$,
- if $\varphi \in \Phi$, $X \in \mathbb{X}$, and every occurrence of X in φ appears within an even number of negations, then $\mu X.\varphi \in \Phi$,
- if $\varphi \in \Phi$ and $p \in \mathtt{Nom}$, then $@_p\varphi \in \Phi$,
- $\mathbb{A} \cup \overline{\mathbb{A}} \subseteq \Pi$,
- if $\varphi \in \Phi$, then $\varphi? \in \Pi$,
- if $\pi_1, \pi_2 \in \Pi$, then $\pi_1 \text{ op } \pi_2 \in \Pi$ for each op $\in \{\cup, \cap, \circ\}$, and
- if $\pi \in \Pi$, then $\pi^* \in \Pi$.

The *size* $|\alpha|$ of each μ-ICPDL$_{\texttt{Nom}}$ formula or program α is inductively defined as follows,

- $|p| = |X| = 1$ for each $p \in \mathbb{P} \cup \texttt{Nom}$ and each $X \in \mathbb{X}$,
- $|a| = |\overline{a}| = 1$ for each $a \in \mathbb{A}$,
- $|\neg\varphi| = |\varphi?| = |\mu X.\varphi| = |@_p\varphi| = |\varphi| + 1$ for each $X \in \mathbb{X}$ and each $p \in \texttt{Nom}$,
- $|\langle\pi\rangle\varphi| = |\pi| + |\varphi| + 1$,
- $|\pi_1 \cup \pi_2| = |\pi_1 \cap \pi_2| = |\pi_1 \circ \pi_2| = |\pi_1| + |\pi_2| + 1$, and
- $|\pi^*| = |\pi| + 1$.

In the following, Kripke structures will have a valuation of fixed point variables as an additional fourth component and each nominal represents a singleton world set. Formally, a *Kripke structure* is a tuple $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \texttt{Nom}\}, \rho)$, where

- W is a set of *worlds*,
- $\rightarrow_a \subseteq W \times W$ is a *transition relation* for each $a \in \mathbb{A}$,
- $W_p \subseteq W$ is a subset for each $p \in \mathbb{P} \cup \texttt{Nom}$ and $|W_p| = 1$ for each $p \in \texttt{Nom}$, and
- $\rho : \mathbb{X} \rightarrow 2^W$ is a *valuation* (of fixed point variables).

For each subset $V \subseteq W$, each fixed point variable $X \in \mathbb{X}$, and each valuation ρ, we denote by $\rho[X \mapsto V]$ the following valuation

$$\rho[X \mapsto V](Y) \quad = \quad \begin{cases} V & \text{if } Y = X \\ \rho(Y) & \text{else} \end{cases} \qquad \text{for each } Y \in \mathbb{X}.$$

Extending the latter notation to Kripke structures, we define

$$K[X \mapsto V] \quad = \quad (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \texttt{Nom}\}, \rho[X \rightarrow V]).$$

Before defining the semantics of μ-ICPDL$_{\texttt{Nom}}$, we need the following theorem due to Knaster and Tarski. Originally and more generally, it was proven for monotone functions on complete lattices; we state the theorem only for monotone functions on power set lattices.

Theorem 3.1 (Knaster-Tarksi fixed point theorem [108]) *Assume that W is an arbitrary set and let $F : 2^W \rightarrow 2^W$ be a monotone function, i.e. for each $V_1, V_2 \subseteq W$ we have that $V_1 \subseteq V_2$ implies $F(V_1) \subseteq F(V_2)$. Then F has a least fixed point $\mathbf{lfp}(F)$ and a greatest fixed point $\mathbf{gfp}(F)$ with respect to set inclusion. Moreover these fixed points satisfy the following equalities:*

$$\begin{aligned} \mathbf{lfp}(F) &\quad = \quad \bigcap\{V \subseteq W \mid F(V) \subseteq V\} \\ \mathbf{gfp}(F) &\quad = \quad \bigcup\{V \subseteq W \mid F(V) \supseteq V\} \end{aligned}$$

The least fixed point $\mathbf{lfp}(F)$ *equals* F^λ *for the least ordinal* λ *with* $F^\lambda = F^{\lambda+1}$*, where*

$$\begin{aligned} F^0 &= \emptyset, \\ F^{\lambda+1} &= F(F^\lambda), \ \textit{and} \\ F^\lambda &= \bigcup_{\beta<\lambda} F^\beta \ \textit{for limit ordinals } \lambda. \end{aligned}$$ □

We are now ready to define the semantics of μ-ICPDL$_{\mathtt{Nom}}$.
Let $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \mathtt{Nom}\}, \rho)$ be a Kripke structure. We define for each formula $\varphi \in \Phi$ a subset $[\![\varphi]\!]_K \subseteq W$ and for each program $\pi \in \Pi$ a binary relation $[\![\pi]\!]_K \subseteq W \times W$ as follows,

$$\begin{aligned} [\![p]\!]_K &= W_p \ \text{ for each } p \in \mathbb{P} \cup \mathtt{Nom}, \\ [\![X]\!]_K &= \rho(X) \ \text{ for each } X \in \mathbb{X}, \\ [\![\neg\varphi]\!]_K &= W \setminus [\![\varphi]\!]_K, \\ [\![\langle\pi\rangle\varphi]\!]_K &= \{x \mid \exists y : (x,y) \in [\![\pi]\!]_K \wedge y \in [\![\varphi]\!]_K\}, \\ [\![\mu X.\varphi]\!]_K &= \mathbf{lfp}(U \mapsto [\![\varphi]\!]_{K[X\mapsto U]}), \\ [\![@_p\varphi]\!]_K &= \begin{cases} W & \text{if } W_p \subseteq [\![\varphi]\!]_K, \\ \emptyset & \text{else} \end{cases} \ \text{ for each } p \in \mathtt{Nom}, \\ [\![a]\!]_K &= \rightarrow_a \ \text{ for each } a \in \mathbb{A}, \\ [\![\overline{a}]\!]_K &= \{(y,x) \mid x \rightarrow_a y\} \ \text{ for each } a \in \mathbb{A}, \\ [\![\varphi?]\!]_K &= \{(x,x) \mid x \in [\![\varphi]\!]_K\}, \\ [\![\pi^*]\!]_K &= [\![\pi]\!]_K^*, \ \text{and} \\ [\![\pi_1 \text{ op } \pi_2]\!]_K &= [\![\pi_1]\!]_K \text{ op } [\![\pi_2]\!]_K \ \text{ for each op} \in \{\cup, \cap, \circ\}. \end{aligned}$$

Note that the function $U \mapsto [\![\varphi]\!]_{K[X\mapsto U]}$ can easily proven to be monotone since we required in formulas $\mu X.\varphi$ that the fixed point variable X occurs only positively in φ. Hence, by application of the Knaster-Tarski fixed point theorem we have

$$[\![\mu X.\varphi]\!]_K \ = \ \mathbf{lfp}(U \mapsto [\![\varphi]\!]_{K[X\mapsto U]}) = \bigcap\{U \subseteq W \mid [\![\varphi]\!]_{K[X\mapsto U]} \subseteq U\}.$$

Analogously, we introduce greatest fixed point formulas $\nu X.\varphi = \neg\mu X.\neg\psi$, where φ is obtained from ψ by replacing each occurrence of X by $\neg X$. Analogously, we have

$$[\![\nu X.\varphi]\!]_K \ = \ \mathbf{gfp}(U \mapsto [\![\varphi]\!]_{K[X\to U]}) = \bigcup\{U \subseteq W \mid [\![\varphi]\!]_{K[X\to U]} \supseteq U\}.$$

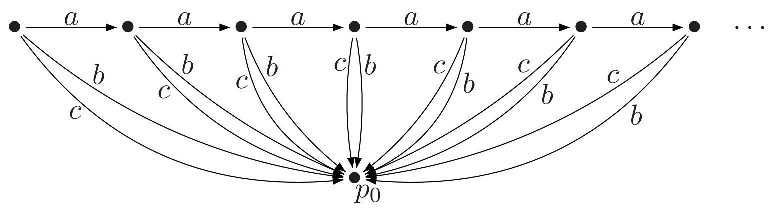
Furthermore, we introduce the usual abbreviations $\varphi_1 \wedge \varphi_2 = \langle\varphi_1?\rangle\varphi_2$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathtt{true} = p \vee \neg p$ for some $p \in \mathbb{P}$, $\mathtt{false} = \neg\mathtt{true}$, and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$. For a μ-ICPDL$_{\mathtt{Nom}}$ formula or program $\alpha \in \Phi \cup \Pi$, let $\mathtt{Nom}(\alpha)$ denote the finite set of nominals that occur in α.

We say a μ-ICPDL$_{\mathtt{Nom}}$ formula or program α is in *normal form*, if for every variable $X \in \mathbb{X}$ occurring in α there exists a unique fixed point subformula of α that binds every occurrence of X in α. Hence in formulas in normal form there do not occur free fixed point variables. It is easy to translate any formula into an equivalent formula in normal form by renaming multiple bound fixed point variables and by replacing unbound fixed point variables by appropriate atomic propositions. Note that whether a μ-ICPDL$_{\texttt{Nom}}$ formula in normal form holds in a world of a Kripke structure K does not depend on the fourth component of K, i.e. for any μ-ICPDL$_{\texttt{Nom}}$ formula φ in normal form, any Kripke structure $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \texttt{Nom}\}, \rho)$, and any Kripke structure K' that agrees with K on the first three components we have $[\![\varphi]\!]_K = [\![\varphi]\!]_{K'}$.

Example 3.2 *Let $p_0 \in \texttt{Nom}$ be a nominal. Any model of the formula*

$$\nu X. (\neg p_0 \wedge \langle a \rangle X \;\wedge\; \mu Y. [\overline{a}] Y \wedge \langle (b \circ p_0? \circ \overline{c}) \cap \texttt{true}? \rangle \texttt{true} \,)$$

contains the following substructure:



□

Infinitary ICPDL with nominals: ICPDL$^{\infty}_{\texttt{Nom}}$

Next, we define ICPDL$^{\infty}_{\texttt{Nom}}$, an infinitary analog of ICPDL that allows the usage of nominals.

Definition 3.3 *The set of* ICPDL$^{\infty}_{\texttt{Nom}}$ formulas Φ_∞ *and the set of* ICPDL$^{\infty}_{\texttt{Nom}}$ programs Π_∞ *are the smallest sets that satisfy the following conditions:*

(1) $\mathbb{X} \subseteq \Phi_\infty$,

(2) $\texttt{Nom} \subseteq \Phi_\infty$,

(3) $\mathbb{P} \subseteq \Phi_\infty$,

(4) if $\varphi \in \Phi_\infty$, then $\neg\varphi \in \Phi_\infty$,

(5) if $S \subseteq \Phi_\infty$, then $\bigvee S \in \Phi_\infty$,

(6) if $\pi \in \Pi_\infty$ and $\varphi \in \Phi_\infty$, then $\langle \pi \rangle \varphi \in \Phi_\infty$,

(7) if $p \in \texttt{Nom}$ and $\varphi \in \Phi_\infty$, then $@_p \varphi \in \Phi_\infty$,

(8) $\mathbb{A} \cup \overline{\mathbb{A}} \subseteq \Pi_\infty$,

(9) if $\varphi \in \Phi_\infty$, then $\varphi? \in \Pi_\infty$,

(10) if $\pi_1, \pi_2 \in \Pi_\infty$, then $\pi_1 \;\mathrm{op}\; \pi_2 \in \Pi_\infty$ for each $\mathrm{op} \in \{\cup, \cap, \circ\}$, and

(11) if $\pi \in \Pi_\infty$, then $\pi^* \in \Pi_\infty$. □

As for μ-ICPDL$_{\texttt{Nom}}$, we denote by $\texttt{Nom}(\alpha)$ the set of nominals that occur in an ICPDL$^\infty_{\texttt{Nom}}$ formula or program α. The set of of $\mathbb{X}$-*free* formulas and programs are the smallest sets that satisfy conditions (2) to (11) of Defintion 3.3.

The semantics of ICPDL$^\infty_{\texttt{Nom}}$ formulas and programs is defined as for μ-ICPDL$_{\texttt{Nom}}$. Formally, if $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \texttt{Nom}\}, \rho)$ is a Kripke structure, we define

$$
\begin{aligned}
[\![X]\!]_K &= \rho(X) \text{ for each } X \in \mathbb{X}, \\
[\![p]\!]_K &= W_p \text{ for each } p \in \mathbb{P} \cup \texttt{Nom}, \\
[\![\neg\varphi]\!]_K &= W \setminus [\![\varphi]\!]_K, \\
[\![\bigvee S]\!]_K &= \bigcup_{\psi \in S} [\![\psi]\!]_K \text{ for each } S \subseteq \Phi_\infty, \\
[\![\langle\pi\rangle\varphi]\!]_K &= \{x \mid \exists y \in [\![\varphi]\!]_K : (x, y) \in [\![\pi]\!]_K\}, \\
[\![@_p\varphi]\!]_K &= \begin{cases} W & \text{if } W_p \subseteq [\![\varphi]\!]_K \\ \emptyset & \text{else} \end{cases} \quad \text{for each } p \in \texttt{Nom}, \\
[\![a]\!]_K &= \to_a \quad \text{for each } a \in \mathbb{A}, \\
[\![\overline{a}]\!]_K &= \{(y, x) \mid x \to_a y\} \text{ for each } a \in \mathbb{A}, \\
[\![\varphi?]\!]_K &= \{(x, x) \mid x \in [\![\varphi]\!]_K\}, \\
[\![\pi^*]\!]_K &= [\![\pi]\!]_K^*, \text{ and} \\
[\![\pi_1 \text{ op } \pi_2]\!]_K &= [\![\pi_1]\!]_K \text{ op } [\![\pi_2]\!]_K \text{ for each op} \in \{\cup, \cap, \circ\}.
\end{aligned}
$$

The following lemma states that, when restricting ourselves to Kripke structures of a certain cardinality, then every μ-ICPDL$_{\texttt{Nom}}$ is equivalent to some ICPDL$^\infty_{\texttt{Nom}}$ formula.

Lemma 3.4 *Let κ be a cardinal. Then for each μ-ICPDL$_{\texttt{Nom}}$ formula φ there exists an ICPDL$^\infty_{\texttt{Nom}}$ formula $T_\kappa(\varphi)$ with $\texttt{Nom}(\varphi) = \texttt{Nom}(T_\kappa(\varphi))$ such that $[\![\varphi]\!]_K = [\![T_\kappa(\varphi)]\!]_K$ for every Kripke structure K of cardinality at most κ. Moreover, if φ is in normal form, then $T_\kappa(\varphi)$ is $\mathbb{X}$-free.* □

Proof Let α be the some ordinal of cardinality at least $\kappa+1$, say the smallest one. We translate each μ-ICPDL$_{\texttt{Nom}}$ formula φ (resp. each μ-ICPDL$_{\texttt{Nom}}$ program π) into an ICPDL$^\infty_{\texttt{Nom}}$ formula

$T(\varphi) = T_\kappa(\varphi)$ (resp. ICPDL$^\infty_{\texttt{Nom}}$ program $T(\pi) = T_\kappa(\pi)$) inductively as follows,

$$\begin{array}{rcl}
T(X) &=& X \text{ for each } X \in \mathbb{X}, \\
T(p) &=& p \text{ for each } p \in \mathbb{P} \cup \texttt{Nom}, \\
T(\neg\psi) &=& \neg T(\psi), \\
T(\langle\pi\rangle\varphi) &=& \langle T(\pi)\rangle T(\varphi), \\
T(\mu X.\psi) &=& T(\psi)^\alpha, \\
T(@_p\varphi) &=& @_p T(\varphi) \text{ for each } p \in \texttt{Nom}, \\
T(a) &=& a \text{ for each } a \in \mathbb{A}, \\
T(\overline{a}) &=& \overline{a} \text{ for each } a \in \mathbb{A}, \\
T(\psi?) &=& T(\psi)?, \\
T(\pi_1 \text{ op } \pi_2) &=& T(\pi_1) \text{ op } T(\pi_2) \text{ for each op} \in \{\cup, \cap, \circ\}, \text{ and} \\
T(\pi^*) &=& T(\pi)^*,
\end{array}$$

where for every μ-ICPDL$_{\texttt{Nom}}$ formula ψ and every ordinal λ we define

$$\begin{array}{rcll}
T(\psi)^\lambda &=& \texttt{false} & \text{if } \lambda = 0 \\
T(\psi)^{\lambda+1} &=& T(\psi)[X \to T(\psi)^\lambda] & \\
T(\psi)^\lambda &=& \bigvee\{T(\psi)^\beta \mid \beta < \lambda\} & \text{if } \lambda \text{ is a limit ordinal}
\end{array}$$

and where $T(\psi)[X \to T(\psi)^\lambda]$ denotes the formula $T(\psi)$ by additionally replacing every occurrence of X by $T(\psi)^\lambda$. Note that applying the above translation to μ-ICPDL$_{\texttt{Nom}}$ formulas (resp. programs) in normal form result in $\mathbb{X}$-free ICPDL$^\infty_{\texttt{Nom}}$ formulas (resp. programs). To show the lemma, it suffices to prove the following claim.

Claim: For every Kripke structure $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup \texttt{Nom}\}, \rho)$ with $|W| \leq \kappa$, every μ-ICPDL$_{\texttt{Nom}}$ formula φ, and every μ-ICPDL$_{\texttt{Nom}}$ program π we have

(1) $[\![\varphi]\!]_K = [\![T(\varphi)]\!]_K$ and

(2) $[\![\pi]\!]_K = [\![T(\pi)]\!]_K$.

We prove the claim by structural induction on φ and π. We start with Point (1) of the claim.

- If $\varphi = X$ for some $X \in \mathbb{X}$, then the claim follows immediately from $T(X) = X$.
- If $\varphi = p$ for some $p \in \mathbb{P} \cup \texttt{Nom}$, then the claim follows immediately from $T(p) = p$.
- If $\varphi = \neg\psi$, then the claim follows from $T(\varphi) = \neg T(\psi)$ and by Point (1) of induction hypothesis.
- If $\varphi = \langle\pi\rangle\psi$, then by Point (2) of induction hypothesis we have $[\![\pi]\!]_K = [\![T(\pi)]\!]_K$ and by Point (1) of induction hypothesis we have $[\![\psi]\!]_K = [\![T(\psi)]\!]_K$. Since $T(\varphi) = \langle T(\pi)\rangle T(\psi)$ the claim follows.

- If $\varphi = \mu X.\psi$, then by Point (1) of induction hypothesis we have $[\![\psi]\!]_K = [\![T(\psi)]\!]_K$. By the choice of α and the Knaster-Tarski fixed point theorem (Theorem 3.1) the fixed point approximation of the monotone function $U \mapsto [\![\psi]\!]_{K[X\mapsto U]}$ terminates after at most α steps. Hence we have
$$[\![\varphi]\!]_K \;=\; F^\alpha,$$
where for each ordinal λ the set F^λ is defined as follows:
$$\begin{array}{rcll} F^\lambda & = & \emptyset & \text{if } \lambda = 0 \\ F^{\lambda+1} & = & [\![\psi]\!]_{K[X\to F^\lambda]} & \\ F^\lambda & = & \bigcup\limits_{\beta<\lambda} F^\beta & \text{if } \lambda \text{ is a limit ordinal} \end{array}$$
Since $T(\varphi) = T(\psi)^\alpha$ it suffices to prove $F^\lambda = [\![T(\psi)^\lambda]\!]_K$ for every ordinal λ by transfinite induction.
If $\lambda = 0$, then $F^\lambda = \emptyset = [\![T(\psi)^\lambda]\!]_K$ since we put $T(\psi)^\lambda = \texttt{false}$.

Next, let us show that $F^{\lambda+1} = [\![T(\psi)^{\lambda+1}]\!]_K$ assuming $F^\lambda = [\![T(\psi)^\lambda]\!]_K$ inductively. Observe that the following equalities hold:
$$\begin{array}{rcl} [\![\psi]\!]_{K[X\to F^\lambda]} & = & [\![T(\psi)]\!]_{K[X\mapsto F^\lambda]} \\ & = & [\![T(\psi)]\!]_{K[X\mapsto [\![T(\psi)^\lambda]\!]_K]} \\ & = & [\![T(\psi)[X \to T(\psi)^\lambda]]\!]_K \end{array} \tag{3.1}$$
Hence, we can conclude
$$\begin{array}{rcl} F^{\lambda+1} & = & [\![\psi]\!]_{K[X\to F^\lambda]} \\ & \overset{(3.1)}{=} & [\![T(\psi)[X \to T(\psi)^\lambda]]\!]_K \\ & = & [\![T(\psi)^{\lambda+1}]\!]_K. \end{array}$$
Finally, let us assume that λ is a limit ordinal. By induction hypothesis we have $F^\beta = [\![T(\psi)^\beta]\!]_K$ for every $\beta < \lambda$. Hence, we have
$$\begin{array}{rcl} F^\lambda & = & \bigcup\limits_{\beta<\lambda} F^\beta \\ & = & \bigcup\limits_{\beta<\lambda} [\![T(\psi)^\beta]\!]_K \\ & = & [\![\bigvee\{T(\psi)^\beta \mid \beta < \lambda\}]\!]_K \\ & = & [\![T(\psi)^\lambda]\!]_K. \end{array}$$
- If $\varphi = @_p\psi$, the claim follows directly by $T(\varphi) = @_pT(\psi)$ and Point (1) of induction hypothesis.

We prove prove Point (2) of the claim by structural induction on π.

- If $\pi = a$ or $\pi = \overline{a}$ for some $a \in \mathbb{A}$, then the claim follows immediately from $T(a) = a$ and $T(\overline{a}) = \overline{a}$.
- If $\pi = \psi?$, then the claim follows directly from $T(\psi?) = T(\psi)?$ and from Point (1) of induction hypothesis.
- If $\pi = \pi_1 \text{ op } \pi_2$ for some $\text{op} \in \{\cup, \cap, \circ\}$ or $\pi = \pi_1^*$, then the claim immediately follows from the definition of T and Point (2) of induction hypothesis. ■

3.2 Models of bounded tree width suffice

The goal of this section is to prove the following: Having fixed a finite set of nominals M, we define an operator $\# = \#(M)$ on pointed Kripke structures such that every pointed Kripke structure K satisfies the same μ-ICPDL$_{\texttt{Nom}}$ formulas that use nominals from M as $K^\#$ and moreover $K^\#$ has tree width at most $|M| + 2$. Moreover K is countable if and only if $K^\#$ is countable.

Before giving the definition of $\#$ we define tree decompositions and tree width; these are important notions from graph theory [26]. Let $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, \rho)$ be a Kripke structure. A *tree decomposition* of K is a tuple $(T, (B_v)_{v \in V})$, where $T = (V, E)$ is an undirected tree, B_v is a subset of W (called *bag*) for each $v \in V$, and the following three conditions are satisfied:

1. $\bigcup_{v \in V} B_v = W$,
2. for every transition $x \rightarrow_a y$ of K, there exists some $v \in V$ with $x, y \in B_v$, and
3. for every $x \in W$, the set $U = \{v \in V \mid x \in B_v\}$ is a connected subset in E, i.e. $(E \cap U \times U)^* = U \times U$.

The *size* $|D|$ of a tree decomposition $D = (T, (B_v)_{v \in V})$ is defined as $|V|$. The *width* of the tree decomposition is the supremum of $\{|B_v| - 1 \mid v \in V\}$. The *tree width* of a Kripke structure K is the minimal k such that K has a tree decomposition of width k. Let us state the main result of this section.

Theorem 3.5 *Let $M \subseteq \texttt{Nom}$ be a finite set of nominals. For every Kripke structure K and every world x_0 of K there exists a Kripke structure $K^\#$ of tree width at most $|M| + 2$ together with some world $x_0^\#$ of $K^\#$ such that for every μ-ICPDL$_{\texttt{Nom}}$ formula φ_0 with $\texttt{Nom}(\varphi_0) \subseteq M$ we have $x_0 \in [\![\varphi_0]\!]_K$ if and only if $x_0^\# \in [\![\varphi_0]\!]_{K^\#}$. Moreover, if K is countable, then $K^\#$ is countable.* □

Before proving Theorem 3.5, let us deduce the following corollary.

Corollary 3.6 *Every satisfiable μ-ICPDL$_{\texttt{Nom}}$ formula that uses at most k different nominals has a countable model of tree width at most $k + 2$.* □

Proof In [37, 48] it is shown that every satisfiable formula of least fixed point logic (LFP) has a countable model. Since μ-ICPDL$_{\texttt{Nom}}$ is a fragment of LFP, every satisfiable μ-ICPDL$_{\texttt{Nom}}$ formula has a countable model. The corollary now follows directly from Theorem 3.5. ■

For the rest of this section we prove Theorem 3.5. Let $k = |M|$ and let $p_1, \ldots, p_k$ be an enumeration of M. Fix a Kripke structure $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P} \cup M\}, \rho)$ and a world $x_0 \in W$. Moreover, let $z_i \in W$ be the unique world of K that corresponds to the nominal p_i for each $i \in [k]$. Firstly, we inductively define an undirected tree $T = (V, E)$ together with a node labeling $t_v \in W \cup W^2 \cup W^3$ for each $v \in V$ given by the following rules:

(1) Start the construction with a root $v_0 \in V$ and put $t_{v_0} = x_0$.

(2) If $t_v = x$ for some $x \in W$ then for every $x' \in W$ add a child v' to v and put $t_{v'} = (x, x')$.

(3) If $t_v = (x_1, x_2) \in W^2$, then for every $x_3 \in W$ add
   - a child v_1 to v with $t_{v_1} = (x_1, x_3, x_2)$ and
   - a child v_2 to v with $t_{v_2} = x_3$.

(4) If $t_v = (x_1, x_2, x_3) \in W^3$, then add children v_1 and v_2 to v with $t_{v_1} = (x_1, x_2)$ and $t_{v_2} = (x_2, x_3)$.

We assume that successors are added at most once to each node and that the rules are applied in a breadth first manner. A *place* is a pair $(v, x) \in V \times W$ such that t_v contains x or $x = z_i$ for some $i \in [k]$. We denote by P_T the *set of places of* T and define $\approx$ to be the smallest equivalence relation over P_T that contains all pairs of the form $((v, x), (v', x))$ such that $(v, v') \in E$. Whenever $(v, x) \in P_T$ we denote by $[v, x]$ the equivalence class of (v, x) with respect to $\approx$. Finally, let us define the Kripke structure

$$K^{\#} \quad = \quad (W', \{\rightarrow'_a \mid a \in \mathbb{A}\}, \{W'_p \mid p \in \mathbb{P} \cup M\}, \rho'),$$

where

- $W' = \{[v, x] \mid (v, x) \in P_T\}$,
- $[v_1, x_1] \rightarrow'_a [v_2, x_2]$ whenever there exists some $v_3 \in V$ such that $(v_1, x_1) \approx (v_3, x_1)$, $(v_3, x_2) \approx (v_2, x_2)$ and moreover $x_1 \rightarrow_a x_2$, for each $a \in \mathbb{A}$,
- $W'_p = \{[v, x] \in W' \mid x \in W_p\}$ for each $p \in \mathbb{P} \cup M$, and
- $\rho'(X) = \{[v, x] \in W' \mid x \in \rho(X)\}$ for each $X \in \mathbb{X}$.

Note that since we restrict ourselves to μ-ICPDL$_{\texttt{Nom}}$ formulas in normal form, the definition of ρ' is irrelevant and hence could have been chosen arbitrarily.

Remark 3.7 *Note that $(v, z_i) \approx (v', z_i)$ for each $v, v' \in V$ and for each $i \in [k]$ due to definition of P_T and $\approx$. Thus $W'_{p_i} = \{[v_0, z_i]\}$ is a singleton for each $i \in [k]$ and hence nominals are well-defined in K' .* □

Lemma 3.8 *For every place $(v, x) \in P_T$ there exists some $v' \in V$ such that $(v, x) \approx (v', x)$ and $t_{v'} = x$.* □

PROOF We prove the lemma by case distinction whether t_v is a singleton, a pair, or a triple. Let us first prove the case when t_v is a pair; we will reduce the remaining two cases to this case.

Assume t_v is a pair, say $t_v = (x_1, x_2)$ for some $x_1, x_2 \in W$. By rule (3) of the definition of T node v has a child v' with $t_{v'} = x$. Altogether we obtain $(v, x) \approx (v', x)$ and $t_{v'} = x$ as required.

Assume t_v is a triple, say $t_v = (x_1, x_2, x_3)$ for some $x_1, x_2, x_3 \in W$. By rule (4) of the definition of T there exist children v_1, v_2 of v such that $t_{v_1} = (x_1, x_2)$ and $t_{v_2} = (x_2, x_3)$. By case distinction whether x corresponds to a nominal world or not one verifies that $(v, x) \approx (v_i, x)$ for some $i \in \{1, 2\}$. We are done since t_{v_i} is a pair.

Finally, assume t_v is a singleton, say $t_v = y$ for some $y \in W$. By rule (2) of the definition of T node v has a child v' such that $t_{v'} = (y, x)$. Since $(v, x) \approx (v', x)$ and $t_{v'}$ is a pair, we are done. ■

We define $x_0^{\#} = [v_0, x_0]$. It is not hard to see that $D = (T, (B_v)_{v \in V})$ with $B_v = \{[v, x] \mid (v, x) \in P_T\}$ for each $v \in V$ is a tree decomposition of $K^{\#}$. Moreover the width of D at most $|M| + 2$ since the size of each bag B_v is bounded by $|M| + 3$. Furthermore, if W is countable, then T (and hence $K^{\#}$) is clearly countable too.

From the following lemma we will deduce below that x_0 and $x_0^{\#}$ satisfy the same μ-ICPDL$_{\texttt{Nom}}$ formulas that use nominals from M.

Lemma 3.9 *Let Φ_K be the set of $\mathbb{X}$-free ICPDL$^{\infty}_{\texttt{Nom}}$ formulas φ and let Π_K be the set of $\mathbb{X}$-free ICPDL$^{\infty}_{\texttt{Nom}}$ programs π over the set of nominals M such that for all $(v, x), (v', x') \in P_T$ the following three conditions are satisfied,*

(i) if $t_v = (x, x')$ and $(x, x') \in [\![\pi]\!]_K$, then $([v, x], [v, x']) \in [\![\pi]\!]_{K^{\#}}$,

(ii) if $([v, x], [v', x']) \in [\![\pi]\!]_{K^{\#}}$, then $(x, x') \in [\![\pi]\!]_K$, and

(iii) $x \in [\![\varphi]\!]_K$ if and only if $[v, x] \in [\![\varphi]\!]_{K^{\#}}$.

Then $M \subseteq \Phi_K$ and moreover Φ_K and Π_K are closed under Points (3) to (11) of Definition 3.3. □

Before proving Lemma 3.9, let us show that Theorem 3.5 follows from it.

First of all the implicant of Lemma 3.9 states that Φ_K (resp. Π_K) equals the *whole* set of $\mathbb{X}$-free ICPDL$^{\infty}_{\texttt{Nom}}$ formulas (resp. programs) over nominals M since the latter are defined to be the *smallest* sets that satisfy conditions (3) to (11) of Definition 3.3. For proving Theorem 3.5, it remains to show that for every μ-ICPDL$_{\texttt{Nom}}$ formula φ_0 with $\texttt{Nom}(\varphi_0) \subseteq M$ we have $x_0 \in [\![\varphi_0]\!]_K$ if and only if $[v_0, x_0] \in [\![\varphi_0]\!]_{K^{\#}}$. For this, let us fix such a μ-ICPDL$_{\texttt{Nom}}$ formula φ_0. Let moreover κ be the cardinality of $K^{\#}$. Observe that the cardinality of K is bounded by κ by construction of $K^{\#}$. By Lemma 3.4 we know that there exists an $\mathbb{X}$-free ICPDL$^{\infty}_{\texttt{Nom}}$ formula $T_\kappa(\varphi_0)$ with $[\![\varphi_0]\!]_K = [\![T_\kappa(\varphi_0)]\!]_K$ and $[\![\varphi_0]\!]_{K^{\#}} = [\![T_\kappa(\varphi_0)]\!]_{K^{\#}}$. Next, Point (iii) of

Lemma 3.9 implies for every $\mathbb{X}$-free ICPDL$^{\infty}_{\mathtt{Nom}}$ formula φ over M that $x_0 \in [\![\varphi]\!]_K$ if and only if $[v_0, x_0] \in [\![\varphi]\!]_{K\#}$, in particular for $\varphi = T_\kappa(\varphi_0)$. Summarizing, the following equalities hold:

$$\begin{aligned} x_0 \in [\![\varphi_0]\!]_K &\overset{\text{Lemma 3.4}}{\Longleftrightarrow} & x_0 \in [\![T_\kappa(\varphi_0)]\!]_K \\ &\overset{\text{Lemma 3.9}}{\Longleftrightarrow} & [v_0, x_0] \in [\![T_\kappa(\varphi_0)]\!]_{K\#} \\ &\overset{\text{Lemma 3.4}}{\Longleftrightarrow} & [v_0, x_0] \in [\![\varphi_0]\!]_{K\#} \end{aligned}$$

Hence Theorem 3.5 follows. We finish this section with the proof of Lemma 3.9.

PROOF (OF LEMMA 3.9) Firstly, let us show that $M \subseteq \Phi_K$. This boils down to showing $x \in [\![p_i]\!]_K$ if and only if $[v, x] \in [\![p_i]\!]_{K\#}$ for each $(v, x) \in P_T$ and each $i \in [k]$. The latter follows directly from $W'_{p_i} = \{[v, z_i]\}$ for each $i \in [k]$, see also Remark 3.7. To complete the proof, we show that Φ_K and Π_K are closed under Points (3) to (11) of Definition 3.3.

(3) Let $p \in \mathbb{P}$ be arbitrary. We prove that Point (iii) holds for p. By definition of W'_p we have $x \in [\![p]\!]_K$ if and only if $[v, x] \in [\![p]\!]_{K\#}$ for all $(v, x) \in P_T$. Thus $p \in \Phi_K$.

(4) Let $\varphi \in \Phi_K$. Then for each $(v, x) \in P_T$ we have

$$\begin{aligned} x \in [\![\neg\varphi]\!]_K &\Longleftrightarrow & x \notin [\![\varphi]\!]_K \\ &\overset{\varphi\in\Phi_K}{\Longleftrightarrow} & [v, x] \notin [\![\varphi]\!]_{K\#} \\ &\Longleftrightarrow & [v, x] \in [\![\neg\varphi]\!]_{K\#}. \end{aligned}$$

This proves Point (iii) and hence $\neg\varphi \in \Phi_K$.

(5) Let $S \subseteq \Phi_K$. Then for every $(v, x) \in P_T$ we have

$$\begin{aligned} x \in [\![\bigvee S]\!]_K &\Longleftrightarrow & x \in \bigcup_{\psi\in S} [\![\psi]\!]_K \\ &\overset{S\subseteq\Phi_K}{\Longleftrightarrow} & [v, x] \in \bigcup_{\psi\in S} [\![\psi]\!]_{K\#} \\ &\Longleftrightarrow & [v, x] \in [\![\bigvee S]\!]_{K\#}. \end{aligned}$$

This proves Point (iii) and hence $\bigvee S \in \Phi_K$.

(6) Let $\pi \in \Pi_K$ and $\varphi \in \Phi_K$. To show that $\langle\pi\rangle\varphi \in \Phi_K$, we must prove that Point (iii) holds for $\langle\pi\rangle\varphi$, i.e. that for each $(v, x) \in P_T$ we have

$$x \in [\![\langle\pi\rangle\varphi]\!]_K \quad \text{if and only if} \quad [v, x] \in [\![\langle\pi\rangle\varphi]\!]_{K\#}.$$

Let us fix some $(v, x) \in P_T$.

"Only if": Assume $x \in [\![\langle\pi\rangle\varphi]\!]_K$. Then there exists some $x' \in W$ such that $(x, x') \in [\![\pi]\!]_K$ and $x' \in [\![\varphi]\!]_K$. We show that $[v, x] \in [\![\langle\pi\rangle\varphi]\!]_{K\#}$ holds. By Lemma 3.8 there exists some $v' \in V$ such that $(v, x) \approx (v', x)$ and $t_{v'} = x$. By construction of T node v' has a successor v'' with $t_{v''} = (x, x')$. Since $\pi \in \Pi_K$ and $(x, x') \in [\![\pi]\!]_K$ we have $([v'', x], [v'', x']) \in$

$[\![\pi]\!]_{K\#}$ by Point (i) and since $\varphi \in \Phi_K$ and $x' \in \Phi_K$ we have $[v'', x'] \in [\![\varphi]\!]_{K\#}$ by Point (iii). Since moreover $(v, x) \approx (v', x) \approx (v'', x)$ we obtain $[v, x] \in [\![\langle\pi\rangle\varphi]\!]_{K\#}$.

"If": Assume that $[v, x] \in [\![\langle\pi\rangle\varphi]\!]_{K\#}$. Then there exists some $[v', x']$ with $([v, x], [v', x']) \in [\![\pi]\!]_{K\#}$ and $[v', x'] \in [\![\varphi]\!]_{K\#}$. Since $\pi \in \Pi_K$ we know by Point (ii) that $(x, x') \in [\![\pi]\!]_K$ and since $\varphi \in \Phi_K$ we know by Point (iii) that $x' \in [\![\varphi]\!]_K$, hence $x \in [\![\langle\pi\rangle\varphi]\!]_K$.

This shows $\langle\pi\rangle\varphi \in \Phi_K$.

(7) Let $\varphi \in \Phi$ and $p_i \in M$. Then for every $(v, x) \in P_T$ we have

$$\begin{aligned} x \in [\![@_{p_i}\varphi]\!]_K \quad &\iff \quad z_i \in [\![\varphi]\!]_K \\ &\overset{\varphi\in\Phi_K}{\iff} \quad [v, z_i] \in [\![\varphi]\!]_{K\#} \\ &\iff \quad [v, x] \in [\![@_{p_i}\varphi]\!]_{K\#}. \end{aligned}$$

This proves Point (iii) and hence $@_{p_i}\varphi \in \Phi_K$.

(8) Let $a \in \mathbb{A}$ and let $(v, x), (v', x') \in P_T$.
To show Point (i), assume $t_v = (x, x')$ and $(x, x') \in [\![a]\!]_K$. By definition of $\rightarrow'_a$ it immediately follows $([v, x], [v, x']) \in [\![a]\!]_{K\#}$.
To show Point (ii), assume $([v, x], [v', x']) \in [\![a]\!]_{K\#}$. By definition of $\rightarrow'_a$ there exists some $v'' \in V$ such that $(v, x) \approx (v'', x)$, $(v'', x') \approx (v', x')$, and $x \rightarrow_a x'$, where the latter was to be shown.
Hence $a \in \Pi_K$. One can prove $\overline{a} \in \Phi_K$ for each $a \in \mathbb{A}$ analogously.

(9) Let $\varphi \in \Phi_K$ and let $(v, x), (v', x') \in P_T$.
To show Point (i) assume $t_v = (x, x')$ and $(x, x') \in [\![\varphi?]\!]_K$. Hence $[v, x] = [v, x']$. Since $\varphi \in \Phi_K$ we have $[v, x] \in [\![\varphi]\!]_{K\#}$ by Point (iii). Thus $([v, x], [v, x']) \in [\![\varphi?]\!]_{K\#}$ which shows Point (i).
For Point (ii) assume $([v, x], [v', x']) \in [\![\varphi?]\!]_{K\#}$. This implies $[v, x] = [v, x']$ and $[v, x] \in [\![\varphi]\!]_{K\#}$. Since $\varphi \in \Phi_K$, it follows $x \in [\![\varphi]\!]_K$ by Point (iii) and thus $(x, x') \in [\![\varphi?]\!]_K$.
Hence, $\varphi? \in \Phi_K$.

(10) Let $\pi_1, \pi_2 \in \Pi_K$. We have to show that $\pi = \pi_1 \text{ op } \pi_2 \in \Pi_K$ for each op $\in \{\cup, \cap, \circ\}$. Let $(v_1, x_1), (v_2, x_2) \in P_T$ be arbitrary. We make a case distinction of op.

- $\pi = \pi_1 \cup \pi_2$: To show Point (i) assume $t_{v_1} = (x_1, x_2)$ and $(x_1, x_2) \in [\![\pi]\!]_K$. Thus $(x_1, x_2) \in [\![\pi_i]\!]_K$ for some $i \in \{1, 2\}$. Since $\pi_i \in \Pi_K$ we know that $([v_1, x_1], [v_1, x_2]) \in [\![\pi_i]\!]_{K\#}$ by Point (i). Hence $([v_1, x_1], [v_1, x_2]) \in [\![\pi]\!]_{K\#}$ which shows Point (i).
  To show Point (ii) assume $([v_1, x_1], [v_2, x_2]) \in [\![\pi]\!]_{K\#}$. Thus $([v_1, x_1], [v_2, x_2]) \in [\![\pi_i]\!]_{K\#}$ for some $i \in \{1, 2\}$. Since $\pi_i \in \Pi_K$ we know that $(x_1, x_2) \in [\![\pi_i]\!]_K$ by Point (ii) and thus $(x_1, x_2) \in [\![\pi]\!]_K$ which proves Point (ii).
  Hence $\pi_1 \cup \pi_2 \in \Pi_K$.
- $\pi = \pi_1 \cap \pi_2$: Can be proven analogously to the case $\pi = \pi_1 \cup \pi_2$.
- $\pi = \pi_1 \circ \pi_2$: To show Point (i) assume $t_{v_1} = (x_1, x_2)$ and $(x_1, x_2) \in [\![\pi]\!]_K$. Hence there exists some world $x_3 \in W$ such that $(x_1, x_3) \in [\![\pi_1]\!]_K$ and $(x_3, x_2) \in [\![\pi_2]\!]_K$.

By construction of T node v_1 has a child v_3 labeled by $t_{v_3} = (x_1, x_3, x_2)$. Moreover v_3 has children v_4 and v_5 with $t_{v_4} = (x_1, x_3)$ and $t_{v_5} = (x_3, x_2)$. Since $\pi_1, \pi_2 \in \Pi_K$ we have $([v_4, x_1], [v_4, x_3]) \in [\![\pi_1]\!]_{K\#}$ and $([v_5, x_3], [v_5, x_2]) \in [\![\pi_2]\!]_{K\#}$ by Point (i). Since moreover $(v_1, x_1) \approx (v_3, x_1) \approx (v_4, x_1)$, $(v_4, x_3) \approx (v_5, x_3)$ and $(v_5, x_4) \approx (v_3, x_4) \approx (v_1, x_4)$ we obtain $([v_1, x_1], [v_1, x_2]) \in [\![\pi]\!]_{K\#}$. This proves Point (i).
For Point (ii) assume $([v_1, x_1], [v_2, x_2]) \in [\![\pi]\!]_{K\#}$. Thus there exists some $(v_3, x_3) \in P_T$ such that $([v_1, x_1], [v_3, x_3]) \in [\![\pi_1]\!]_{K\#}$ and $([v_3, x_3], [v_2, x_2]) \in [\![\pi_2]\!]_K$. Since $\pi_1, \pi_2 \in \Pi_K$ we know that $(x_1, x_3) \in [\![\pi_1]\!]_K$ and $(x_3, x_2) \in [\![\pi_2]\!]_K$ by Point (ii). Hence $(x_1, x_2) \in [\![\pi]\!]_K$ which proves Point (ii).
Hence $\pi_1 \circ \pi_2 \in \Pi_K$.

(11) Let $\pi_1 \in \Pi_K$ and let $(v_1, x_1), (v_2, x_2) \in P_T$. We prove that $\pi = \pi_1^*$ satisfies Points (i) and (ii) of the lemma.
For Point (i) assume $t_{v_1} = (x_1, x_2)$. We prove that

$$(x_1, x_2) \in [\![\pi_1]\!]_K^i \text{ implies } ([v_1, x_1], [v_1, x_2]) \in [\![\pi_1]\!]_{K\#}^i$$

by induction on $i \geq 0$. The case $i = 0$ is clear since then $x_1 = x_2$ and thus $[v_1, x_1] = [v_1, x_2]$. For the induction step let $i \geq 0$ and assume $(x_1, x_2) \in [\![\pi_1]\!]_K^{i+1}$. Then there exists some $x_3 \in W$ such that $(x_1, x_3) \in [\![\pi_1]\!]_K$ and $(x_3, x_2) \in [\![\pi_1]\!]_K^i$. Since $t_{v_1} = (x_1, x_2)$ node v_1 has a child v_3 such that $t_{v_3} = (x_1, x_3, x_2)$ by construction of T. Moreover v_3 has children v_4 and v_5 where $t_{v_4} = (x_1, x_3)$ and $t_{v_5} = (x_3, x_2)$. Since $\pi_1 \in \Pi_K$ we know that $([v_4, x_1], [v_4, x_3]) \in [\![\pi_1]\!]_{K\#}$ by Point (ii). By induction hypothesis we know that $([v_5, x_3], [v_5, x_2]) \in [\![\pi_1]\!]_{K\#}^i$. Since $(v_1, x_1) \approx (v_3, x_1) \approx (v_4, x_1)$, $(v_4, x_3) \approx (v_5, x_3)$, and $(v_5, x_2) \approx (v_3, x_2) \approx (v_1, x_2)$ we have $([v_1, x_1], [v_5, x_3]) \in [\![\pi_1]\!]_{K\#}$ and $([v_5, x_3], [v_1, x_2]) \in [\![\pi_1]\!]_{K\#}^i$. Thus $([v_1, x_1], [v_1, x_2]) \in [\![\pi_1]\!]_{K\#}^{i+1}$, which proves Point (i).

For Point (ii) we prove by induction on $i \geq 0$ that $([v_1, x_1], [v_2, x_2]) \in [\![\pi_1]\!]_{K\#}^i$ implies $(x_1, x_2) \in [\![\pi_1]\!]_K^i$. The case $i = 0$ is clear since $[v_1, x_1] = [v_2, x_2]$ implies $x_1 = x_2$. For the induction step let $i \geq 0$ and assume $([v_1, x_1], [v_2, x_2]) \in [\![\pi_1]\!]_{K\#}^{i+1}$. Hence there exists some $(v_3, x_3) \in P_T$ such that $([v_1, x_1], [v_3, x_3]) \in [\![\pi_1]\!]_{K\#}$ and $([v_3, x_3], [v_2, x_2]) \in [\![\pi_1]\!]_{K\#}^i$. Since $\pi_1 \in \Pi_K$ we know that $(x_1, x_3) \in [\![\pi_1]\!]_K$ by Point (ii). By induction hypothesis we have $(x_3, x_2) \in [\![\pi_1]\!]_K^i$. Hence we conclude $(x_1, x_2) \in [\![\pi_1]\!]_K^{i+1}$ which proves Point (ii). ∎

3.3 Translation to ω-regular tree satisfiability

In this section, we will reduce the satisfiability problem in μ-ICPDL$_{\texttt{Nom}}$ to a special reasoning problem that we call ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$. Informally, ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$ is to decide, given a μ-ICPDL$_{\texttt{Nom}}$ formula φ and a two-way alternating parity tree automaton over infinite trees (TWAPTA) $\mathcal{T}$ – the definition will be given below – whether there is a tree accepted by $\mathcal{T}$ that, when viewed as a Kripke structure, is a model of φ.

Let Σ_N be a finite node alphabet and Σ_E a finite edge alphabet. A *Σ_N-labeled Σ_E-tree* is a partial function $T : \Sigma_E^* \to \Sigma_N$ such that the set of *nodes* $\mathrm{dom}(T)$ is prefix-closed. If

$\mathrm{dom}(T) = \Sigma_E^*$, then T is called *complete*. In the following we mostly work with complete trees. For a node $va \in \mathrm{dom}(T)$ with $a \in \Sigma_E$, we say that the node va is the *(a-)successor* of v and v is the *(a-)predecessor* of va. We use $\mathrm{tree}(\Sigma_N, \Sigma_E)$ to denote the set of all complete Σ_N-labeled Σ_E-trees. If Σ_E is not important, we simply talk of Σ_N-labeled trees.

The trees accepted by TWAPTAs in this chapter are complete 2^{P}-labeled A-trees, where $\mathsf{A} \subseteq \mathbb{A}$ and $\mathsf{P} \subseteq \mathbb{P} \cup \mathtt{Nom} \cup \mathbb{X}$ are finite sets. Any such tree $T : \mathsf{A}^* \to 2^{\mathsf{P}}$ such that additionally $\{u \in \mathsf{A}^* \mid p \in T(u)\}$ is a singleton for each $p \in \mathsf{P} \cap \mathtt{Nom}$ can be identified with the Kripke structure

$$(\mathsf{A}^*, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{T_p \mid p \in \mathsf{P} \cap (\mathbb{P} \cup \mathtt{Nom})\}, \rho),$$

where

- $\rightarrow_a = \{(u, ua) \mid u \in \mathsf{A}^*\}$ for each $a \in \mathsf{A}$,
- $T_p = \{u \in \mathsf{A}^* \mid p \in T(u)\}$ for each $p \in \mathsf{P} \cap (\mathbb{P} \cup \mathtt{Nom})$, and
- $\rho(X) = \{u \in \mathsf{A}^* \mid X \in T(u)\}$ for each $X \in \mathsf{P} \cap \mathbb{X}$.

Observe that Kripke structures derived in this way are total with respect to A and deterministic.

For an edge alphabet Σ_E, let $\overline{\Sigma_E} = \{\overline{a} \mid a \in \Sigma_E\}$ be a disjoint copy of Σ_E. For $u \in \Sigma_E^*$ and $d \in \Sigma_E \uplus \overline{\Sigma_E} \uplus \{\varepsilon\}$ define

$$u \cdot d = \begin{cases} ud & \text{if } d \in \Sigma_E \\ u & \text{if } d = \varepsilon \\ v & \text{if there exists } a \in \Sigma_E \text{ with } d = \overline{a} \text{ and } u = va \\ \text{undefined} & \text{else, i.e. if } d = \overline{a} \text{ for } a \in \Sigma_E \text{ but } u \text{ does not end with } a. \end{cases}$$

Before defining TWAPTAs, recall that over each finite set A we denote by $\mathcal{B}^+(A)$ the set of positive boolean formulas over A.

Definition 3.10 *A* two-way alternating parity tree automaton (TWAPTA) *over complete Σ_N-labeled Σ_E-trees is a tuple $\mathcal{T} = (S, \delta, s_0, \mathrm{Acc})$, where*

- *S is a finite non-empty set of* states*,*
- *$\delta : S \times \Sigma_N \to \mathcal{B}^+(\mathrm{ext}(\Sigma_E))$ is the* transition function*, where $\mathrm{ext}(\Sigma_E) = S \times (\Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\})$ is the set of* moves*,*
- *$s_0 \in S$ is the* initial state*, and*
- *$\mathrm{Acc} : S \to \mathbb{N}$ is the* priority function*.* □

For $s \in S$ and $d \in \Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\}$ we write the corresponding move as $\langle s, d\rangle$. Intuitively, a move $\langle s, a\rangle$, with $a \in \Sigma_E$, means that the TWAPTA sends a copy of itself in state s to the a-successor of the current tree node. Similarly, $\langle s, \overline{a}\rangle$ means to change to state s and to move to the a-predecessor (if the current node has one at all), where $\langle s, \varepsilon\rangle$ means to change to state s but to stay in the current node. Formally, the behaviour of TWAPTAs is defined in terms of runs. Let $\mathcal{T}$ be a TWAPTA as above, $T \in \mathrm{tree}(\Sigma_N, \Sigma_E)$ a tree, $u \in \Sigma_E^*$ a node, and $s \in S$ a state of $\mathcal{T}$. An *(s, u)-run* of $\mathcal{T}$ on T is a (not necessarily complete) $(S \times \Sigma_E^*)$-labeled tree T_R such that

- $T_R(\varepsilon) = (s, u)$, and
- for all $w \in \text{dom}(T_R)$, if $T_R(w) = (p, v)$ and $\delta(p, T(v)) = \theta$, then there is a subset $Y \subseteq \text{ext}(\Sigma_E)$ that satisfies θ and such that for all $(p', d) \in Y$, $v \cdot d$ is defined and there exists a successor w' of w in T_R with $T_R(w') = (p', v \cdot d)$.

We say that an (s, u)-run T_R is *successful* if for every infinite path $w_1, w_2 \cdots$ in T_R (i.e. $w_1 = \varepsilon$ and w_{i+1} is a successor of w_i for each $i \geq 0$), the number

$$\min\{\text{Acc}(s) \mid s \in S \text{ and } T_R(w_i) \in \{s\} \times \Sigma_E^* \text{ for infinitely many } i\}$$

is even. For $s \in S$ define

$$\begin{aligned} [\![\mathcal{T}, s]\!] &= \{(T, u) \mid T \in \text{tree}(\Sigma_N, \Sigma_E), u \in \Sigma_E^*, \text{ and} \\ &\qquad \text{there exists a successful } (s, u)\text{-run of } \mathcal{T} \text{ on } T\} \quad \text{and} \\ [\![\mathcal{T}]\!] &= [\![\mathcal{T}, s_0]\!]. \end{aligned}$$

The *language* $L(\mathcal{T})$ accepted by $\mathcal{T}$ is defined as the set of all trees $T \in \text{tree}(\Sigma_N, \Sigma_E)$ such that $(T, \varepsilon) \in [\![\mathcal{T}]\!]$. We remark that our model of TWAPTAs differs slightly from other definitions that can be found in the literature [121, 66]. Firstly, we run TWAPTAs only on complete trees, which will facilitate some technical constructions when reducing general satisfiability to ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$. Secondly, standard TWAPTAs have moves of the form $\langle s, -1\rangle$ for moving to the parent node. In our model, we use moves of the form $\langle s, \overline{a}\rangle$, which can only be executed if the current node is an a-successor of its parent node. It is not hard to see that these two models are equivalent. In particular, it is easy to see that the result of Vardi stated in Theorem 3.11 below also applies to our version of TWAPTAs. For a TWAPTA $\mathcal{T} = (S, \delta, s_0, \text{Acc})$, we define its *size* $|\mathcal{T}| = |S|$ as its number of states and we define its *index* $i(\mathcal{T})$ as $\max\{\text{Acc}(s) \mid s \in S\}$. The size $|\delta|$ of the transition function δ is the sum of the sizes of all positive boolean functions that appear in the range of δ. For a TWAPTA $\mathcal{T}$, let $S(\mathcal{T})$ denote its set of states, $\delta(\mathcal{T})$ its transition function, $s_0(\mathcal{T})$ its initial state, and $\text{Acc}(\mathcal{T})$ its priority function.

Theorem 3.11 ([121]) *For a given TWAPTA $\mathcal{T}$ with transition function δ, it can be checked in time* $\exp(|\mathcal{T}| + i(\mathcal{T}))\cdot \text{poly}(|\delta|)$ *whether* $L(\mathcal{T}) \neq \emptyset$. □

We can now formally define ω-regular tree satisfiability. Let φ be a μ-ICPDL$_{\texttt{Nom}}$ formula, let $\mathsf{A} = \{a \in \mathbb{A} \mid a \text{ occurs in } \varphi\}$ and $\mathsf{P} = \{p \in \mathbb{P} \cup \texttt{Nom} \mid p \text{ occurs in } \varphi\} \cup \{X \in \mathbb{X} \mid X \text{ occurs in } \varphi\}$. The formula φ is *satisfiable with respect to a TWAPTA* $\mathcal{T} = (S, \delta, s_0, \text{Acc})$ over 2^{P}-labeled A-trees, if there is a $T \in L(\mathcal{T})$ such that $\varepsilon \in [\![\varphi]\!]_T$. Thus, *$\omega$-regular tree satisfiability* is the problem to decide, given such φ and $\mathcal{T}$, whether φ is satisfiable with respect to $\mathcal{T}$.

The translation

For representing tree decompositions of countable Kripke structures suitably as an ω-regular tree language later, we call tree decomposition good, if its underlying tree is the complete binary tree and bags of adjacent nodes are comparable with respect to set inclusion. Formally, a tree decomposition $(T = (V, E), (B_v)_{v \in V})$ is *good* if

- $V = \{a,b\}^*$,
- $E = \{\{v, vc\} \mid v \in \{a,b\}^*, c \in \{a,b\}\}$, and
- $B_v \subseteq B_u$ or $B_u \subseteq B_v$ for each $\{u,v\} \in E$.

Recall that the size of a tree decomposition is the size of the node set of its underlying tree. Clearly, every good tree decomposition is countable. The following lemma states that requiring a tree decomposition of a countable Kripke structure to be good is not a restriction.

Lemma 3.12 *Every countable Kripke structure of tree width k has a good tree decomposition of width k.* $\square$

PROOF Let K be a countable Kripke structure of tree width k. Then there exists a tree decomposition $(T = (V,E), (B_v)_{v\in V})$ of width k of K, where V is countable. We show how to convert $(T, (B_v)_{v\in V})$ into a good tree decomposition in two steps. Firstly, every edge $\{u,v\}$ of T, such that neither $B_u \subseteq B_v$ nor $B_v \subseteq B_u$, is replaced by two edges $\{u,w\}$ and $\{w,v\}$, where w is a new node. The bag B_w is $B_u \cap B_v$. Secondly, we convert T into a binary tree. Let $u \in V$ have $\ell > 2$ many children $v_1, v_2, \ldots$. Since K is countable, ℓ is finite or $\aleph_0$. Then we introduce a chain of edges $\{u_1,u_2\},\{u_2,u_3\},\ldots$ of length ℓ, where $u_1 = u$ and $u_2, u_3, \ldots$ are new nodes. Each edge $\{u, v_i\}$ is replaced by $\{u_i, v_i\}$ and the nodes u_i all receive the same bag as u. It remains to account for every node to have precisely two children: To nodes $u \in V$ with $l < 2$ children we add $2 - l$ children that have the same bag as u. ■

Combining Corollary 3.6 and Lemma 3.12 yields that every satisfiable μ-ICPDL$_{\texttt{Nom}}$ φ has a countable model that has a good tree decomposition of width at most $|\texttt{Nom}(\varphi)| + 2$. To reduce satisfiability in μ-ICPDL$_{\texttt{Nom}}$ to ω-regular tree satisfiability, we show how to translate a μ-ICPDL$_{\texttt{Nom}}$ formula φ into a μ-ICPDL$_{\texttt{Nom}}$ formula φ' and a TWAPTA $\mathcal{T}$ such that φ is satisfiable if and only if φ' is satisfiable with respect to $\mathcal{T}$. So let us assume that φ is in normal form and let $M = \{\widehat{p_1}, \ldots, \widehat{p_k}\}$ be a disjoint copy of $\texttt{Nom}(\varphi) = \{p_1, \ldots, p_k\}$. The formula φ' will not contain any nominals, i.e. $\texttt{Nom}(\varphi') = \emptyset$. Let prop$(\varphi) = \{p \in \mathbb{P} \mid p$ occurs in $\varphi\}$ denote the atomic propositions that occur in φ, let prog$(\varphi) = \{a \in \mathbb{A} \mid a$ occurs in $\varphi\}$ denote that atomic programs that occur in φ, and finally let var$(\varphi) = \{X \in \mathbb{X} \mid X$ occurs in $\varphi\}$ denote the fixed point variables that occur in φ. The formula φ' will use atomic propositions from

$$\mathsf{P} \quad = \quad \{t\} \uplus \text{prop}(\varphi) \uplus \text{var}(\varphi) \uplus M \uplus ([k+3] \times \text{prog}(\varphi) \times [k+3]) \uplus \binom{[k]}{2},$$

and atomic programs from

$$\mathsf{A} \quad = \quad \{a,b\} \cup [k+3].$$

Our TWAPTA $\mathcal{T}$ works on complete 2^{P}-labeled A-trees. Intuitively, for every countable Kripke structure K of tree width $w \leq k+2$ there exists some tree T accepted by $\mathcal{T}$ that encodes K together with some good tree decomposition of K of width w. Moreover K is a model of φ if and only if the root of T satisfies φ'.

Conversely, each tree T accepted by $\mathcal{T}$ encodes a Kripke structure K together with a good tree decomposition of K of width at most $k+2$, and the root of T satisfies φ' if and only if K is a model of φ.

Firstly, we describe the mentioned encoding on an intuitive level. Let K be a Kripke structure and $D = (U, (B_v)_{v \in V})$ a good tree decomposition of K of width at most $k+2$, with $U = (V, E)$. Let us recall that $V = \{a, b\}^*$, that ε the root of the tree U, and that for each $v \in V$ the nodes va and vb are the children of v. The tree T that is supposed to encode K and D has roughly the structure of U. But besides its children va and vb each node $v \in T$ has additional children $v1, v2, \ldots, v(k+3)$. Intuitively, when talking about T, we think of each node $v \in \{a, b\}^*$ as providing $k+3$ slots which can be empty or filled with a world of the Kripke structure K, i.e. the $k+3$ slots correspond to an ordered representation of B_v. The additional successors $v1, v2, \ldots, v(k+3)$ of v in T describe these $k+3$ slots. Whenever slot vi ($i \in [k+3]$) is occupied by a world of K, then vi receives the special label $t \in \mathsf{P}$ in T. Additionally, each node vi, that corresponds to some world x of K, is labeled with the same atomic propositions (resp. fixed point variables) that hold in x (resp. that K's fixed point valuation maps to sets that contain x), if any. Recall that in the proof of Theorem 3.5 we can assume that in each bag of U all worlds that are represented by nominals $p_1, \ldots, p_k$ occur. We establish to locate these nominals in the first k slots of each bag. But note that we must not require that different nominals represent different worlds in K, i.e. vi might represent the same world as vj with $i \neq j$: To account for this, we allow the labeling of v to contain the pair $\{i, j\}$ but demand that this labeling is consistent for all nodes from $\{a, b\}^*$. Information about the transitions of K are stored in T using nodes from $\{a, b\}^*$. For example, if there is a γ-transition in K from the world represented by vi to the world represented by vj, then the labeling of v contains the tuple (i, γ, j).

Let us now formally define this. A complete 2^{P}-labeled A-tree T is called *valid* if for every $v, v' \in \mathsf{A}^*$ we have

(V1) if $v \notin \{a, b\}^* \cup \{a, b\}^*[k+3]$, then $T(v) = \emptyset$,

(V2) if $v \in \{a, b\}^*$, then

- $\{t, \widehat{p_i}\} \subseteq T(vi) \subseteq \{t, \widehat{p_i}\} \cup \text{prop}(\varphi) \cup \text{var}(\varphi)$ for all $i \in [k]$, and
- $T(vi) = \emptyset$ or $\{t\} \subseteq T(vi) \subseteq \{t\} \cup \text{prop}(\varphi) \cup \text{var}(\varphi)$ for all $k+1 \leq i \leq k+3$,

set $B_v = \{i \mid t \in T(vi)\}$ describing the slots of v that represent worlds of K,

(V3) if $v \in \{a, b\}^*$, then $T(v) \subseteq (B_v \times \text{prog}(\varphi) \times B_v) \cup \binom{[k]}{2}$,

(V4) if $v \in \{a, b\}^*$ and $c \in \{a, b\}$, then $B_v \subseteq B_{vc}$ or $B_{vc} \subseteq B_v$, and

(V5) if $v, v' \in \{a, b\}^*$, then for each $i, j \in [k]$ with $i \neq j$ we have

$$\{i, j\} \in T(v) \quad \Leftrightarrow \quad \{i, j\} \in T(v').$$

Let us fix some valid 2^{P}-labeled A-tree T. We now make precise the Kripke structure $K(T)$ that is described by T. Define a set of *places*

$$P_T \quad = \quad \{u \in \{a, b\}^*[k+3] \mid t \in T(u)\}$$

and let $\sim_T$ be the smallest equivalence relation on P_T which contains all pairs $(vi, vci) \in P_T \times P_T$ and all pairs $(vi, vj) \in P_T \times P_T$ with $\{i, j\} \in T(v)$, where $v \in \{a, b\}^*$, $c \in \{a, b\}$, and $i, j \in [k+3]$. For $u \in P_T$, we use $[u]$ to denote the equivalence class of u with respect to $\sim_T$. Now set

$$K(T) \quad = \quad (W, \{\rightarrow_\gamma \mid \gamma \in \text{prog}(\varphi)\}, \{W_p \mid p \in \text{prop}(\varphi) \cup \mathtt{Nom}(\varphi)\}, \rho),$$

where

$$\begin{array}{rcl}
W & = & \{[u] \mid u \in P_T\}, \\
\rightarrow_\gamma & = & \{([vi],[vj]) \mid v \in \{a,b\}^*, (i,\gamma,j) \in T(v)\} \text{ for each } \gamma \in \text{prog}(\varphi), \\
W_p & = & \{[u] \in W \mid p \in T(u)\} \text{ for each } p \in \text{prop}(\varphi), \\
W_{p_i} & = & \{[u] \in W \mid \widehat{p_i} \in T(u)\} \text{ for each } p_i \in \mathtt{Nom}(\varphi), \text{ and} \\
\rho(X) & = & \{[u] \in W \mid X \in T(u)\} \text{ for each } X \in \text{var}(\varphi).
\end{array}$$

Note again that the choice of ρ is irrelevant. Also recall that the structure $K(T)$ should not be confused with T *viewed as a Kripke structure* over P and A as discussed at the beginning of this section: The original formula φ whose satisfiability is to be decided is interpreted in $K(T)$ whereas the reduction formula φ', to be defined below, is interpreted in T viewed as a Kripke structure over P and A.
The next remark states that nominals represent singleton sets in $K(T)$.

Remark 3.13 *For every $p_i \in \mathtt{Nom}(\varphi)$ we have*

$$[\![p_i]\!]_{K(T)} \quad = \quad \{[vi] \mid v \in \{a,b\}^*\} \quad = \quad \{[i]\}.$$

The following lemma establishes the correctness of our encoding.

Lemma 3.14 *The formula φ is satisfiable if and only if there exists a valid tree $T \in$ tree$(2^{\mathsf{P}}, \mathsf{A})$ such that $K(T)$ is a model of φ.* $\square$

PROOF (SKETCH) The "if" direction is trivial. For the "only-if"-direction, assume φ is satisfiable. Then by Corollary 3.6 and Lemma 3.12 there exists a countable model K of φ and a good tree decomposition $(U, (B_v)_{v \in \{a,b\}^*})$ of K of width at most $k+2$. Moreover by the proof of Theorem 3.5 we can assume that each bag B_v contains all worlds of K that correspond to some nominal from $\{p_1, \ldots, p_k\}$. By a consistent ordering of the members in each bag B_v it is easy but tedious to give a valid 2^{P}-labeled A-tree T such that $K = K(T)$. $\blacksquare$

It still remains to convert the μ-ICPDL$_{\mathtt{Nom}}$ formula φ, whose satisfiability is to be decided, into a μ-ICPDL$_{\mathtt{Nom}}$ formula φ' over propositions P and programs A and a TWAPTA $\mathcal{T}$.
Before defining the formula φ', we introduce the auxiliary program

$$\pi^1_\sim = \bigcup_{i \in [k+3]} t? \circ \overline{i} \circ (a \cup b \cup \overline{a} \cup \overline{b}) \circ i \circ t? \cup \bigcup_{i,j \in [k]} t? \circ \overline{i} \circ \{i,j\}? \circ j \circ t?$$

and set $\pi_\sim = t? \circ (\pi^1_\sim)^*$. It is easy to see that for each valid tree $T \in$ tree$(2^{\mathsf{P}}, \mathsf{A})$ the relation $[\![\pi_\sim]\!]_T$ equals $\sim_T$. For a μ-ICPDL$_{\mathtt{Nom}}$ formula or program α, let $\tau(\alpha)$ be obtained from α by replacing

- every occurrence of an atomic program $\gamma \in \mathrm{prog}(\varphi)$ by

$$\bigcup_{i,j\in[k+3]} \pi_\sim \circ \bar{i} \circ (i,\gamma,j)? \circ j \circ \pi_\sim,$$

- every occurrence of an atomic proposition $p \in \mathrm{prop}(\varphi)$ by $\langle \pi_\sim \rangle p$,
- every occurrence of a nominal $p_i \in \mathtt{Nom}(\varphi)$ by $\langle \pi_\sim \rangle \widehat{p_i}$,
- every occurrence of a fixed point variable $X \in \mathbb{X}$ that is not directly behind a "μ" in α by $\langle \pi_\sim \rangle X$, and
- every occurrence of $@_{p_i}$ by

$$\left\langle \bigcup_{j\in[k+3]} t? \circ \bar{j} \circ i \circ t? \right\rangle .$$

Observe that the definition of τ directly reflects the definition of $K(T)$. Formally, we have the following lemma.

Lemma 3.15 *For all subformulas ψ of φ, subprograms π of φ, valid trees $T \in tree(2^{\mathsf{P}}, \mathsf{A})$, and places $u, v \in P_T$, we have:*

(1) $u \in [\![\tau(\psi)]\!]_T$ if and only if $[u] \in [\![\psi]\!]_{K(T)}$ and

(2) $(u, v) \in [\![\tau(\pi)]\!]_T$ if and only if $([u],[v]) \in [\![\pi]\!]_{K(T)}$. □

PROOF The lemma follows from $[\![\pi_\sim]\!]_T = \sim_T$ and from an easy induction on the structure of ψ and π. ■

We define

$$\varphi' \quad = \quad \langle (a \cup b)^* \rangle \bigvee_{i\in[k+3]} \langle i \rangle \left(t \wedge \tau(\varphi)\right).$$

Remark 3.16 *Observe that $|\varphi'| \in \mathrm{poly}(|\varphi|)$.* □

Lemma 3.17 *The formula φ is satisfiable if and only if there exists some valid $T \in tree(2^{\mathsf{P}}, \mathsf{A})$ such that $\varepsilon \in [\![\varphi']\!]_T$.* □

PROOF Assume that φ is satisfiable. By Lemma 3.14 there exists a valid tree $T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$ such that $K(T)$ is a model of φ, say $[u] \in [\![\varphi]\!]_{K(T)}$ for some $u \in P_T$. By Point (1) of Lemma 3.15 we have $u \in [\![\tau(\varphi)]\!]_T$. By definition of φ' it follows $\varepsilon \in [\![\varphi']\!]_T$.
Conversely assume that $\varepsilon \in [\![\varphi']\!]_T$ for some valid tree $T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$. Hence, there exists some place $u \in P_T$ with $u \in [\![\tau(\varphi)]\!]_K$ by definition of φ'. By Point (1) of Lemma 3.15 it follows $[u] \in [\![\varphi]\!]_{K(T)}$. Hence φ is satisfiable. ■

Let us state the main result of this section.

Theorem 3.18 *From a given μ-ICPDL$_{\texttt{Nom}}$ formula φ one can compute a TWAPTA $\mathcal{T}$ and a μ-ICPDL$_{\texttt{Nom}}$ formula φ' in time $\exp(|\varphi|)$ such that the following holds,*

(1) φ *is satisfiable if and only if there exists some* $T \in L(\mathcal{T})$ *such that* $\varepsilon \in [\![\varphi']\!]_T$*,*

(2) $|\varphi'| \in \text{poly}(|\varphi|)$*,*

(3) $|\mathcal{T}| \in \text{poly}(|\varphi|)$*,*

(4) $|\delta(\mathcal{T})| \in \exp(|\varphi|)$*, and*

(5) $i(\mathcal{T}) \in \mathcal{O}(1)$. □

PROOF It is straightforward to compute in time exponential in $|\varphi|$ a TWAPTA $\mathcal{T}$ satisfying conditions (3), (4), and (5) of the theorem such that $L(\mathcal{T})$ equals the set of valid trees 2^{P}-labeled A-trees by realizing Points (V1) to (V5). We choose φ' as described above. Point (1) of the theorem follows directly from Lemma 3.17. Point (2) follows immediately from Remark 3.16. ■

3.4 An upper bound for ω-regular tree satisfiability

In this section we show that ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$ is in 2EXP. The matching lower bound will easily follow from Theorem 3.35, where it is shown that the data complexity already of test-free IPDL over BPAs is 2EXP-hard. Too, ω-regular tree satisfiability is 2EXP-hard by a result from [72], where it is shown that satisfiability in test-free IPDL over trees is already 2EXP-hard. Therefore, we concentrate on the upper bound for ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$.

We prove containment in 2EXP via a reduction to the non-emptiness problem for TWAPTAs. We inductively translate each μ-ICPDL$_{\texttt{Nom}}$ formula φ to a TWAPTA that recognizes precisely those trees that satisfy φ in their root. However, to handle μ-ICPDL$_{\texttt{Nom}}$ programs, we need to introduce a notion of NFAs that navigate through 2^{P}-labeled A-trees. So for the rest of this section, fix a finite set $\mathsf{P} \subseteq \mathbb{P} \cup \texttt{Nom} \cup \mathbb{X}$ and a finite set $\mathsf{A} \subseteq \mathbb{A}$. In this section, let tree$(2^{\mathsf{P}}, \mathsf{A})$ denote the set of 2^{P}-labeled A-trees $T : \mathsf{A}^* \to 2^{\mathsf{P}}$ such that $\{u \in \mathsf{A}^* \mid p \in T(u)\}$ is a singleton for each $p \in \mathsf{P} \cap \texttt{Nom}$. Recall that we identify each $T \in$ tree$(2^{\mathsf{P}}, \mathsf{A})$ with a corresponding Kripke structure.

Some notation

To handle the behaviour of μ-ICPDL$_{\texttt{Nom}}$ programs we introduce NFAs that navigate up and down trees from tree$(2^{\mathsf{P}}, \mathsf{A})$. Formally, a *nondeterministic finite automaton (NFA) $\mathcal{A}$ over a TWAPTA $\mathcal{T} = (S, \delta, s_0, \text{Acc})$* is a tuple

$$\mathcal{A} \quad = \quad (Q, p_0, q_0, \to_{\mathcal{A}}),$$

where Q is a finite set of *states*, p_0 and q_0 are two *selected states*, and $\to_{\mathcal{A}}$ is a set of *labeled transitions* of the following form, where $q, q' \in Q$ and $a \in \mathsf{A}$,

$$q \xrightarrow{a}_{\mathcal{A}} q', \quad q \xrightarrow{\overline{a}}_{\mathcal{A}} q', \quad \text{or} \quad q \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q' \text{ with } s \in S.$$

Transitions of the third kind are called *test transitions*.

Definition 3.19 *For $T \in tree(2^{\mathsf{P}}, \mathsf{A})$ let $\Rightarrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ be the smallest relation such that for all $u \in \mathsf{A}^*$, $a \in \mathsf{A}$, $p, q \in Q$, and $s \in S$, we have*

$$(u, p) \Rightarrow_{\mathcal{A},T} (ua, q) \quad \text{if} \quad p \xrightarrow{a}_{\mathcal{A}} q, \tag{3.2}$$

$$(ua, p) \Rightarrow_{\mathcal{A},T} (u, q) \quad \text{if} \quad p \xrightarrow{\overline{a}}_{\mathcal{A}} q, \text{ and} \tag{3.3}$$

$$(u, p) \Rightarrow_{\mathcal{A},T} (u, q) \quad \text{if} \quad p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q \text{ and } (T, u) \in [\![\mathcal{T}, s]\!]. \tag{3.4}$$

□

Define

$$[\![\mathcal{A}]\!] \quad = \quad \{(T, u, v) \mid T \in \text{tree}(2^{\mathsf{P}}, \mathsf{A}), u, v \in \mathsf{A}^*, \text{and } (u, p_0) \Rightarrow^*_{\mathcal{A},T} (v, q_0)\}.$$

When considering an NFA over a TWAPTA $\mathcal{T}$, the initial state of $\mathcal{T}$ is obviously useless. Thus, in the context of NFAs, a TWAPTA will henceforth only be a 3-tuple. The *disjoint union* $\mathcal{T}_1 \uplus \mathcal{T}_2$ of TWAPTAs $\mathcal{T}_i = (S_i, \delta_i, \mathrm{Acc}_i)$, $i \in \{1, 2\}$, both defined over NFAs is defined as $\mathcal{T}_1 \uplus \mathcal{T}_2 = (S_1 \uplus S_2, \delta', \mathrm{Acc}')$, where $\delta'(s, \gamma) = \delta_i(s, \gamma)$ for every $\gamma \subseteq \mathsf{P}$ whenever $s \in S_i$ with $i \in \{1, 2\}$. Define $\mathrm{Acc}'(s) = \mathrm{Acc}_i(s)$ for every $s \in S_i$ with $i \in \{1, 2\}$.

Next, we define for each μ-ICPDL$_{\texttt{Nom}}$ formula or program α the *set of subformulas* $\text{subf}(\alpha)$ and *the set of subprograms* $\text{subp}(\alpha)$ inductively as follows,

- $\text{subp}(a) = \{a\}, \text{subp}(\overline{a}) = \{a, \overline{a}\}$, and $\text{subf}(a) = \text{subf}(\overline{a}) = \emptyset$ for every $a \in \mathbb{A}$,
- $\text{subp}(\pi) = \{\pi\} \cup \text{subp}(\pi_1) \cup \text{subp}(\pi_2)$ and $\text{subf}(\pi) = \text{subf}(\pi_1) \cup \text{subf}(\pi_2)$ for every $\pi = \pi_1 \text{ op } \pi_2$ where $\text{op} \in \{\cup, \cap, \circ\}$,
- $\text{subp}(\pi^*) = \{\pi^*\} \cup \text{subp}(\pi)$ and $\text{subf}(\pi^*) = \text{subf}(\pi)$,
- $\text{subp}(\varphi?) = \{\varphi?\} \cup \text{subp}(\varphi)$ and $\text{subf}(\varphi?) = \text{subf}(\varphi)$,
- $\text{subp}(\alpha) = \emptyset$ and $\text{subf}(\alpha) = \alpha$ for every $\alpha \in \mathbb{P} \cup \texttt{Nom} \cup \mathbb{X}$,
- $\text{subp}(\neg\varphi) = \text{subp}(\varphi)$ and $\text{subf}(\neg\varphi) = \{\neg\varphi\} \cup \text{subf}(\varphi)$,
- $\text{subp}(\langle\pi\rangle\varphi) = \text{subp}(\pi) \cup \text{subp}(\varphi)$ and $\text{subf}(\langle\pi\rangle\varphi) = \{\langle\pi\rangle\varphi\} \cup \text{subf}(\pi) \cup \text{subf}(\varphi)$,
- $\text{subp}(\mu X.\varphi) = \text{subp}(\varphi)$ and $\text{subf}(\mu X.\varphi) = \{\mu X.\varphi\} \cup \text{subf}(\varphi)$, and
- $\text{subp}(@_p\varphi) = \text{subp}(\varphi)$ and $\text{subf}(@_p\varphi) = \{@_p\varphi\} \cup \text{subf}(\varphi)$.

Observe that $\varphi \in \mathrm{subf}(\varphi)$ for each μ-ICPDL$_{\mathtt{Nom}}$ formula φ.

For the rest of this section, let us fix a TWAPTA $\mathcal{T}_0$ and a μ-ICPDL$_{\mathtt{Nom}}$ formula φ_0 in normal form, where $\mathsf{A} = \{a \in \mathbb{A} \mid a \text{ occurs in } \varphi_0\}$ and $\mathsf{P} = \{p \in \mathbb{P} \cup \mathtt{Nom} \mid p \text{ occurs in } \varphi_0\} \cup \{X \in \mathbb{X} \mid X \text{ occurs in } \varphi_0\}$. So both $\mathcal{T}_0$ and φ_0 are interpreted over $\mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$.

Define a strict partial order $\prec$ on $\mathrm{subf}(\varphi_0) \cup \mathrm{subp}(\varphi_0)$ as follows, where $\alpha, \beta \in \mathrm{subf}(\varphi_0) \cup \mathrm{subp}(\varphi_0)$,

$$\alpha \prec \beta \quad \Leftrightarrow \quad \alpha \neq \beta \ \wedge \ \alpha \in \mathrm{subf}(\beta) \cup \mathrm{subp}(\beta).$$

Observe that, since we assumed that φ_0 is in normal form, for each $X \in \mathrm{subf}(\varphi_0) \cap \mathbb{X}$ there exists a unique formula $\beta_X = \mu X.\psi \in \mathrm{subf}(\varphi_0)$ such that $X \prec \mu X.\psi$.

Let us, for each $\psi \in \mathrm{subf}(\varphi_0)$ and each $\pi \in \mathrm{subp}(\varphi_0)$, define

$$\begin{aligned} [\![\psi]\!] &= \{(T, u) \mid T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A}), u \in \mathsf{A}^*, \text{ and } u \in [\![\psi]\!]_T\} \quad \text{and} \\ [\![\pi]\!] &= \{(T, u, v) \mid T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A}), u, v \in \mathsf{A}^*, \text{ and } (u, v) \in [\![\pi]\!]_T\}. \end{aligned}$$

3.4.1 From μ-ICPDL$_{\mathtt{Nom}}$ to automata

In this section we convert

(1) each $\alpha \in \mathrm{subf}(\varphi_0) \cup \mathrm{subp}(\varphi_0)$ into a TWAPTA $\mathcal{T}(\alpha)$ such that

- $\{\langle\theta\rangle \mid \theta \in \mathrm{subf}(\alpha)\} \subseteq S(\mathcal{T}(\alpha))$,
- $[\![\mathcal{T}(\alpha), \langle\theta\rangle]\!] = \begin{cases} [\![\theta]\!] & \text{if } \theta \notin \mathbb{X} \text{ or } (\theta = X \in \mathbb{X} \text{ and } \beta_X \notin \mathrm{subf}(\alpha)) \\ [\![\beta_X]\!] & \text{if } \theta = X \in \mathbb{X} \text{ and } \beta_X \in \mathrm{subf}(\alpha) \end{cases}$
  for each $\theta \in \mathrm{subf}(\alpha)$, and
- $s_0(\mathcal{T}(\alpha)) = \langle\alpha\rangle$ if $\alpha \in \mathrm{subf}(\varphi_0)$,

(2) and each program $\pi \in \mathrm{subp}(\varphi_0)$ into an NFA $\mathcal{A}(\pi)$ over $\mathcal{T}(\pi)$ such that $[\![\mathcal{A}(\pi)]\!] = [\![\pi]\!]$.

Assume we have achieved (1) and (2). Then observe that $[\![\varphi_0]\!] = [\![\mathcal{T}(\varphi_0)]\!]$.

The NFAs and TWAPTAs we construct all work over trees from $\mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$. The construction is by induction on $\prec$. We start with defining the TWAPTA $T(\psi)$ for each formula $\psi \in \mathrm{subf}(\varphi_0)$. Note that P denotes the atomic propositions, the nominals, and the fixed point variables that occur in φ_0.

If $\psi \in \mathsf{P}$, we put $\mathcal{T}(\psi) = (\{\langle\psi\rangle\}, \delta, \langle\psi\rangle, \langle\psi\rangle \mapsto 1)$, where for all $\gamma \subseteq \mathsf{P}$ we have $\delta(\langle\psi\rangle, \gamma) = \mathtt{true}$ if $\psi \in \gamma$ and $\delta(\langle\psi\rangle, \gamma) = \mathtt{false}$ otherwise. Clearly, $[\![\mathcal{T}(\psi), \langle\psi\rangle]\!] = [\![\psi]\!]$.

If $\psi = \neg\theta$, then we have inductively already constructed $\mathcal{T}(\theta) = (S, \delta, \langle\theta\rangle, \mathrm{Acc})$. The TWAPTA $\mathcal{T}(\psi)$ is basically obtained from $\mathcal{T}(\theta)$ and by applying the standard complementation procedure where all positive boolean formulas on the right-hand side of the transition function are dualized and the acceptance condition is complemented by increasing the priority of every state by one, see e.g. [87]. Formally, define $\mathcal{T}(\psi) = (S \uplus \{\langle\psi\rangle\}, \delta', \langle\psi\rangle, \mathrm{Acc}')$ such that $\mathrm{Acc}'(s) = \mathrm{Acc}(s) +$

1 for each $s \in S$, $\mathrm{Acc}'(\langle\psi\rangle) = 1$, $\delta'(\langle\psi\rangle, \gamma) = (\langle\theta\rangle, \varepsilon)$ for each $\gamma \subseteq \mathsf{P}$ and $\delta'(s, \gamma) = \mathrm{dual}(\delta(s, \gamma))$ for each $s \in S$ and $\gamma \subseteq \mathsf{P}$, where

$$\begin{aligned}
\mathrm{dual}(\texttt{true}) &= \texttt{false}, \\
\mathrm{dual}(\texttt{false}) &= \texttt{true}, \\
\mathrm{dual}(e) &= e \quad \text{for each } e \in \mathrm{ext}(\mathsf{A}) = S \times (\mathsf{A} \uplus \overline{\mathsf{A}} \uplus \{\varepsilon\}), \\
\mathrm{dual}(\varphi_1 \wedge \varphi_2) &= \mathrm{dual}(\varphi_1) \vee \mathrm{dual}(\varphi_2), \text{ and} \\
\mathrm{dual}(\varphi_1 \vee \varphi_2) &= \mathrm{dual}(\varphi_1) \wedge \mathrm{dual}(\varphi_2).
\end{aligned}$$

If $\psi = \langle\pi\rangle\theta$, then we have inductively constructed an NFA $\mathcal{A}(\pi) = (Q, p_0, q_0, \rightarrow_{\mathcal{A}})$ over a TWAPTA $\mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$ such that $[\![\pi]\!] = [\![\mathcal{A}(\pi)]\!]$. We have also constructed a TWAPTA $\mathcal{T}(\theta) = (S_2, \delta_2, \langle\theta\rangle, \mathrm{Acc}_2)$ such that $[\![\theta]\!] = [\![\mathcal{T}(\theta)]\!]$. We may assume that Q, S_1, and S_2 are pairwise disjoint. The TWAPTA $\mathcal{T}(\psi)$ first simulates $\mathcal{A}(\pi)$ and then $\mathcal{T}(\theta)$. Set $\mathcal{T}(\psi) = (S, \delta, \langle\psi\rangle, \mathrm{Acc})$ with $S = Q \cup S_1 \cup S_2 \cup \{\langle\psi\rangle\}$. For states in S_1 and S_2, the transitions of $\mathcal{T}(\psi)$ are as in $\mathcal{T}(\pi)$ and $\mathcal{T}(\theta)$, respectively. To simulate $\mathcal{A}(\pi)$ we add an ε-transition from $\langle\psi\rangle$ to p_0. Next, note that handling transitions $q \xrightarrow{a}_{\mathcal{A}(\pi)} r$ and $q \xrightarrow{\overline{a}}_{\mathcal{A}(\pi)} r$ is easy: $\mathcal{T}(\psi)$ simply navigates up and down the tree as required. To handle a transition $q \xrightarrow{\mathcal{T}(\pi), s}_{\mathcal{A}(\pi)} r$, we branch universally to simulate $\mathcal{T}(\pi)$ in state s *and* to simulate $\mathcal{A}(\pi)$ in state r. To ensure that the simulation of $\mathcal{A}(\pi)$ terminates, the priority function of $\mathcal{T}(\psi)$ assigns 1 to all states of Q. To start the simulation of $\mathcal{T}(\theta)$ after the simulation of $\mathcal{A}(\pi)$ has terminated, we add an ε-transition from q_0 to $\langle\theta\rangle$. Formally, $\delta(\langle\psi\rangle, \gamma) = \langle p_0, \varepsilon\rangle$ for each $\gamma \subseteq \mathsf{P}$. For each for $q \in Q$ and each $\gamma \subseteq \mathsf{P}$, we define

$$\begin{aligned}
\delta(q, \gamma) \;=\;& \bigvee\{\langle r, a\rangle \mid q \xrightarrow{a}_{\mathcal{A}(\pi)} r\} \vee \bigvee\{\langle r, \overline{a}\rangle \mid q \xrightarrow{\overline{a}}_{\mathcal{A}(\pi)} r\} \;\vee \\
& \bigvee\{\langle s, \varepsilon\rangle \wedge \langle r, \varepsilon\rangle \mid q \xrightarrow{\mathcal{T}(\pi), s}_{\mathcal{A}(\pi)} r\}
\end{aligned}$$

with an additional disjunct $\langle\langle\theta\rangle, \varepsilon\rangle$ if $q = q_0$. The priority function Acc is defined by setting $\delta(\langle\psi\rangle) = 1$, $\mathrm{Acc}(s) = 1$ whenever $s \in Q$, and $\mathrm{Acc}(s) = \mathrm{Acc}_i(s)$ whenever $s \in S_i$ with $i \in \{1, 2\}$. It is straightforward to check that $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$.

If $\psi = \beta_X = \mu X.\theta$, then we have inductively already constructed $\mathcal{T}(\theta) = (S, \delta, \langle\theta\rangle, \mathrm{Acc})$. Recall that $\{\langle\alpha\rangle \mid \alpha \in \mathrm{subf}(\theta)\} \subseteq S$, inter alia $\langle X\rangle \in S$. Define $\mathcal{T}(\psi) = (S \uplus \{\langle\psi\rangle\}, \delta', \langle\psi\rangle, \mathrm{Acc})$, where

$$\delta'(s, \gamma) \;=\; \begin{cases} \langle\langle\psi\rangle, \varepsilon\rangle & \text{if } s = \langle X\rangle \\ \langle\langle\theta\rangle, \varepsilon\rangle & \text{if } s = \langle\psi\rangle \\ \delta(s, \gamma) & \text{else} \end{cases}$$

for each $s \in S \uplus \{\langle\psi\rangle\}$ and each $\gamma \subseteq \mathsf{P}$. The priority function is defined as $\mathrm{Acc}'(s) = \mathrm{Acc}(s) + 2$ for each $s \in S$ and $\mathrm{Acc}'(\langle\psi\rangle) = 1$. The proof that $[\![\mathcal{T}(\psi), \langle\psi\rangle]\!] = [\![\mathcal{T}(\psi), \langle X\rangle]\!] = [\![\psi]\!]$ can be shown in analogy to the translation of μ-calculus formulas into alternating parity tree automata from [127, 88, 130] via parity games and is omitted therefore.

If $\psi = @_p\theta$, then we have inductively already constructed $\mathcal{T}(\theta) = (S, \delta, \langle\theta\rangle, \mathrm{Acc})$. The TWAPTA $\mathcal{T}(\psi)$ basically seeks for the unique world that is represented by the nominal p and checks if it satisfies the formula θ. Formally, we put $\mathcal{T}(\psi) = (S \uplus \{\psi\}, \delta', \langle\psi\rangle, \mathrm{Acc}')$, where $\delta'(s,\gamma) = \delta(s,\gamma)$ for all $s \in S$ and all $\gamma \subseteq \mathsf{P}$, and moreover

$$\delta'(\langle\psi\rangle,\gamma) \quad = \quad \bigvee \{\langle\langle\psi\rangle, a\rangle \vee \langle\langle\psi\rangle, \overline{a}\rangle \mid a \in \mathsf{A}\}$$

for each $\gamma \subseteq \mathsf{P}$ with an additional disjunct $\langle\langle\theta\rangle, \varepsilon\rangle$ whenever $p \in \gamma$. Define $\mathrm{Acc}'(s) = \mathrm{Acc}(s)$ for all $s \in S$, and $\mathrm{Acc}'(\langle\psi\rangle) = 1$. We put $\mathrm{Acc}'(\langle\psi\rangle) = 1$ since we require termination of the search for the world which is represented by the nominal p.

Next, we describe the inductive construction of $\mathcal{A}(\pi)$ and $\mathcal{T}(\pi)$ for μ-ICPDL$_{\mathtt{Nom}}$ programs $\pi \in \mathrm{subp}(\varphi_0)$.

If $\pi = a$ or $\pi = \overline{a}$ with $a \in \mathsf{A}$, then the NFA $\mathcal{A} = \mathcal{A}(\pi)$ has its only transition between its two selected states p_0 and q_0, namely $p_0 \xrightarrow{a}_{\mathcal{A}} q_0$ or $p_0 \xrightarrow{\overline{a}}_{\mathcal{A}} q_0$, respectively. Clearly, $[\![\mathcal{A}(\pi)]\!] = [\![\pi]\!]$. Since $\mathcal{A}(\pi)$ has no test transitions, the TWAPTA $\mathcal{T}(\pi)$ is not important. For estimating the size of the constructed automata, we assume that $\mathcal{T}(\pi)$ has a single state and its index is 1.

If $\pi = \psi?$, then by induction we have constructed a TWAPTA $\mathcal{T}(\psi) = (S, \delta, \langle\psi\rangle, \mathrm{Acc})$ such that $[\![\psi]\!] = [\![\mathcal{T}(\psi)]\!]$. The TWAPTA $\mathcal{T}(\pi)$ is $\mathcal{T}(\psi)$. The NFA $\mathcal{A}(\pi)$ has only the two selected states p_0 and q_0 and the transition $p_0 \xrightarrow{\mathcal{T}(\pi),\langle\psi\rangle}_{\mathcal{A}(\pi)} q_0$. Hence, we have $[\![\pi]\!] = \{(T,u,u) \mid (T,u) \in [\![\mathcal{T}(\psi)]\!]\} = [\![\mathcal{A}(\pi)]\!]$.

If $\pi = \pi_1 \cup \pi_2, \pi = \pi_1 \circ \pi_2$, or $\pi = \chi^*$, we construct $\mathcal{A}(\pi)$ 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 define $\mathcal{T}(\pi)$ as the disjoint union of $\mathcal{T}(\pi_1)$ and $\mathcal{T}(\pi_2)$, whereas for $\pi = \chi^*$, we set $\mathcal{T}(\pi) = \mathcal{T}(\chi)$.

It remains to construct $\mathcal{A}(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$, which is the most difficult step of the construction. Assume that the NFA

$$\mathcal{A}(\pi_i) = (Q_i, p_i, q_i, \rightarrow_{\mathcal{A}(\pi_i)}) \text{ over the TWAPTA } \mathcal{T}(\pi_i) \tag{3.5}$$

has already been constructed for $i \in \{1,2\}$. Thus $[\![\mathcal{A}(\pi_i)]\!] = [\![\pi_i]\!]$. A natural idea for defining an NFA for $\pi_1 \cap \pi_2$ is to simply apply a product construction to $\mathcal{A}(\pi_1)$ and $\mathcal{A}(\pi_2)$. This does not work since the product construction forces $\mathcal{A}(\pi_1)$ and $\mathcal{A}(\pi_2)$ to travel along the same path, whereas $\mathcal{A}(\pi_1)$ and $\mathcal{A}(\pi_2)$ may travel from a world $u \in \mathsf{A}^*$ to a world $v \in \mathsf{A}^*$ along two *different* paths when working independently. Fortunately, these two paths are then closely related: the automata both travel along the unique shortest path P from u to v, but can make different "detours" from this path P, i.e., they may divert from P and eventually return to the world on P where the diversion started. In order to eliminate this problem, we modify $\mathcal{A}(\pi_1)$ and $\mathcal{A}(\pi_2)$ by admitting additional test transitions that allow to shortcut these detours. The modified NFA can always travel along the shortest path without any detours, and thus the product construction is applicable.

The mentioned modification proceeds in two steps. Consider an NFA $\mathcal{A} = (Q, q_0, p_0, \rightarrow_{\mathcal{A}})$ over a TWAPTA $\mathcal{T} = (S, \delta, \mathrm{Acc})$. Our aim is to modify $\mathcal{A}$ and $\mathcal{T}$ such that $\mathcal{A}$ can shortcut the detours described above. First, we extend $\mathcal{T}$ into a new TWAPTA $\widehat{\mathcal{T}}$ that recognizes detours made by

$\mathcal{A}$. And second, we extend $\mathcal{A}$ to an NFA $\widehat{\mathcal{A}}$ over $\widehat{\mathcal{T}}$ by adding new test transitions that allow to shortcut detours, exploiting the additional capabilities of $\widehat{\mathcal{T}}$. We can assume that $i(\mathcal{T}) \geq 1$ since we apply the modification process only to TWAPTAs from our construction.

Step 1: Constructing a TWAPTA $\widehat{\mathcal{T}}$ that recognizes detours of $\mathcal{A}$

Let $\mathcal{A}$ and $\mathcal{T}$ be as above, and let $T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$. We define a relation $\mathrm{loop}_{\mathcal{A},T} \subseteq \mathsf{A}^* \times Q \times Q$ that describes the detours that $\mathcal{A}$ makes on input T. As we will prove, we will have $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ whenever in the tree T the NFA $\mathcal{A}$ can do a detour starting from u in state p and ending at u in state q. Formally, $\mathrm{loop}_{\mathcal{A},T}$ is the smallest set such that,

(i) $(u,q,q) \in \mathrm{loop}_{\mathcal{A},T}$ for each $u \in \mathsf{A}^*$ and each $q \in Q$,

(ii) if $(T,u) \in [\![\mathcal{T},s]\!]$ and $p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q$ for $s \in S$, then $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$,

(iii) if $(ua,p',q') \in \mathrm{loop}_{\mathcal{A},T}$, $p \xrightarrow{a}_{\mathcal{A}} p'$ and $q' \xrightarrow{\overline{a}}_{\mathcal{A}} q$, then $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$,

(iv) if $(u,p',q') \in \mathrm{loop}_{\mathcal{A},T}$, $p \xrightarrow{\overline{a}}_{\mathcal{A}} p'$, and $q' \xrightarrow{a}_{\mathcal{A}} q$, then $(ua,p,q) \in \mathrm{loop}_{\mathcal{A},T}$, and

(v) if $(u,p,r) \in \mathrm{loop}_{\mathcal{A},T}$ and $(u,r,q) \in \mathrm{loop}_{\mathcal{A},T}$, then $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$.

The following lemma states that $\mathrm{loop}_{\mathcal{A},T}$ is as required. For this, recall that the relation $\Rightarrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ was introduced in Definition 3.19 .

Lemma 3.20 $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ *if and only if* $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,q)$. □

PROOF "only if": Assume $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$. Let $n \geq 1$ be the smallest number such that $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ follows from n applications of the rules (i)-(v) from above. We prove by induction on n that $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,q)$ holds.
Assume $n = 1$. If $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ follows from application of rule (i), then $p = q$ and thus clearly $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,q)$. If $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ follows from application of rule (ii), then for some $s \in S$ we have $(T,u) \in [\![\mathcal{T},s]\!]$, and $p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q$, and thus $(u,p) \Rightarrow_{\mathcal{A},T} (u,q)$ by definition of $\Rightarrow_{\mathcal{A},T}$.
For the inductive step, assume $n > 1$. If $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ is obtained by finally applying rule (iii), we know that for some $a \in \mathsf{A}$ we have $p \xrightarrow{a}_{\mathcal{A}} p'$, $q' \xrightarrow{\overline{a}}_{\mathcal{A}} q$, and $(ua,p',q') \in \mathrm{loop}_{\mathcal{A},T}$. Since $(ua,p',q') \in \mathrm{loop}_{\mathcal{A},T}$ must follow from $n-1$ applications of the above rules, it follows by induction that $(ua,p') \Rightarrow^*_{\mathcal{A},T} (ua,q')$. Altogether, we obtain $(u,p) \Rightarrow_{\mathcal{A},T} (ua,p') \Rightarrow^*_{\mathcal{A},T} (ua,q') \Rightarrow_{\mathcal{A},T} (u,q)$ and we are done. In case $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ is obtained by finally applying rule (iv), then there exists some $v \in \mathsf{A}^*$ and some $a \in \mathsf{A}$ such that $u = va$, $p \xrightarrow{\overline{a}}_{\mathcal{A}} p'$, $q' \xrightarrow{a}_{\mathcal{A}} q$, and $(v,p',q') \in \mathrm{loop}_{\mathcal{A},T}$. Again, by induction, we have $(v,p') \Rightarrow^*_{\mathcal{A},T} (v,q')$. Altogether we obtain $(u,p) \Rightarrow_{\mathcal{A},T} (v,p') \Rightarrow^*_{\mathcal{A},T} (v,q') \Rightarrow_{\mathcal{A},T} (u,q)$ and we are done. Finally assume that $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ is obtained by finally applying rule (v). Then there exists some state $r \in Q$ such that $(u,p,r) \in \mathrm{loop}_{\mathcal{A},T}$ and $(u,r,q) \in \mathrm{loop}_{\mathcal{A},T}$. Since both $(u,p,r) \in \mathrm{loop}_{\mathcal{A},T}$ and $(u,r,q) \in \mathrm{loop}_{\mathcal{A},T}$ must follow from less than n applications of the above rules, we inductively conclude $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,r) \Rightarrow^*_{\mathcal{A},T} (u,q)$ and we are done.

"if": Assume that $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,q)$. Hence, there are $n \geq 1$, $u_1,\ldots,u_n \in \mathsf{A}^*$, and $q_1,\ldots,q_n \in Q$ such that

1. $u_1 = u_n = u$,
2. $q_1 = p, q_n = q$, and
3. $(u_i, q_i) \Rightarrow_{\mathcal{A},T} (u_{i+1}, q_{i+1})$ for $1 \leq i < n$.

We show $(u,p,q) \in \text{loop}_{\mathcal{A},T}$ by induction on n. If $n = 1$, then $p = q$ and by Point (i) we have $(u,p,q) \in \text{loop}_{\mathcal{A},T}$. If $n = 2$, then there must be a test transition $p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q$ such that $(T,u) \in [\![\mathcal{T},s]\!]$. By Point (ii), we have $(u,p,q) \in \text{loop}_{\mathcal{A},T}$. Now assume that $n \geq 3$. We distinguish the following cases:

Case 1. There exists $1 < i < n$ such that $u_i = u$. By induction, $(u,p,q_i),(u,q_i,q) \in \text{loop}_{\mathcal{A},T}$. Point (v) yields $(u,p,q) \in \text{loop}_{\mathcal{A},T}$.

Case 2. There does not exist $1 < i < n$ such that $u_i = u$. Then one of the the following two subcases applies:

Case 2A. For some $a \in \mathsf{A}$, we have $p \xrightarrow{a}_{\mathcal{A}} q_2$, $q_{n-1} \xrightarrow{\overline{a}}_{\mathcal{A}} q$, and $u_2 = ua = u_{n-1}$. By induction hypothesis, $(ua, q_2, q_{n-1}) \in \text{loop}_{\mathcal{A},T}$. Thus, Point (iii) yields $(u,p,q) \in \text{loop}_{\mathcal{A},T}$.

Case 2B. For some $a \in \mathsf{A}$ and $v \in \mathsf{A}^*$, we have $u = va$, $p \xrightarrow{\overline{a}}_{\mathcal{A}} q_2$, $q_{n-1} \xrightarrow{a}_{\mathcal{A}} q$, and $u_2 = v = u_{n-1}$. By induction hypothesis, $(v, q_2, q_{n-1}) \in \text{loop}_{\mathcal{A},T}$. Thus, Point (iv) yields $(u,p,q) \in \text{loop}_{\mathcal{A},T}$. ■

The following lemma states that Points (i)–(v) can easily be translated into a TWAPTA.

Lemma 3.21 *There is a TWAPTA $\widehat{\mathcal{T}} = (S', \delta', \text{Acc}')$ with $S' = S \uplus (Q \times Q)$ such that for all $s \in S$ and $p,q \in Q$ the following holds,*

(i) $[\![\widehat{\mathcal{T}}, s]\!] = [\![\mathcal{T}, s]\!]$,

(ii) $[\![\widehat{\mathcal{T}}, (p,q)]\!] = \{(T,u) \mid T \in \textit{tree}(2^{\mathsf{P}}, \mathsf{A}), u \in \mathsf{A}^*,$ *and* $(u,p) \Rightarrow^*_{\mathcal{A},T} (u,q)\}$, *and*

(iii) $i(\widehat{\mathcal{T}}) = i(\mathcal{T})$. □

PROOF For states in S, the transitions of $\widehat{\mathcal{T}}$ are the same as for $\mathcal{T}$. For $q \in Q$ and $\gamma \subseteq \mathsf{P}$, we define $\delta'((q,q),\gamma) = \texttt{true}$. For each $p,q \in Q$ with $p \neq q$ and each $\gamma \subseteq \mathsf{P}$ we realize the definition of $\text{loop}_{\mathcal{A},T}$ and define

$$\begin{aligned}
\delta'((p,q),\gamma) \quad = \quad & \bigvee \{\langle s, \varepsilon\rangle \mid p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q\} \;\vee \\
& \bigvee \{\langle (p',q'), a\rangle \mid p \xrightarrow{a}_{\mathcal{A}} p', q' \xrightarrow{\overline{a}}_{\mathcal{A}} q\} \;\vee \\
& \bigvee \{\langle (p',q'), \overline{a}\rangle \mid p \xrightarrow{\overline{a}}_{\mathcal{A}} p', q' \xrightarrow{a}_{\mathcal{A}} q\} \;\vee \\
& \bigvee \{\langle (p,r), \varepsilon\rangle \wedge \langle (r,q), \varepsilon\rangle \mid r \in Q\}.
\end{aligned}$$

We define the priority function Acc' as follows:

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

We put $\mathrm{Acc}'(p,q) = 1$ for all $p,q \in Q$ since $\widehat{\mathcal{T}}$ has to ensure that the check $(u,p,q) \in \mathrm{loop}_{\mathcal{A},T}$ terminates. Using the construction and Lemma 3.20, it is not hard to show that $\widehat{\mathcal{T}}$ is as required. ■

Step 2: Constructing an NFA $\widehat{\mathcal{A}}$ over $\widehat{\mathcal{T}}$ that can avoid detours

Now we show how to enhance the NFA $\mathcal{A} = (Q, q_0, p_0, \rightarrow_{\mathcal{A}})$ over $\mathcal{T}$ into an NFA $\widehat{\mathcal{A}}$ over $\widehat{\mathcal{T}}$ such that $\widehat{\mathcal{A}}$ can shortcut detours via test transitions that simulate $\widehat{\mathcal{T}}$ in states from $Q \times Q$: $\widehat{\mathcal{A}} = (Q, p_0, q_0, \rightarrow_{\widehat{\mathcal{A}}})$ is the extension of $\mathcal{A}$ obtained by adding, for every pair $(p,q) \in Q \times Q$, the test transition $p \xrightarrow{\widehat{\mathcal{T}},(p,q)}_{\widehat{\mathcal{A}}} q$. In the following, we prove that $\widehat{\mathcal{A}}$ is as required, i.e. runs of $\widehat{\mathcal{A}}$ can be restricted to ones that first invokes transitions *not* from A followed by transitions *not* from $\overline{\mathsf{A}}$.

Again, let us fix some $T \in \mathrm{tree}(2^{\mathsf{P}}, \mathsf{A})$. For every $u, v \in \mathsf{A}^*$ let $\inf(u,v)$ denote the longest common prefix of u and v, i.e. the lowest common ancestor of u and v in the tree T. Define the relation $\Uparrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ in the same way as $\Rightarrow_{\mathcal{A},T}$ (Definition 3.19), except that clause (3.2) is dropped. The relation $\Downarrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ is defined analogously, with clause (3.3) dropped. Thus, the relation $\Uparrow_{\mathcal{A},T}$ (resp. $\Downarrow_{\mathcal{A},T}$) allows the NFA $\mathcal{A}$ to walk only up (resp. down) the tree or to stay in the same world by executing a test transition. The following lemma states that $\widehat{\mathcal{A}}$ is as required. Its proof shows how $\widehat{\mathcal{A}}$ shortcuts detours of $\mathcal{A}$.

Lemma 3.22 *For all $u, v \in \mathsf{A}^*$ the following three statements are equivalent:*

(i) $(T,u,v) \in [\![\mathcal{A}]\!]$.

(ii) $(T,u,v) \in [\![\widehat{\mathcal{A}}]\!]$.

(iii) There exists $r \in Q$ with $(u,p_0) \Uparrow^*_{\widehat{\mathcal{A}},T} (\inf(u,v), r) \Downarrow^*_{\widehat{\mathcal{A}},T} (v, q_0)$. □

PROOF Trivially, (iii) implies (ii).

For (ii) implies (i), note that for every test-transition $p \xrightarrow{\widehat{\mathcal{T}},(p,q)}_{\widehat{\mathcal{A}}} q$ of $\widehat{\mathcal{A}}$ and every $w \in [\![\widehat{\mathcal{T}},(p,q)]\!]$ we have $(w,p,q) \in \mathrm{loop}_{\mathcal{A},T}$, hence $(w,p) \Rightarrow^*_{\mathcal{A},T} (w,q)$ by Lemma 3.20.

Finally, it remains to prove that (i) implies (iii). Assume $(T,u,v) \in [\![\mathcal{A}]\!]$. Then there exist worlds $w_0, \ldots, w_n \in \mathsf{A}^*$ and states $r_0, \ldots, r_n \in Q$ such that

$$(u,p_0) = (w_0,r_0) \Rightarrow_{\mathcal{A},T} (w_1,r_1) \;\cdots\; \Rightarrow_{\mathcal{A},T} (w_n,r_n) = (v,q_0). \tag{3.6}$$

Let $y_1, \ldots, y_k \in \{w_0, \ldots, w_n\}$ be the unique worlds such that, for some $j \in [k]$, the following conditions are satisfied:

(a) $y_1 = u$, $y_k = v$,

(b) $y_j = \inf(u, v)$,

(c) for each $1 \le i < j$, we have $y_i = y_{i+1} a_i$ for some $a_i \in \mathsf{A}$, and

(d) for each $j < i \le k$, we have $y_i = y_{i-1} a_i$ for some $a_i \in \mathsf{A}$.

Define the mapping $\phi : [k] \to [n]$ such that $\phi(i) = \max\{\ell \mid y_i = w_\ell\}$ for each $i \in [k]$. There exist states $r'_1, \ldots, r'_k \in Q$ such that the run (3.6) can be factorized as follows:

$$\begin{array}{c}(u, p_0) = (y_1, r'_1) \Rightarrow^*_{\mathcal{A},T} (y_1, r_{\phi(1)}) \Uparrow_{\mathcal{A},T} (y_2, r'_2) \Rightarrow^*_{\mathcal{A},T} (y_2, r_{\phi(2)}) \\ \cdots \Rightarrow^*_{\mathcal{A},T} (y_{j-1}, r_{\phi(j-1)}) \Uparrow_{\mathcal{A},T} (\inf(u, v), r'_j) \Rightarrow^*_{\mathcal{A},T} (\inf(u, v), r_{\phi(j)}) \Downarrow_{\mathcal{A},T} \\ \cdots \Rightarrow^*_{\mathcal{A},T} (y_{k-1}, r_{\phi(k-1)}) \Downarrow_{\mathcal{A},T} (y_k, r'_k) \Rightarrow^*_{\mathcal{A},T} (y_k, r_{\phi(k)}) = (v, q_0)\end{array}$$

Observe that the states $r_{\phi(1)}, \ldots, r_{\phi(k)}$ are the ones in the run (3.6) that $\mathcal{A}$ is in when $y_1, \ldots, y_k$ are visited the last time. By construction of $\widehat{\mathcal{A}}$, every loop $(y_i, r'_i) \Rightarrow^*_{\mathcal{A},T} (y_i, r_{\phi(i)})$ can be replaced by a test transition $r'_i \xrightarrow{\widehat{\mathcal{T}},(r'_i, r_{\phi(i)})} r_{\phi(i)}$ in $\widehat{\mathcal{A}}$. Moreover, since $\Uparrow_{\mathcal{A},T} \subseteq \Uparrow_{\widehat{\mathcal{A}},T}$ and $\Downarrow_{\mathcal{A},T} \subseteq \Downarrow_{\widehat{\mathcal{A}},T}$, we can conclude $(u, p_0) \Uparrow^*_{\widehat{\mathcal{A}},T} (\inf(u, v), r'_j) \Downarrow^*_{\widehat{\mathcal{A}},T} (v, q_0)$. ■

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

Next, we return to the construction of $\mathcal{A}(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$ from the NFAs $\mathcal{A}_i = \mathcal{A}(\pi_i)$ over the TWAPTAs $\mathcal{T}_i = \mathcal{T}(\pi_i)$, $i \in \{1, 2\}$, as fixed in (3.5). We convert $\mathcal{T}_i$ into the TWAPTA $\widehat{\mathcal{T}_i}$ and $\mathcal{A}_i$ into the NFA $\widehat{\mathcal{A}_i} = (Q_i, p_i, q_i, \to_{\widehat{\mathcal{A}_i}})$ over $\widehat{\mathcal{T}_i}$, for $i \in \{1, 2\}$. Define $\mathcal{T}(\pi_1 \cap \pi_2)$ as the disjoint union of $\widehat{\mathcal{T}_1}$ and $\widehat{\mathcal{T}_2}$. The NFA $\mathcal{A}(\pi_1 \cap \pi_2)$ is the product automaton of $\widehat{\mathcal{A}_1}$ and $\widehat{\mathcal{A}_2}$, where test transitions can be carried out asynchronously, i.e.

$$\mathcal{A}(\pi_1 \cap \pi_2) \quad = \quad (Q_1 \times Q_2, (p_1, p_2), (q_1, q_2), \to_{\mathcal{A}(\pi_1 \cap \pi_2)}),$$

with $\to_{\mathcal{A}(\pi_1 \cap \pi_2)}$ the smallest relation such that

- $r_1 \xrightarrow{a}_{\widehat{\mathcal{A}_1}} r'_1$ and $r_2 \xrightarrow{a}_{\widehat{\mathcal{A}_2}} r'_2$ implies $(r_1, r_2) \xrightarrow{a}_{\mathcal{A}(\pi_1 \cap \pi_2)} (r'_1, r'_2)$ for each $a \in \mathsf{A}$,
- $r_1 \xrightarrow{\overline{a}}_{\widehat{\mathcal{A}_1}} r'_1$ and $r_2 \xrightarrow{\overline{a}}_{\widehat{\mathcal{A}_2}} r'_2$ implies $(r_1, r_2) \xrightarrow{\overline{a}}_{\mathcal{A}(\pi_1 \cap \pi_2)} (r'_1, r'_2)$ for each $a \in \mathsf{A}$,
- $r_1 \xrightarrow{\widehat{\mathcal{T}}(\pi_1),s}_{\widehat{\mathcal{A}_1}} r'_1$ implies $(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2),s}_{\mathcal{A}(\pi_1 \cap \pi_2)} (r'_1, r_2)$ for each $r_2 \in Q_2$, and
- $r_2 \xrightarrow{\widehat{\mathcal{T}}(\pi_2),s}_{\widehat{\mathcal{A}_2}} r'_2$ implies $(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2),s}_{\mathcal{A}(\pi_1 \cap \pi_2)} (r_1, r'_2)$ for each $r_1 \in Q_1$.

The following lemma states the correctness of our construction.

Lemma 3.23 $[\![\mathcal{A}(\pi_1 \cap \pi_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$. □

PROOF Since $[\![\pi_1 \cap \pi_2]\!] = [\![\pi_1]\!] \cap [\![\pi_2]\!]$, it suffices to prove that $[\![\mathcal{A}_1]\!] \cap [\![\mathcal{A}_2]\!] = [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$. For $[\![\mathcal{A}_1]\!] \cap [\![\mathcal{A}_2]\!] \subseteq [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$, assume $(T, u, v) \in [\![\mathcal{A}_i]\!]$ for all $i \in \{1, 2\}$. Then Point (iii) of Lemma 3.22 implies the existence of a state $r_i \in Q_i$ such that $(u, p_i) \Uparrow^*_{\widehat{\mathcal{A}_i},T} (\inf(u, v), r_i) \Downarrow^*_{\widehat{\mathcal{A}_i},T} (v, q_i)$ for each $i \in \{1, 2\}$. This implies

$$(u, (p_1, p_2)) \Uparrow^*_{\mathcal{A}(\pi_1\cap\pi_2),T} (\inf(u, v), (r_1, r_2)) \Downarrow^*_{\mathcal{A}(\pi_1\cap\pi_2),T} (v, (q_1, q_2)).$$

Thus, we have $(T, u, v) \in [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$.
For $[\![\mathcal{A}(\pi_1 \cap \pi_2)]\!] \subseteq [\![\mathcal{A}_1]\!] \cap [\![\mathcal{A}_2]\!]$, observe that any run witnessing $(T, u, v) \in [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$ is a witness for $(T, u, v) \in [\![\widehat{\mathcal{A}_i}]\!]$ for each $i \in \{1, 2\}$. By Lemma 3.22, we obtain $(T, u, v) \in [\![\mathcal{A}_i]\!]$ for each $i \in \{1, 2\}$. ∎

This finishes the inductive translation of μ-ICPDL$_{\texttt{Nom}}$ formulas and programs into automata.

3.4.2 Automata size

We analyze the size of the automata constructed in the previous section and show that our translation of formulas and programs involves at most a single exponential blow-up. For analyzing the fragment loop-CPDL of μ-ICPDL$_{\texttt{Nom}}$ later on, it is useful to carry out a careful analysis that takes into account the nestings of the intersection operator. Since mere counting of occurrences of the intersection operator would be too coarse in order to get sharp complexity bounds for loop-CPDL, we will introduce a finer measure, which we call intersection width. This measure comes in two versions. The first version is used to estimate the size of NFAs, and the second is for TWAPTAs.

First let us consider the NFA version. The *intersection width* $\mathrm{iw}(\pi)$ of an μ-ICPDL$_{\texttt{Nom}}$ program π is defined inductively,

$$\begin{aligned}
\mathrm{iw}(a) = \mathrm{iw}(\overline{a}) &= 1 \quad \text{for each } a \in \mathsf{A}, \\
\mathrm{iw}(\theta?) &= 1, \\
\mathrm{iw}(\pi_1 \cup \pi_2) = \mathrm{iw}(\pi_1 \circ \pi_2) &= \max\{\mathrm{iw}(\pi_1), \mathrm{iw}(\pi_2)\}, \\
\mathrm{iw}(\pi^*) &= \mathrm{iw}(\pi), \text{and} \\
\mathrm{iw}(\pi_1 \cap \pi_2) &= \mathrm{iw}(\pi_1) + \mathrm{iw}(\pi_2).
\end{aligned}$$

Note that iw is non-montonic with respect to $\prec$: We have $\mathrm{iw}(\theta?) = 1$ although there might exist $\langle\pi\rangle\psi \prec \theta$ with $\mathrm{iw}(\pi) > 1$.
For an NFA $\mathcal{A} = (Q, p_0, q_0, \rightarrow_{\mathcal{A}})$ over some TWAPTA, we define the *size* $|\mathcal{A}|$ as $|Q|$. Note that the size of the TWAPTA, over which $\mathcal{A}$ is defined, is not taken into account.
Recall that φ_0 is our original μ-ICPDL$_{\texttt{Nom}}$ formula want to translate into a TWAPTA $\mathcal{T}(\varphi_0)$.

Lemma 3.24 *For every program $\pi \in subp(\varphi_0)$ we have $|\mathcal{A}(\pi)| \leq 2 \cdot |\pi|^{\mathrm{iw}(\pi)}$.* □

PROOF The proof is by induction on $\prec$.

Base cases. Let $\pi = a$ or $\pi = \overline{a}$ with $a \in \mathsf{A}$. Then $|\mathcal{A}(\pi)| = 2 = 2 \cdot |\pi|^{\mathrm{iw}(\pi)}$.
Inductive cases.

Let $\pi = \theta?$. Then $|\mathcal{A}(\theta?)| = 2 \leq 2 \cdot |\pi|^{\mathrm{iw}(\pi)}$.

Let $\pi = \pi_1^*$. The standard construction for Kleene star and induction hypothesis yield $|\mathcal{A}(\pi)| = |\mathcal{A}(\pi_1)| \leq 2 \cdot |\pi_1|^{\mathrm{iw}(\pi_1)} \leq 2 \cdot |\pi|^{\mathrm{iw}(\pi)}$.

Let $\pi = \pi_1 \cup \pi_2$ or $\pi = \pi_1 \circ \pi_2$. By the standard constructions for union and composition,

$$\begin{array}{rcl}
|\mathcal{A}(\pi)| & = & |\mathcal{A}(\pi_1)| + |\mathcal{A}(\pi_2)| \\
& \overset{\text{IH}}{\leq} & 2 \cdot |\pi_1|^{\mathrm{iw}(\pi_1)} + 2 \cdot |\pi_2|^{\mathrm{iw}(\pi_2)} \\
& \leq & 2 \cdot |\pi_1|^{\mathrm{iw}(\pi)} + 2 \cdot |\pi_2|^{\mathrm{iw}(\pi)} \\
& \leq & 2 \cdot (|\pi_1| + |\pi_2|)^{\mathrm{iw}(\pi)} \\
& \leq & 2 \cdot |\pi|^{\mathrm{iw}(\pi)}.
\end{array}$$

Let $\pi = \pi_1 \cap \pi_2$. Then

$$\begin{array}{rcl}
|\mathcal{A}(\pi)| & = & |\mathcal{A}(\pi_1)| \cdot |\mathcal{A}(\pi_2)| \\
& \overset{\text{IH}}{\leq} & 2 \cdot |\pi_1|^{\mathrm{iw}(\pi_1)} \cdot 2 \cdot |\pi_2|^{\mathrm{iw}(\pi_2)} \\
& \leq & 2 \cdot (|\pi_1| + |\pi_2|)^{\mathrm{iw}(\pi_1)+\mathrm{iw}(\pi_2)} \\
& \leq & 2 \cdot |\pi|^{\mathrm{iw}(\pi)}.
\end{array}$$

The inequality $2 \cdot |\pi_1|^{\mathrm{iw}(\pi_1)} \cdot |\pi_2|^{\mathrm{iw}(\pi_2)} \leq (|\pi_1| + |\pi_2|)^{\mathrm{iw}(\pi_1)+\mathrm{iw}(\pi_2)}$ follows from the binomial theorem. ■

We now define the second version of intersection width, where the difference is in the treatment of the test operator. If α is a μ-ICPDL$_{\texttt{Nom}}$ program or formula, then

$$\mathrm{IW}(\alpha) \; = \; \begin{cases} 1 & \text{if } \mathrm{subp}(\alpha) = \emptyset \\ \max\{\mathrm{iw}(\pi) \mid \pi \in \mathrm{subp}(\alpha)\} & \text{otherwise.} \end{cases}$$

Observe that IW is monotone with respect to $\prec$. Recall that, for a TWAPTA $\mathcal{T}$, $i(\mathcal{T})$ denotes its index and its size $|\mathcal{T}|$ was defined to be the number of its states. In the following lemma, for each formula or program α, we estimate the size and the index of the resulting TWAPTA $\mathcal{T}(\alpha)$.

Lemma 3.25 *For every $\alpha \in subf(\varphi_0) \cup subp(\varphi_0)$ we have*

- $|\mathcal{T}(\alpha)| \leq |\alpha| + 8 \cdot |\alpha| \cdot |\alpha|^{2 \cdot \mathrm{IW}(\alpha)}$ *and*
- $i(\mathcal{T}(\alpha)) \leq 2 \cdot |\alpha| - 1.$ □

PROOF The proof is by induction on $\prec$.

Base cases.

Let $\pi = a$ or $\pi = \overline{a}$, with $a \in \mathsf{A}$. Then $|\mathcal{T}(\pi)| = 1 \leq |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2 \cdot \mathrm{IW}(\pi)}$ and $i(\mathcal{T}(\pi)) = 1 = 2 \cdot |\pi| - 1$.

Let $\psi \in \mathsf{P}$. Then $|\mathcal{T}(\psi)| = 1 \leq |\psi| + 8 \cdot |\psi| \cdot |\psi|^{2 \cdot \mathrm{IW}(\psi)}$ and $i(\mathcal{T}(\psi)) = 1 = 2 \cdot |\psi| - 1$.

Inductive cases.

Let $\psi = \neg\theta$. The complementation construction yields

$$|\mathcal{T}(\psi)| = |\mathcal{T}(\theta)| + 1 \overset{\text{IH}}{\leq} |\theta| + 1 + 8 \cdot |\theta| \cdot |\theta|^{2\cdot\text{IW}(\theta)} \leq |\psi| + 8 \cdot |\psi| \cdot |\psi|^{2\cdot\text{IW}(\psi)}$$

and $i(\mathcal{T}(\psi)) = i(\mathcal{T}(\theta)) + 1 \overset{\text{IH}}{\leq} 2 \cdot |\theta| - 1 + 1 \leq 2 \cdot |\psi| - 1$.

Let $\psi = \langle\pi\rangle\theta$. Then

$$\begin{array}{rcl}
|\mathcal{T}(\psi)| & = & |\mathcal{A}(\pi)| + |\mathcal{T}(\pi)| + |\mathcal{T}(\theta)| + 1 \\
& \overset{\text{Lemma 3.24, IH}}{\leq} & 2 \cdot |\pi|^{\text{iw}(\pi)} + |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot\text{IW}(\pi)} + |\theta| + 8 \cdot |\theta| \cdot |\theta|^{2\cdot\text{IW}(\theta)} + 1 \\
& \leq & 2 \cdot |\psi|^{2\cdot\text{IW}(\psi)} + |\psi| + 8 \cdot |\pi| \cdot |\psi|^{2\cdot\text{IW}(\psi)} + 8 \cdot |\theta| \cdot |\psi|^{2\cdot\text{IW}(\psi)} \\
& \leq & |\psi| + 8 \cdot (1 + |\pi| + |\theta|) \cdot |\psi|^{2\cdot\text{IW}(\psi)} \\
& = & |\psi| + 8 \cdot |\psi| \cdot |\psi|^{2\cdot\text{IW}(\psi)}.
\end{array}$$

Moreover, $i(\mathcal{T}(\psi)) = \max\{i(\mathcal{T}(\pi)), i(\mathcal{T}(\theta))\} \overset{\text{IH}}{\leq} \max\{2 \cdot |\pi| - 1, 2 \cdot |\theta| - 1\} \leq 2 \cdot |\psi| - 1$.

Let $\psi = \mu X.\theta$. Then $|\mathcal{T}(\psi)| = |\mathcal{T}(\theta)| + 1$ and hence $|\mathcal{T}(\psi)|$ is as required. Moreover $i(\mathcal{T}(\psi)) = i(\mathcal{T}(\theta)) + 2 \overset{\text{IH}}{\leq} 2 \cdot |\theta| - 1 + 2 = 2 \cdot (|\theta| + 1) - 1 = 2 \cdot |\psi| - 1$.

Let $\psi = @_p\theta$. That $|\mathcal{T}(\psi)|$ and $i(\mathcal{T}(\psi))$ are as required follows easily from induction hypothesis.

Let $\pi = \theta?$. That $|\mathcal{T}(\pi)|$ and $i(\mathcal{T}(\pi))$ are as required follows easily from induction hypothesis.

Let $\pi = \pi_1^*$. That $|\mathcal{T}(\pi)|$ and $i(\mathcal{T}(\pi))$ are as required follows easily from induction hypothesis.

Let $\pi = \pi_1 \cup \pi_2$ or $\pi = \pi_1 \circ \pi_2$. Then

$$\begin{array}{rcl}
|\mathcal{T}(\pi)| & = & |\mathcal{T}(\pi_1)| + |\mathcal{T}(\pi_2)| \\
& \overset{\text{IH}}{\leq} & |\pi_1| + 8 \cdot |\pi_1| \cdot |\pi_1|^{2\cdot\text{IW}(\pi_1)} + |\pi_2| + 8 \cdot |\pi_2| \cdot |\pi_2|^{2\cdot\text{IW}(\pi_2)} \\
& \leq & |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot\text{IW}(\pi)}.
\end{array}$$

Moreover, $i(\mathcal{T}(\pi)) = \max\{i(\mathcal{T}(\pi_1)), i(\mathcal{T}(\pi_2))\} \overset{\text{IH}}{\leq} \max\{2 \cdot |\pi_1| - 1, 2 \cdot |\pi_2| - 1\} \leq 2 \cdot |\pi| - 1$.

Let $\pi = \pi_1 \cap \pi_2$. Then

$$\begin{array}{rcl}
|\mathcal{T}(\pi)| & = & |\mathcal{T}(\pi_1)| + |\mathcal{T}(\pi_2)| + |\mathcal{A}(\pi_1)|^2 + |\mathcal{A}(\pi_2)|^2 \\
& \overset{\text{Lemma 3.24,IH}}{\leq} & |\pi_1| + 8 \cdot |\pi_1| \cdot |\pi_1|^{2\cdot\text{IW}(\pi_1)} + |\pi_2| + 8 \cdot |\pi_2| \cdot |\pi_2|^{2\cdot\text{IW}(\pi_2)} + \\
& & 4 \cdot |\pi_1|^{2\cdot\text{IW}(\pi_1)} + 4 \cdot |\pi_2|^{2\cdot\text{IW}(\pi_2)} \\
& \leq & |\pi| + 8 \cdot |\pi_1| \cdot |\pi|^{2\cdot\text{IW}(\pi)} + 8 \cdot |\pi_2| \cdot |\pi|^{2\cdot\text{IW}(\pi)} + 8 \cdot |\pi|^{2\cdot\text{IW}(\pi)} \\
& = & |\pi| + 8 \cdot (|\pi_1| + |\pi_2| + 1)|\pi|^{2\cdot\text{IW}(\pi)} \\
& = & |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot\text{IW}(\pi)}.
\end{array}$$

Finally, $i(\mathcal{T}(\pi)) = \max\{i(\mathcal{T}(\pi_1)), i(\mathcal{T}(\pi_2))\} \overset{\text{IH}}{\leq} \max\{2 \cdot |\pi_1| - 1, 2 \cdot |\pi_2| - 1\} \leq 2 \cdot |\pi| - 1.$ ■

In the following remark we roughly estimate the size of the transition function of the constructed TWAPTAs.

Remark 3.26 *Let $\alpha \in subf(\varphi_0) \cup subp(\varphi_0)$. A straightforward estimate together with Lemma 3.25 implies $|\delta(\mathcal{T}(\alpha))| \leq |\mathsf{A}| \cdot 2^{|\mathsf{P}|} \cdot |\alpha|^{\mathcal{O}(\mathrm{IW}(\alpha))}$. Since $|\mathsf{P}|, |\mathsf{A}| \leq |\alpha|$ we obtain*

$$|\delta(\mathcal{T}(\alpha))| \quad \leq \quad 2^{|\alpha|} \cdot |\alpha|^{\mathcal{O}(\mathrm{IW}(\alpha))}.$$

This completes the estimation of the size of automata.

3.4.3 The upper bound

The algorithm for deciding ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ works as follows. Given a μ-ICPDL$_{\mathtt{Nom}}$ formula φ_0 and a TWAPTA $\mathcal{T}_0$ over 2^{P}-labeled A-trees, we convert φ_0 into a TWAPTA $\mathcal{T}(\varphi_0)$ such that $[\![\mathcal{T}(\varphi_0)]\!] = [\![\varphi_0]\!]$, as described in Section 3.4.1. Then we construct a TWAPTA $\mathcal{T}$ such that $L(\mathcal{T}) = L(\mathcal{T}_0) \cap L(\mathcal{T}(\varphi_0))$. For alternating automata, intersection is trivial and $\mathcal{T}$ can be computed in linear time from $\mathcal{T}_0$ and $\mathcal{T}(\varphi_0)$. Clearly, $L(\mathcal{T}) \neq \emptyset$ if and only if there is a tree $T \in L(\mathcal{T}_0)$ with $\varepsilon \in [\![\varphi_0]\!]_T$. By combining the upper bounds for $|\mathcal{T}(\varphi_0)|$ and $i(\mathcal{T}(\varphi_0))$ (Lemma 3.25), the upper bound for $|\delta(\mathcal{T}(\varphi_0))|$ (Remark 3.26), and the complexity upper bound for deciding non-emptiness of TWAPTAs (Theorem 3.11), we obtain the following upper bound for ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$.

Theorem 3.27 *For a TWAPTA $\mathcal{T}_0$ with transition function δ and a μ-ICPDL$_{\mathtt{Nom}}$ formula φ_0, we can decide in time $\exp(|\mathcal{T}_0| + i(\mathcal{T}_0) + |\varphi_0|^{\mathrm{IW}(\varphi_0)}) \cdot \mathrm{poly}(|\delta|)$ whether there exists some $T \in L(\mathcal{T}_0)$ such that $\varepsilon \in [\![\varphi_0]\!]_T$. Hence, ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is in* 2EXP. □

Note that Theorem 3.27 yields a single exponential upper bound if the intersection width of the input formula is bounded by a constant.

3.5 Satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is 2EXP-complete

In this section, we bring together the results established in the previous sections for proving upper bounds for the satisfiability problem in μ-ICPDL$_{\mathtt{Nom}}$ and fragments.

Theorem 3.28 *Satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is* 2EXP*-complete. Moreover, given a μ-ICPDL$_{\mathtt{Nom}}$ formula φ, we can decide satisfiability of φ in time $\exp(|\varphi|^{\mathrm{IW}(\varphi)})$.* □

PROOF The 2EXP lower bound for satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ already holds for IPDL due to Theorem 2.21.

For the 2EXP upper bound, let φ be some μ-ICPDL$_{\mathtt{Nom}}$ formula. By Theorem 3.18 we can compute a TWAPTA $\mathcal{T}_0$ and a μ-ICPDL$_{\mathtt{Nom}}$ formula φ_0 in time exponential in $|\varphi|$ such that the following five conditions hold:

(1) φ is satisfiable if and only if there exists some $T \in L(\mathcal{T}_0)$ such that $\varepsilon \in [\![\varphi_0]\!]_T$,

(2) $|\varphi_0| \in \mathrm{poly}(|\varphi|)$,

(3) $|\mathcal{T}_0| \in \text{poly}(|\varphi|)$,

(4) $|\delta(\mathcal{T}_0)| \in \exp(|\varphi|)$, and

(5) $i(\mathcal{T}_0) \in \mathcal{O}(1)$.

Observe that the translation from φ to φ_0 has the property that $\mathrm{IW}(\varphi) = \mathrm{IW}(\varphi_0)$. By finally applying Theorem 3.27 to $\mathcal{T}_0$ and φ_0, we obtain that satisfiability of φ can altogether be decided in time $\exp(|\varphi|^{\mathrm{IW}(\varphi)})$. ■

Since the reduction from satisfiability to ω-regular tree satisfiability preserves the intersection width of formulas, we obtain the following corollary.

Corollary 3.29 *For every constant $c \geq 1$, satisfiability in*

$$\{\varphi \mid \varphi \textit{ is a } \mu\textit{-ICPDL}_{\texttt{Nom}} \textit{ formula with IW}(\varphi) \leq c\}$$

is EXP*-complete.* □

As every formula of loop-CPDL is easily translatable into an ICPDL formula of intersection width at most two, we obtain the following corollary.

Corollary 3.30 *Satisfiability in loop-CPDL is* EXP*-complete.* □

3.6 Infinite state model checking of μ-ICPDL$_{\texttt{Nom}}$

In this section we investigate the combined, expression and data complexity of model checking μ-ICPDL$_{\texttt{Nom}}$ over basic process algebras (BPAs), pushdown systems (PDSs), and prefix-recognizable systems (PRSs).

Recall that in the context of infinite state model checking, atomic propositions and in particular nominals need not to be accounted for in the finite description of the structures. This simplifies notation and can be done without loss of generality since an atomic proposition or nominal p is easily simulated using a reflexive loop of a new atomic program a_p, which can then be tested by the formula $\langle a_p \rangle \texttt{true}$. Thus, in the following section we assume without loss of generality that `true` and `false` are the only atomic formulas that occur in formulas. Furthermore, we assume that $\mathsf{A} \subseteq \mathbb{A}$ always denotes a finite set of atomic programs.

Firstly, we establish upper complexity bounds in Section 3.6.1 by a reduction to ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$. Secondly, matching lower bounds are established in Section 3.6.2. Table 3.1 summarizes our results.

3.6.1 Upper bounds: Reduction to ω-regular tree satisfiability

In this section, we reduce model checking μ-ICPDL$_{\texttt{Nom}}$ over prefix-recognizable systems to ω-regular tree satisfiability in μ-ICPDL$_{\texttt{Nom}}$. For this, let φ be an input μ-ICPDL$_{\texttt{Nom}}$ formula, let $\mathcal{Z} = (\mathsf{A}, \phi)$ a PRS over the atomic programs $\mathsf{A}_{\mathcal{Z}}$, and let $x_0 \in \mathsf{A}^*$ a world of $K(\mathcal{Z})$. We want to check whether $x_0 \in [\![\varphi]\!]_{K(\mathcal{Z})}$. Let $\mathsf{P} = \{\perp\}$ be a dummy node labeling set. Let the 2^{P}-labeled

<table>
<tr><th colspan="2"></th><th>BPA/PDS/PRS</th></tr>
<tr><td rowspan="3">IPDL, μ-ICPDL$_{\texttt{Nom}}$
constant intersection width</td><td>data</td><td rowspan="4">EXP-complete</td></tr>
<tr><td>expression</td></tr>
<tr><td>combined</td></tr>
<tr><td rowspan="3">μ-ICPDL$_{\texttt{Nom}}$</td><td>data</td></tr>
<tr><td>expression</td><td rowspan="2">2EXP-complete</td></tr>
<tr><td>combined</td></tr>
</table>

Table 3.1: Complexity of infinite state model checking of μ-ICPDL$_{\texttt{Nom}}$

A-tree $T_0 : \mathsf{A}^* \to 2^{\mathsf{P}}$ be defined as $T_0(v) = \emptyset$ for each $v \in \mathsf{A}^*$. The Kripke structure that corresponds to T_0 is depicted in Figure 3.1 and is formally defined as $(\mathsf{A}^*, \{\to_a \mid a \in \mathsf{A}\}, \{T_\perp\})$, where $\to_a = \{(u, ua) \mid u \in \mathsf{A}^*\}$ for each $a \in \mathsf{A}$ and $T_\perp = \emptyset$. We compute from $\mathcal{Z}$, x_0, and φ a μ-ICPDL$_{\texttt{Nom}}$ formula φ' over 2^{P}-labeled A-trees such that $x_0 \in [\![\varphi]\!]_{K(\mathcal{Z})}$ if and only if $\varepsilon \in [\![\varphi']\!]_{T_0}$. Let us assume that for each $a \in \mathsf{A}_{\mathcal{Z}}$ the prefix-recognizable relation

$$\phi(a) \quad = \quad \bigcup_{i=1}^{n_a} (L(\alpha_i^a) \times L(\beta_i^a))L(\gamma_i^a)$$

is given by a collection of triples of regular expressions $(\alpha_i^a, \beta_i^a, \gamma_i^a)_{i \in [n_a]}$. We define the μ-ICPDL$_{\texttt{Nom}}$ formula $\varphi_{\mathcal{Z}}$ to be obtained from φ by replacing every occurrence of $a \in \mathsf{A}$ by

$$\bigcup_{i=1}^{n_a} \overleftarrow{\alpha_i^a} \circ \left(\langle \overleftarrow{\gamma_i^a} \rangle \bigwedge_{a \in \mathsf{A}} \neg \langle \overline{a} \rangle \texttt{true} \right) ? \circ \overrightarrow{\beta_i^a},$$

where the μ-ICPDL$_{\texttt{Nom}}$ programs $\overleftarrow{\alpha_i^a}, \overleftarrow{\gamma_i^a}$ and $\overrightarrow{\beta_i^a}$ respectively, are inductively defined as follows,

$$\begin{array}{rclcrcll}
\overleftarrow{a} & = & \overline{a}, & & \overrightarrow{a} & = & a, & \text{for each } a \in \mathsf{A}_{\mathcal{Z}}, \\
\overleftarrow{\sigma_1 + \sigma_2} & = & \overleftarrow{\sigma_1} \cup \overleftarrow{\sigma_2}, & & \overrightarrow{\sigma_1 + \sigma_2} & = & \overrightarrow{\sigma_1} \cup \overrightarrow{\sigma_2}, & \\
\overleftarrow{\sigma_1 \cdot \sigma_2} & = & \overleftarrow{\sigma_1} \circ \overleftarrow{\sigma_2}, & & \overrightarrow{\sigma_1 \cdot \sigma_2} & = & \overrightarrow{\sigma_2} \circ \overrightarrow{\sigma_1}, & \\
\overleftarrow{\sigma^*} & = & \overleftarrow{\sigma}^*, & & \overrightarrow{\sigma^*} & = & \overrightarrow{\sigma}^*. &
\end{array}$$

We put $\varphi' = \langle \overrightarrow{x_0} \rangle \varphi_{\mathcal{Z}}$. Clearly, φ' can be interpreted in T_0. It is not hard to verify that $x_0 \in [\![\varphi]\!]_{K(\mathcal{Z})}$ if and only if $\varepsilon \in [\![\varphi']\!]_{T_0}$. In order to give a reduction to ω-regular tree satisfiability, it remains to construct a TWAPTA $\mathcal{T}$ such that $L(\mathcal{T}) = \{T_0\}$, which is a triviality. Thus, $x_0 \in [\![\varphi]\!]_{K(\mathcal{Z})}$ if and only if there is a $T \in L(\mathcal{T})$ with $\varepsilon \in [\![\varphi']\!]_T$. From Theorem 3.27 we obtain the following theorem.

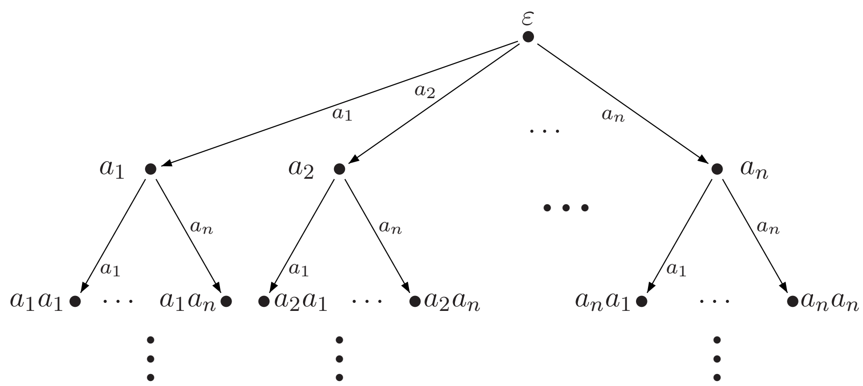


Figure 3.1: The complete A-tree

Theorem 3.31 *For a PRS $\mathcal{Z}$, a world x_0 of $K(\mathcal{Z})$, and a μ-ICPDL$_{\mathtt{Nom}}$ formula φ, we can decide in time $\exp((|x_0| + |\varphi| + |\mathcal{Z}|)^{\mathrm{IW}(\varphi)})$ whether $x_0 \in [\![\varphi]\!]_{K(\mathcal{Z})}$. Hence, model checking PRSs in μ-ICPDL$_{\mathtt{Nom}}$ is in* 2EXP *with respect to combined (and thus also expression) complexity, and in* EXP *with respect to data complexity.* □

Since the formula φ' has the same intersection width as φ, we obtain the following corollary.

Corollary 3.32 *For every constanct c the combined complexity of model checking μ-ICPDL$_{\mathtt{Nom}}$ formulas of intersection width at most c over PRSs is in* EXP. □

3.6.2 Lower bounds

We prove two lower bounds for infinite state model checking of fragments of μ-ICPDL$_{\mathtt{Nom}}$. The first result establishes EXP-hardness of the data complexity of model checking loop-PDL over BPAs. This is somewhat surprising since the data complexity of the modal μ-calculus over pushdown systems is in P [125]. From the proof we can deduce a corollary that the data complexity of test-free IPDL over BPAs is EXP-hard too. We do not show EXP-hardness of the expression complexity of model checking IPDL formulas of constant intersection over BPAs since in Chapter 5 we prove that the corresponding lower bound already holds for PDL (Corollary 5.14), see also Table 5.1. The second result that is proven in this section is that the expression complexity of model checking test-free IPDL over BPAs is 2EXP-hard. Thus, we obtain tight complexity bounds as summarized in Table 3.1. Both lower bounds are proved by a reduction of the membership problem of space-bounded alternating Turing machines.

Data complexity in loop-PDL

We start with a summary of the main ideas behind the proof, which follows [124]. Let $\mathcal{M} = (Q, \Gamma, \Sigma, q_0, \delta, \Box)$ be a $p(n)$-space bounded ATM (where $p(n)$ is some polynomial) whose membership problem is EXP-hard, and let $w \in \Sigma^*$ an input of length n. Our aim is to construct a BPA

$\mathcal{X}$ depending on $\mathcal{M}$ and w and an IPDL formula φ depending only on $\mathcal{M}$ such that $w \in L(\mathcal{M})$ if and only if $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$. We define $\mathcal{X}$ such that certain worlds of $K(\mathcal{X})$ represent sequences of configurations of $\mathcal{M}$ of length at most $p(n)$, separated by certain markers. These worlds correspond to nodes of computation trees of $\mathcal{M}$. Our formula φ verifies the existence of an acceptance tree of $\mathcal{M}$ on w by traversing $K(\mathcal{X})$ in a depth-first and left-to-right fashion.

To give more details, let us first introduce the mentioned markers,

- $\mathrm{Dir}_\forall = \{L(\mu_1, \mu_2), R(\mu_1, \mu_2) \mid (\mu_1, \mu_2) \in \delta(Q_\forall, \Gamma)\}$ is the set of *universal direction markers*,
- $\mathrm{Dir}_\exists = \{E(\mu_1), E(\mu_2) \mid (\mu_1, \mu_2) \in \delta(Q_\exists, \Gamma)\}$ is the set of *existential direction markers*, and
- $\mathrm{Dir} = \mathrm{Dir}_\forall \cup \mathrm{Dir}_\exists$ is the set of *direction markers*.

Since $\mathcal{X}$ is a BPA, we will henceforth refer to the worlds of $K(\mathcal{X})$ as the stack. As already mentioned, the stack consists of a sequence of configurations (in a suitable encoding), separated by direction markers. In particular, a marker $L(\mu_1, \mu_2)$ on top of a universal configuration means that we are currently exploring the left child of that node, whereas $R(\mu_1, \mu_2)$ refers to the right child.

The main ingredient to the formula φ is a program "$\mathrm{traverse}$", which implements the individual steps of a depth-first and left-to-right traversal of computation trees. It starts from the world in which the stack consists of the initial configuration of $\mathcal{M}$ on input w (without any direction markers), and then applies the following rules:

(1) If the top of the stack consists of a universal configuration c in current state $q \in Q_\forall$, scanning a symbol γ, and $\delta(q, \gamma) = (\mu_1, \mu_2)$, then the direction marker $L(\mu_1, \mu_2)$ is pushed on the stack and on top of it the left successor configuration of c, i.e. the one that corresponds to the move μ_1.

(2) If the top of the stack consists of an existential configuration c in current state $q \in Q_\exists$, scanning a symbol γ, and $\delta(q, \gamma) = (\mu_1, \mu_2)$, then we nondeterministically push a direction marker $E(\mu)$ and on top of it the successor configuration of c that corresponds to the move $\mu \in \{\mu_1, \mu_2\}$.

(3) If the top of the stack consists of a configuration with current state $q \in Q_\forall$, scanning a symbol γ, and $\delta(q, \gamma) = \bot$, then the configuration can be popped.

(4) If the top of the stack consists of a universal configuration c with a direction marker $L(\mu_1, \mu_2)$ on top, then $L(\mu_1, \mu_2)$ is replaced with $R(\mu_1, \mu_2)$ and on top of it the right successor configuration of c, i.e. the one that corresponds to the move μ_2.

(5) If the top of the stack consists of a universal configuration with a direction marker $R(\mu_1, \mu_2)$ on top or of an existential configuration with a direction marker $E(\mu)$ on top, then both the configuration and the marker are popped.

By a straightforward induction, one can prove that the following two statements are equivalent:

- The ATM $\mathcal{M}$ has an acceptance tree on input w.
- In $K(\mathcal{X})$ from the initial configuration the empty stack can be reached by applying rules (1)–(5) from above.

Theorem 3.33 *There exists a loop-PDL formula φ such that the following problem is* EXP-*hard:*
INPUT: A BPA $\mathcal{X}$.
QUESTION: $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$? □

PROOF Let $\mathcal{M} = (Q, \Gamma, \Sigma, q_0, \delta, \Box)$ be a $p(n)$-space bounded ATM (where $p(n)$ is a polynomial) with an EXP-hard membership problem. Let $w = w_1 w_2 \cdots w_n \in \Sigma^*$ be an input of length n. We give a logspace computable BPA $\mathcal{X} = \mathcal{X}(\mathcal{M}, w)$ and a loop-PDL formula $\varphi = \varphi(\mathcal{M})$ such that $w \in L(\mathcal{M})$ if and only if $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$. Let $N = p(n)$, $\Omega = Q \cup \Gamma$, and $\Omega_N = \{a(i) \mid a \in \Omega, i \in [0, N]\}$. A configuration $c = a_0 a_1 \cdots a_{i-1} q a_{i+1} \cdots a_N \in \bigcup_{0 \le i \le N-1} \Gamma^i Q \Gamma^{N-i}$ is encoded by the string

$$a_0(0)a_1(1)\cdots a_{i-1}(i-1)q(i)a_{i+1}(i+1)\cdots a_N(N).$$

In the following, we confuse configurations and their encodings. The stack alphabet of $\mathcal{X}$ is $\Gamma_{\mathcal{X}} = \Omega_N \cup \mathrm{Dir}$. Let us now define the rewriting rules of $\mathcal{X}$. The set of atomic programs A of $\mathcal{X}$ is implicitly given as the set of all programs occurring in the following rules:

$$\begin{array}{rll}
\varepsilon \xrightarrow{\text{input}}_{\mathcal{X}} & q_0(0)w_1(1)\cdots w_n(n)\Box(n+1)\cdots\Box(N) & \\
\varepsilon \xrightarrow{d}_{\mathcal{X}} & d & \text{for all } d \in \mathrm{Dir} \\
d \xrightarrow{\text{push}}_{\mathcal{X}} & a(N)d & \text{for all } d \in \mathrm{Dir} \text{ and } a \in \Omega \\
a(i) \xrightarrow{\text{push}}_{\mathcal{X}} & a'(i-1)a(i) & \text{for all } i \in [1, N] \text{ and } a, a' \in \Omega \\
d \xrightarrow{\text{check}_{\mathrm{Dir}}}_{\mathcal{X}} & d & \text{for all } d \in \mathrm{Dir} \\
a(0) \xrightarrow{\text{check}_0}_{\mathcal{X}} & a(0) & \text{for all } a \in \Omega \\
a(N) \xrightarrow{\text{pop}_N}_{\mathcal{X}} & \varepsilon & \text{for all } a \in \Omega \\
a(i) \xrightarrow{\bar{a}}_{\mathcal{X}} & \varepsilon & \text{for all } i \in [0, N-1] \text{ and } a \in \Omega \\
d \xrightarrow{\bar{d}}_{\mathcal{X}} & \varepsilon & \text{for all } d \in \mathrm{Dir} \\
a(i) \xrightarrow{\text{copy}}_{\mathcal{X}} & a(i)a(i) & \text{for all } i \in [0, N] \text{ and } a \in \Omega \\
a(i) \xrightarrow{\text{shift}}_{\mathcal{X}} & a(i)x & \text{for all } i \in [0, N], a \in \Omega, \text{ and } x \in \Gamma_{\mathcal{X}} \\
a(i) \xrightarrow{(a/b)}_{\mathcal{X}} & b(i) & \text{for all } i \in [0, N] \text{ and } a, b \in \Omega
\end{array}$$

Observe that not all worlds in $K(\mathcal{X})$ are proper encodings of configuration sequences separated by direction markers. This cannot be avoided and needs to be taken care of in the formula φ. Before we can define it, we need several auxiliary programs. In the following, for $X \subseteq \mathrm{Dir} \cup \Omega$, we use pop_X to denote $\bigcup_{x \in X} \overline{x}$.

- The program $\overline{\mathrm{conf}} = \mathrm{pop}_\Omega^* \circ \mathrm{pop}_N$ pops the top configuration from the stack.

- The program conf pushes a potential configuration onto the stack, i.e., a string of the form $a_0(0)a_1(1)\cdots a_N(N)$, where $a_i \in \Omega$:

$$\mathrm{conf} \quad = \quad \mathrm{check}_{\mathrm{Dir}} \circ \mathrm{push}^* \circ \mathrm{check}_0$$

- Assume that the top of the stack consists of a suffix of a configuration followed by a direction marker $d \in \mathrm{Dir}$ underneath, followed by a complete configuration, i.e., the top of the stack (reading from left to right) has the following form:

$$a_k(k)a_{k+1}(k+1)\cdots a_N(N)da'_0(0)\cdots a'_N(N)$$

with $a_i, a'_i \in \Omega$. For $a, b \in \Omega$, we give a program $\mathrm{test}_{a,b}$ that is executable only if $a_k = a$ and $a'_k = b$, and that pops $a_k(k)$:

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

- For $\mu \in \mathrm{Moves}$, the program check_μ is executable only if the top of the stack has the form $c'dc$, where c and c' are configurations, $d \in \mathrm{Dir}$, and c' is the successor of c according to the move μ. Execution of check_μ leaves the stack unchanged. We only treat the case $\mu = (p, b, \leftarrow)$ explicitly:

$$\begin{aligned} \mathrm{check}_\mu \;=\; \mathrm{loop} \Bigg[& \Big(\bigcup_{a\in\Gamma} \mathrm{test}_{a,a}\Big)^* \circ \bigcup_{q\in Q, a, a' \in \Gamma} \big(\mathrm{test}_{p,a'} \circ \mathrm{test}_{a',q} \circ \mathrm{test}_{b,a}\big) \circ \\ & \Big(\bigcup_{a\in\Gamma} \mathrm{test}_{a,a}\Big)^* \circ \mathrm{conf} \Bigg]? \end{aligned}$$

- For $q \in Q$ and $a \in \Gamma$, the program $\mathrm{scan}_{q,a}$ checks if the top configuration on the stack is in current state q, scanning the symbol a:

$$\mathrm{scan}_{q,a} \quad = \quad \mathrm{loop}(\mathrm{pop}_\Gamma^* \circ \overline{q} \circ \overline{a} \circ \mathrm{push}^*)?$$

Next, we define the main ingredient to the formula φ: the program $\mathrm{traverse}$. As discussed above, $\mathrm{traverse}$ travels one edge in an accepting configuration tree of $\mathcal{M}$ on w, proceeding in a depth-first and left-to-right fashion.

$$\begin{aligned}
\text{traverse} \quad = \quad & \bigcup_{\substack{q\in Q_\forall, a\in\Gamma\\ \delta(q,a)=(\mu_1,\mu_2)}} \text{scan}_{q,a}\circ L(\mu_1,\mu_2)\circ\text{conf}\circ\text{check}_{\mu_1}\\
\cup & \bigcup_{\substack{q\in Q_\exists, a\in\Gamma\\ \delta(q,a)=(\mu_1,\mu_2),\mu\in\{\mu_1,\mu_2\}}} \text{scan}_{q,a}\circ E(\mu)\circ\text{conf}\circ\text{check}_{\mu}\\
\cup & \bigcup_{\substack{q\in Q_\forall, a\in\Gamma\\ \delta(q,a)=\bot}} \text{scan}_{q,a}\circ\overline{\text{conf}}\\
\cup & \bigcup_{(\mu_1,\mu_2)\in\delta(Q_\forall,\Gamma)} \overline{L(\mu_1,\mu_2)}\circ R(\mu_1,\mu_2)\circ\text{conf}\circ\text{check}_{\mu_2}\\
\cup & \bigcup_{R(\mu_1,\mu_2)\in\text{Dir}_\forall} \overline{R(\mu_1,\mu_2)}\circ\overline{\text{conf}} \;\cup \bigcup_{E(\mu)\in\text{Dir}_\exists}\overline{E(\mu)}\circ\overline{\text{conf}}
\end{aligned}$$

Finally, the formula φ is $\langle\text{input}\circ\text{traverse}^*\rangle\neg\langle\text{pop}_\Omega\rangle\texttt{true}$. It is tedious but straightforward to verify that, indeed, $w\in L(\mathcal{M})$ if and only if $\varepsilon\in[\![\varphi]\!]_{K(\mathcal{X})}$. ■

By introducing an additional atomic program loopp (for *loop p*rogram) with the rule

$$\varepsilon \xrightarrow{\text{loopp}}_{\mathcal{X}} \varepsilon$$

and replacing programs $\text{loop}(\pi)?$ by $\pi\cap\text{loopp}$, one can modify the above construction such that φ is a test-free IPDL formula. Hence, we obtain the following corollary.

Corollary 3.34 *There exists a test-free IPDL formula φ such that the following problem is* EXP*-hard:*
INPUT: A BPA $\mathcal{X}$.
QUESTION: $\varepsilon\in[\![\varphi]\!]_{K(\mathcal{X})}$? □

Expression Complexity in test-free IPDL

The proof is similar to the one given in the previous section. In particular, in the relevant worlds of the BPA, the stack consists of a sequence of configurations separated by direction markers. Again, we ensure the existence of an acceptance tree by means of a depth-first and left-to-right traversal using a program traverse.

The main differences to the previous proof are as follows. First, we aim at proving 2EXP-hardness, and thus reduce the membership problem of *exponentially* space bounded alternating Turing machines. This makes it necessary to encode a configuration $c=a_0a_1\cdots a_{i-1}qa_{i+1}\cdots a_{2^N-1}$ in a different way, namely as the word

$$a_0(0)a_1(1)\cdots a_{i-1}(i-1)q(i)a_{i+1}(i+1)\cdots a_{2^N-1}(2^N-1), \tag{3.7}$$

where $(k)\in\{0,1\}^N$ denotes the binary representation of $k\in[0,2^N-1]$ in N bits, with the least significant bit left-most. During the proof, we will refer to a subword of the form $a(i)$ as a *cell*. Second, we work with simpler BPAs that provide less structure than in the previous proof. This is compensated by making full use of the intersection operator, combining ideas from [72] and [124]. And third, we move the description of the initial configuration from the BPA to the formula because we aim at expression complexity instead of data complexity.

Theorem 3.35 *There exists a fixed BPA $\mathcal{X}$ such that the following problem is* $2\mathsf{EXP}$*-hard:*
INPUT: A test-free IPDL formula φ.
QUESTION:$\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$? □

Proof Let $\mathcal{M} = (Q, \Gamma, \Sigma, q_0, \delta, \square)$ be a fixed $(2^{p(n)} - 1)$-space bounded ATM (where $p(n)$ is a polynomial) with a 2EXP-hard membership problem. Let $w = w_1 w_2 \cdots w_n \in \Sigma^*$ be an input of length n. We give a basic process algebra $\mathcal{X} = \mathcal{X}(\mathcal{M})$ and a logarithmic space computable test-free IPDL formula $\varphi = \varphi(\mathcal{M}, w)$ such that $w \in L(\mathcal{M})$ if and only if $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$. Let $N = p(n)$ and $\Omega = Q \uplus \Gamma$. The stack alphabet of the BPA $\mathcal{X}$ is $\Gamma_{\mathcal{X}} = \Omega \uplus \{0,1\} \uplus \mathrm{Dir}$ and the atomic programs are $\mathsf{A} = \Gamma_{\mathcal{X}} \cup \overline{\Gamma}_{\mathcal{X}} \cup \{\mathrm{loopp}\}$, where $\overline{\Gamma}_{\mathcal{X}} = \{\overline{a} \mid a \in \Gamma_{\mathcal{X}}\}$ is a disjoint copy of $\Gamma_{\mathcal{X}}$. We define $\mathcal{X}$ such that $K(\mathcal{X})$ is the complete $\Gamma_{\mathcal{X}}$-tree with backward edges (the elements of $\Gamma_{\mathcal{X}}$ travel forward and those of $\overline{\Gamma}_{\mathcal{X}}$ backwards) as depicted in Figure 2.3, and where additionally each world has a reflexive transition for the program loopp. Formally, the rewrite rules of $\mathcal{X}$ are as follows,

$$
\begin{array}{lcll}
\varepsilon & \xrightarrow{a}_{\mathcal{X}} & a & \text{for each } a \in \Gamma_{\mathcal{X}}, \\
a & \xrightarrow{\overline{a}}_{\mathcal{X}} & \varepsilon & \text{for each } a \in \Gamma_{\mathcal{X}}, \text{ and} \\
\varepsilon & \xrightarrow{\mathrm{loopp}}_{\mathcal{X}} & \varepsilon. &
\end{array}
$$

Next, we give a number of auxiliary programs, to be used in φ.

- The following programs pop symbols from the stack:
  - for $X \subseteq \Gamma_{\mathcal{X}}$, $\overline{X} = \bigcup_{x \in X} \overline{x}$ pops a single symbol from X;
  - for $i < N$, $\mathrm{pop}_i = \overline{\{0,1\}}^i$ pops i bits;
  - $\overline{\mathrm{cell}} = \overline{\Omega} \circ \mathrm{pop}_N$ pops a cell $a(i)$;
  - $\overline{\mathrm{cell}_0} = \overline{\Omega} \circ \overline{0}^N$ pops a cell $a(0)$;
  - $\overline{\mathrm{cell}_1} = \overline{\Omega} \circ \overline{1}^N$ pops a cell $a(2^N - 1)$.
- For $X \subseteq \Gamma_{\mathcal{X}}$, $X = \bigcup_{x \in X} x$ pushes a single symbol from X onto the stack.
- The program $\overline{\mathrm{inc}}$ is executable only if the top of the stack has the form $a(i)a'(i+1)$, and it pops $a(i)$ during its execution. In order to define $\overline{\mathrm{inc}}$, we use additional programs $\chi_{j,\beta}$, for $j \leq N-1$ and $\beta \in \{0,1\}$. The program $\chi_{j,\beta}$ pops j bits from the stack, then pops the bit β, and then pops the rest of the cell and the whole subsequent cell, ensuring that the $j+1$-st bit of the latter is also β:

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

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

$$\overline{\mathrm{inc}} = \left[\left((\overline{\mathrm{cell}} \circ \overline{\mathrm{cell}}) \cap \overline{\Omega} \circ \bigcup_{i=0}^{N-1} \left(\overline{1}^i \circ \overline{0} \circ \overline{\{0,1\}}^* \circ \overline{\mathrm{cell}} \cap \overline{\{0,1\}}^* \circ \overline{\Omega} \circ \overline{0}^i \circ \overline{1} \circ \overline{\{0,1\}}^* \cap \bigcap_{j=i+1}^{N-1} (\chi_{j,0} \cup \chi_{j,1}) \right)\right) \circ \Gamma_{\mathcal{X}}^* \right] \cap \overline{\mathrm{cell}}$$

Note that the final composition with $\Gamma_{\mathcal{X}}^*$ and the final intersection with $\overline{\mathrm{cell}}$ are needed to ensure that $\overline{\mathrm{inc}}$ pops only one cell, but not two. The union ranges over the possible positions for the leftmost 0-bit in the first cell.

- The program $\overline{\mathrm{conf}}$ pops a configuration from the stack, ensuring that it is of the shape (3.7):

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

- For $a, a' \in \Omega$, we give a program $\mathrm{test}_{a,a'}$ that will be executed only when the top of the stack consists of a suffix of a configuration followed by a direction marker $d \in \mathrm{Dir}$ and a complete configuration, i.e., if it is of the form

$$a_k(k)a_{k+1}(k+1)\cdots a_{2^N-1}(2^N-1)\ d\ a'_0(0)a'_1(1)\cdots a'_{2^N-1}(2^N-1).$$

The program $\mathrm{test}_{a,a'}$ verifies that $a_k = a$ and $a'_k = a'$, and pops $a_k(k)$. To formulate it, we use subprograms $\sigma_{j,\beta}$, which pops j bits from the stack, then pops the bit β, and then pops the rest of the cell:

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

We now give $\mathrm{test}_{a,a'}$:

$$\mathrm{test}_{a,a'} = \left(\left(\bigcap_{i=0}^{N-1} \bigcup_{\beta\in\{0,1\}} \overline{a} \circ \sigma_{i,\beta} \circ \overline{\mathrm{cell}}^* \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{cell}}^* \circ \overline{a'} \circ \sigma_{i,\beta}\right) \circ \Gamma_{\mathcal{X}}^*\right) \cap \overline{\mathrm{cell}}$$

- The program $\mathrm{test}_=$ is executed in the same situation as $\mathrm{test}_{a,a'}$. It checks whether the content of the top cell $a(k)$ is identical to the same cell in the subsequent configuration:

$$\mathrm{test}_= = \bigcup_{a\in\Omega} \mathrm{test}_{a,a}$$

- For $\mu \in \mathrm{Moves}$, the program check_μ is executable if the top of the stack is of the form $c'dc$, where c and c' are configurations, $d \in \mathrm{Dir}$, and c' is the successor configuration of c

corresponding to the move μ. Execution of check_μ leaves the stack unchanged. We show only the case where $\mu = (p, b, \leftarrow)$:

$$\begin{aligned}
\mathrm{check}_\mu \quad = \quad & \Bigg[\Bigg(\overline{\mathrm{conf}} \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \ \cap \\
& \mathrm{test}^*_= \circ \bigcup_{q \in Q, a, c \in \Gamma} (\mathrm{test}_{p,c} \circ \mathrm{test}_{c,q} \circ \mathrm{test}_{b,a}) \circ \mathrm{test}^*_= \circ \overline{\mathrm{Dir}} \circ \overline{\mathrm{conf}} \Bigg) \circ \Gamma^*_{\mathcal{X}}\Bigg] \\
& \cap \mathrm{loopp}
\end{aligned}$$

- For $q \in Q$ and $a \in \Gamma$, the program $\mathrm{scan}_{q,a}$ checks if the top configuration is in current state q and scans symbol a. Its execution leaves the stack unchanged:

$$\mathrm{scan}_{q,a} \quad = \quad \overline{\mathrm{cell}}^* \circ \overline{q} \circ \mathrm{pop}_N \circ \overline{a} \circ \Gamma^*_{\mathcal{X}} \ \cap \ \mathrm{loopp}$$

- $\mathrm{cells} = (\{0,1\}^N \circ \Omega)^*$ pushes finitely many cells on the stack.

We now define the program traverse:

$$\begin{aligned}
\mathrm{traverse} \quad = \quad & \bigcup_{\substack{q \in Q_\forall, a \in \Gamma \\ \delta(q,a) = (\mu_1, \mu_2)}} \mathrm{scan}_{q,a} \circ L(\mu_1, \mu_2) \circ \mathrm{cells} \circ \mathrm{check}_{\mu_1} \\
\cup & \bigcup_{\substack{q \in Q_\exists, a \in \Gamma \\ \delta(q,a) = (\mu_1, \mu_2), \mu \in \{\mu_1, \mu_2\}}} \mathrm{scan}_{q,a} \circ E(\mu) \circ \mathrm{cells} \circ \mathrm{check}_\mu \\
\cup & \bigcup_{\substack{q \in Q_\forall, a \in \Gamma \\ \delta(q,a) = \perp}} \mathrm{scan}_{q,a} \circ \overline{\mathrm{conf}} \\
\cup & \bigcup_{(\mu_1, \mu_2) \in \delta(Q_\forall, \Gamma)} \overline{L(\mu_1, \mu_2)} \circ R(\mu_1, \mu_2) \circ \mathrm{cells} \circ \mathrm{check}_{\mu_2} \\
\cup & \bigcup_{R(\mu_1, \mu_2) \in \mathrm{Dir}_\forall} \overline{R(\mu_1, \mu_2)} \circ \overline{\mathrm{conf}} \ \cup \bigcup_{E(\mu) \in \mathrm{Dir}_\exists} \overline{E(\mu)} \circ \overline{\mathrm{conf}}
\end{aligned}$$

Finally, we need a program check_w which checks that the initial configuration is on top of the stack, leaving the stack unchanged:

$$\begin{aligned}
\mathrm{check}_w \quad = \quad & \Bigg(\Bigg(\overline{q_0} \circ \mathrm{pop}_N \circ (\overline{w_1} \circ \mathrm{pop}_N) \circ \cdots \circ (\overline{w_n} \circ \mathrm{pop}_N) \circ (\overline{\Box} \circ \mathrm{pop}_N)^* \\
& \cap \ \overline{\mathrm{conf}}\Bigg) \ \circ \ \Gamma^*_{\mathcal{X}}\Bigg) \cap \mathrm{loopp}
\end{aligned}$$

Then, we set $\varphi = \langle \mathrm{cells} \circ \mathrm{check}_w \circ \mathrm{traverse}^* \rangle \neg \langle \overline{\Gamma}_{\mathcal{X}} \rangle \mathtt{true}$. It is tedious but not difficult to verify that $w \in L(\mathcal{M})$ if and only if $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$. ■

3.7 High undecidability of IPDL with atomic program negation

It is well-known that adding full program negation to PDL results satisfiability to become undecidable (Theorem 2.15), whereas PDL with program negation restricted to atomic programs remains decidable and EXP-complete (Theorem 2.16). Therefore, it is a natural question if satisfiability remains decidable when we extend μ-ICPDL$_{\mathtt{Nom}}$ with atomic program negation. The purpose of this section is to answer this question negatively: Already in IPDL extended with atomic program negation, satisfiability is complete for Σ_1^1, the first level of the analytic hierarchy [96].

We use IPDL$^{(\neg)}$ to denote the extension of IPDL with negation of atomic programs. The latter is written as $\neg a$ ($a \in \mathbb{A}$). Recall that the semantics of the new constructor is defined as $[\![\neg a]\!]_K = (W \times W) \setminus [\![a]\!]_K$ for each Kripke structure K with world set W. Since fixed points do not occur in IPDL$^{(\neg)}$, Kripke structures will be triples in this section.

Our proof of the Σ_1^1 lower bound is by reduction of a tiling problem that asks for a recurring tiling of the first quadrant of the plane. As shown in [52], this problem is Σ_1^1-complete. A *tiling system* $\mathcal{S} = (T, H, V, t_r)$ consists of a finite set of *tile types* T, *horizontal and vertical matching relations* $H, V \subseteq T \times T$, and a *recurring tile* $t_r \in T$. A *solution* to $\mathcal{S}$ is a mapping $\tau : \mathbb{N} \times \mathbb{N} \to T$ such that:

- for all $(x, y) \in \mathbb{N} \times \mathbb{N}$, if $\tau(x, y) = t$ and $\tau(x + 1, y) = t'$, then $(t, t') \in H$,
- for all $(x, y) \in \mathbb{N} \times \mathbb{N}$, if $\tau(x, y) = t$ and $\tau(x, y + 1) = t'$, then $(t, t') \in V$,
- for all $y \in \mathbb{N}$ there exists $y' > y$ with $\tau(0, y') = t_r$

The last condition states that the recurring tile t_r has to occur infinitely often in the first column. The *tiling problem* is to decide, given a tiling system $\mathcal{S}$, whether $\mathcal{S}$ has a solution.

To reduce the tiling problem to satisfiability in IPDL$^{(\neg)}$, we translate a tiling system $\mathcal{S} = (T, H, V, t_r)$ into an IPDL$^{(\neg)}$ formula $\varphi_{\mathcal{S}}$ such that $\mathcal{S}$ has a solution if and only if $\varphi_{\mathcal{S}}$ is satisfiable. In the formula $\varphi_{\mathcal{S}}$, we use two atomic programs a_x and a_y for representing the grid $\mathbb{N} \times \mathbb{N}$ and we use the tile types T as atomic propositions. The formula $\varphi_{\mathcal{S}}$ is the conjunction consisting of the following five conjuncts:

(a) every element of a (connected) model of $\varphi_{\mathcal{S}}$ represents an element of $\mathbb{N} \times \mathbb{N}$ and is labelled with a unique tile type:

$$[(a_x \cup a_y)^*] \left(\bigvee_{t \in T} t \;\wedge \bigwedge_{t,t' \in T, t \neq t'} \neg(t \wedge t') \right)$$

(b) every element has an a_x-successor and an a_y-successor:

$$[(a_x \cup a_y)^*] \big(\langle a_x \rangle \mathtt{true} \wedge \langle a_y \rangle \mathtt{true} \big)$$

(c) the programs a_x and a_y are confluent:

$$[(a_x \cup a_y)^*]\ \neg\langle (a_y \circ a_x) \cap (a_x \circ \neg a_y)\rangle \texttt{true}$$

(d) the horizontal and vertical matching conditions are respected:

$$[(a_x \cup a_y)^*]\left(\bigwedge_{t\in T} t\ \rightarrow\ \left([a_x]\bigvee_{(t,t')\in H} t'\ \wedge\ [a_y]\bigvee_{(t,t')\in V} t'\right)\right)$$

(e) we start in column 0 and the recurring tile t_r occurs infinitely often:

$$\langle a_y^*\rangle t_r\ \wedge\ [a_y^*]\ (\ t_r \rightarrow \langle a_y^+\rangle t_r\ ).$$

Lemma 3.36 *The tiling system $\mathcal{S}$ has a solution if and only if $\varphi_{\mathcal{S}}$ is satisfiable.* □

PROOF Since the "only if" direction is simple, we only prove the "if" direction. Thus, let $\varphi_{\mathcal{S}}$ be satisfiable and let $K = (W, \{\rightarrow_{a_x}, \rightarrow_{a_y}\}, \{W_t \mid t \in T\})$ be a model of $\varphi_{\mathcal{S}}$. We have to construct a solution τ to $\mathcal{S}$. To prepare for this, we first define a mapping $\pi : \mathbb{N}\times\mathbb{N} \rightarrow W$, which is done in two steps. In the first step, we pick $x \in [\![\varphi_{\mathcal{S}}]\!]_K$ and set $\pi(0,0) = x$. Next, we define $\pi(0,k)$ for all $k > 0$ as follows: By (e) we can find a sequence of worlds $(x_i)_{i\geq 0}$ such that $x_0 = \pi(0,0)$ and $(x_i, x_{i+1}) \in [\![a_y]\!]_K$ for all $i \geq 0$ and such that $x_j \in W_{t_r}$ for infinitely many $j \geq 0$. Observe that the sequence $(x_i)_{i\geq 0}$ is not necessarily repetition-free. We set $\pi(0,i) = x_i$. Finally, we define $\pi(i,j)$ for all $i > 0$: Assume that $\pi(i,j)$ is already defined. By (b), we can choose $z \in W$ such that $(\pi(i,j), z) \in [\![a_x]\!]_K$ and set $\pi(i+1,j) = z$. By construction, we have $(\pi(i,j), \pi(i+1,j)) \in [\![a_x]\!]_K$ for all $i,j \geq 0$. Moreover, the confluence property (c) implies that also $(\pi(i,j), \pi(i,j+1)) \in [\![a_y]\!]_K$ for all $i,j \geq 0$.

The resulting mapping π gives rise to a mapping $\tau : \mathbb{N}\times\mathbb{N} \rightarrow T$ in the obvious way: by (a), we can define $\tau(i,j)$ as the unique $t \in T$ with $\pi(i,j) \in [\![t]\!]_K$. Finally, (d) and our choice of the values $\pi(0,i)$, $i \geq 0$, imply that τ is a solution to $\mathcal{S}$. ■

To show that satisfiability in IPDL$^{(\neg)}$ is in Σ^1_1 observe that IPDL$^{(\neg)}$ is a fragment of LFP and every satisfiable LFP formula has a countable model by [37, 48]: It is now straightforward to give an existential second-order formula of arithmetic over the naturals that expresses, for a given IPDL$^{(\neg)}$ formula, the existence of a countable model. We have thus established the following result.

Theorem 3.37 *Satisfiability in IPDL$^{(\neg)}$ is Σ^1_1-complete.* □

Since intersection of programs can be defined in terms of program union and (full) program negation, this also proves Σ^1_1-completeness of PDL with full program negation, improving upon the undecidability result given by Harel in [53], using a (trivial) reduction of the equivalence problem for the algebra of binary relations with complementation.

3.8 Open problems

Finite satisfiability

Note that Proposition 2.17 states that already IPDL does not have the finite model property. Hence the *finite satisfiability problem* in μ-ICPDL$_{\mathtt{Nom}}$ is a natural one to consider. It asks, given some μ-ICPDL$_{\mathtt{Nom}}$ formula φ, whether φ has a *finite* model.

Unsurprisingly, it follows directly from the 2EXP lower bound for general satisfiability in IPDL (Theorem 2.21) that also finite satisfiability in IPDL, and hence in μ-ICPDL$_{\mathtt{Nom}}$, is hard for 2EXP. Currently, the decidability status of the finite satisfiability problem already in IPDL is unknown to the author. A first approach to show its decidability could be to prove the existence of a computable function $f : \mathbb{N} \to \mathbb{N}$ such that each finitely satisfiable formula φ has a finite model of tree width at most $f(|\varphi|)$. In constrast to the unsuccessful search for such a computable function, the author is not aware of any μ-ICPDL$_{\mathtt{Nom}}$ formula φ that is finitely satisfiable, but each finite model of φ having tree width at least $2^{|\varphi|}$. Instead of trying to bound the tree width of finite models, one could as well try to bound their *clique width*, a graph-thereotic notion introduced in [21]. Roughly speaking, clique width is a more general notion than tree width in the sense that small tree width of a Kripke structure implies small clique width but not vice versa. Similarly as we encoded countable models of bounded tree width in Section 3.3, it is possible to encode countable models of bounded clique width by some ω-regular tree language.

In [11] Bojańczyk extends monadic second order logic by a so called *bounding quantifier* $\mathbb{B}$, yielding MSOL+$\mathbb{B}$. Formulas $\mathbb{B}X.\varphi(X)$ hold whenever there exists a finite bound on the sets satisfying $\varphi(X)$. For two fragments of MSOL+$\mathbb{B}$ it is shown in [11] that the corresponding theory over the complete binary tree is decidable. One of the applications of these decidability results is the decidability of the finite satisfiability problem of the modal μ-calculus with backwards modalities – the precise complexity of the latter problem is established in [10]. If one could define in one of the decidable fragments of MSOL+$\mathbb{B}$ the set $\{K^{\#} \mid K$ is a finite Kripke structure$\}$, where $\#$ is the operator on pointed Kripke structure from Section 3.2, then there is hope for the finite satisfiability problem in μ-ICPDL$_{\mathtt{Nom}}$ to be decidable.

Precise complexity of Data Analysis Logic

One major motivation to consider already ICPDL is that Data Analysis Logic (DAL) of Farinas et al. [33] is a polytime translatable fragment of it and hence upper bounds for satisfiability in ICPDL carry over to DAL. The Kripke structures with world set W that are considered in DAL are restricted to be ones where each transition relation $\to_a$ is an equivalence relation on W. The syntax of DAL formulas φ and programs π is given by the following grammar, where p ranges over $\mathbb{P}$ and a ranges over $\mathbb{A}$:

$$\begin{array}{lcl} \varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi \\ \pi & ::= & a \mid \pi \sqcup \pi \mid \pi \sqcap \pi \end{array}$$

The semantics of DAL should be clear, except that for DAL programs π_1 and π_2 the binary relation $[\![\pi_1 \sqcup \pi_2]\!]_K$ is inductively defined as the smallest equivalence relation that contains both $[\![\pi_1]\!]_K$ and $[\![\pi_2]\!]_K$, where $[\![\pi_1 \sqcap \pi_2]\!]_K$ is defined to be $[\![\pi_1]\!]_K \cap [\![\pi_2]\!]_K$, which is again an equivalence relation. It is not hard to see that a DAL formula φ is satisfiable if and only if the ICPDL formula φ' is satisfiable, where φ' is obtained from φ by replacing each atomic program a that occurs in φ by $(a \cup \overline{a})^*$ and each occurring program of the kind $\pi_1 \sqcup \pi_2$ by $(\pi_1 \cup \pi_2)^*$.

Moreover, one can show that satisfiability in DAL is EXP-hard [79], where the best upper bound known to the author is 2EXP inherited from ICPDL. The model theory of DAL is also not too easy: One can construct a satisfiable DAL formula φ such that each model of φ is not the reflexive transitive closure of any undirected tree [79].

Expressiveness in μ-ICPDL$_{\mathtt{Nom}}$

Note that we called a world x a *recovery world* of a Kripke structure $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, \rho)$ if for each $y \in W$ that is reachable from x, i.e. $(x, y) \in (\cup_{a \in \mathbb{A}} \to_a)^*$, we have that x is also reachable from y. In Example 2.23 we showed that the recovery world property is already expressible in IPDL's fragment loop-PDL as long as we restrict ourselves to *deterministic* Kripke structures. The natural question arises, when not restricting ourselves to deterministic Kripke structures, whether we can express the recovery world property in μ-ICPDL$_{\mathtt{Nom}}$. The following proposition answers the latter question negatively.

Proposition 3.38 *The recovery world property is not expressible in μ-ICPDL$_{\mathtt{Nom}}$. I.e. there does not exist a μ-ICPDL$_{\mathtt{Nom}}$ formula φ such that for every Kripke structure K over the nominals $\mathtt{Nom}(\varphi)$ and every world x of K we have $x \in [\![\varphi]\!]_K$ if and only if x is a recovery world of K.* $\square$

PROOF Assume by contradiction that there exists a μ-ICPDL$_{\mathtt{Nom}}$ formula φ such that for each Kripke structure K over $\mathtt{Nom}(\varphi)$ we have $[\![\varphi]\!]_K = \{x \mid x \text{ is a recovery world of } K\}$. Let $p_1, \ldots, p_k$ be an enumeration of $\mathtt{Nom}(\varphi)$. Consider the following two Kripke structures K_1 and K_2 over $\mathtt{Nom}(\varphi)$,

where atomic propositions hold in none of the worlds of K_1 and K_2. Moreover, each nominal p_i is represented by the world x_i (resp. y_i) in K_1 (resp. K_2). Obviously x_{k+1} is a recovery world of K_1, whereas y_{k+1} is not a recovery world of K_2. Since over K_1 and K_2 the formula φ is equivalent to some ICPDL$^{\infty}_{\mathtt{Nom}}$ formula by Lemma 3.4, the proposition follows from Point (i) of the following claim, since it implies that in particular x_{k+1} and y_{k+1} satisfy the same ICPDL$^{\infty}_{\mathtt{Nom}}$ formulas over $\mathtt{Nom}(\varphi)$.

Claim. Let Φ denote the set of $\mathbb{X}$-free ICPDL$^{\infty}_{\mathtt{Nom}}$ formulas over $\mathtt{Nom}(\varphi)$ and let Π denote the set of $\mathbb{X}$-free ICPDL$^{\infty}_{\mathtt{Nom}}$ programs over $\mathtt{Nom}(\varphi)$ such that for each $\psi \in \Phi$ and for each $\pi \in \Pi$ the following four conditions are satisfied,

(i) $x_i \in [\![\psi]\!]_{K_1} \Leftrightarrow y_i \in [\![\psi]\!]_{K_2}$ for each $i \in [k+1]$,

(ii) $y_{k+1} \in [\![\psi]\!]_{K_2} \Leftrightarrow y_{k+2} \in [\![\psi]\!]_{K_2}$,

(iii) $(x_i, x_i) \in [\![\pi]\!]_{K_1} \Leftrightarrow (y_i, y_i) \in [\![\pi]\!]_{K_2}$ for each $i \in [k+1]$, and

(iv) $(y_{k+i}, y_{k+j}) \in [\![\pi]\!]_{K_2}$ implies both $(y_{k+1}, y_{k+1}) \in [\![\pi]\!]_{K_2}$ and $(y_{k+2}, y_{k+2}) \in [\![\pi]\!]_{K_2}$ for each $i, j \in \{1, 2\}$.

Then $\mathtt{Nom}(\varphi) \subseteq \Phi$ and moreover Φ and Π are closed under Points (3) to (11) of Definition 3.3. □

Let us first prove that $\mathtt{Nom}(\varphi) \subseteq \Phi$. For Point (i) observe that for each $i \in [k]$ we have $x_i \in [\![p_i]\!]_{K_1}$ if and only if $y_i \in [\![p_i]\!]_{K_2}$. For Point (ii) observe that neither y_{k+1} nor y_{k+2} satisfies any nominal from $\mathtt{Nom}(\varphi)$.

Before proving that Φ and Π are closed under Points (3) to (11) of Definition 3.3, observe that it is easily verifiable that for each ICPDL$^{\infty}_{\mathtt{Nom}}$ program π we have

$$\begin{aligned} [\![\pi]\!]_{K_1} &\subseteq \{(x_i, x_i) \mid i \in [k+1]\} && (3.8)\\ [\![\pi]\!]_{K_2} &\subseteq \{(y_i, y_i) \mid i \in [k]\} \cup \{(y_{k+i}, y_{k+j}) \mid i, j \in \{1, 2\}\} && (3.9)\end{aligned}$$

We now prove that Φ and Π are closed under Points (3) to (11) of Definition 3.3.

(3) $\mathbb{P} \subseteq \Phi$ follows from $[\![p]\!]_{K_1} = [\![p]\!]_{K_2} = \emptyset$ for each $p \in \mathbb{P}$.

(4) If $\varphi \in \Phi$, then Points (i) and (ii) hold for $\neg\varphi$ immediately.

(5) If $S \subseteq \Phi$, then Points (i) and (ii) hold for $\bigvee S$ immediately.

(6) Assume $\pi \in \Pi$ and $\psi \in \Phi$. We prove that Points (i) and (ii) hold for $\langle\pi\rangle\psi$.
To show Point (i) let us assume first $i \in [k]$. Hence we have $(x_i, x_i) \in [\![\pi]\!]_{K_1}$ if and only if $(y_i, y_i) \in [\![\pi]\!]_{K_2}$ by Point (iii) and $x_i \in [\![\psi]\!]_{K_1}$ if and only if $y_i \in [\![\psi]\!]_{K_2}$ by Point (i). Hence

$$\begin{aligned} x_i \in [\![\langle\pi\rangle\psi]\!]_{K_1} &\overset{(3.8)}{\iff} (x_i, x_i) \in [\![\pi]\!]_{K_1} \wedge x_i \in [\![\psi]\!]_{K_1}\\ &\iff (y_i, y_i) \in [\![\pi]\!]_{K_2} \wedge y_i \in [\![\psi]\!]_{K_2}\\ &\overset{(3.9)}{\iff} y_i \in [\![\langle\pi\rangle\psi]\!]_{K_2}.\end{aligned}$$

For Point (i) it remains to prove the case $i = k+1$. On the one hand, assume $x_{k+1} \in [\![\langle\pi\rangle\psi]\!]_{K_1}$. Then by (3.8) we have $(x_{k+1}, x_{k+1}) \in [\![\pi]\!]_{K_1}$ and $x_{k+1} \in [\![\psi]\!]_{K_1}$. Hence $y_{k+1} \in [\![\psi]\!]_{K_2}$ by Point (i) and $(y_{k+1}, y_{k+1}) \in [\![\pi]\!]_{K_2}$ by Point (iii). Thus $y_{k+1} \in [\![\langle\pi\rangle\psi]\!]_{K_2}$. On the other hand assume $y_{k+1} \in [\![\langle\pi\rangle\psi]\!]_{K_2}$. Then by (3.9) there exists some $j\{1, 2\}$ with $(y_{k+1}, y_{k+j}) \in [\![\pi]\!]_{K_2}$ and $y_{k+j} \in [\![\psi]\!]_{K_2}$. This implies $(y_{k+1}, y_{k+1}) \in [\![\pi]\!]_{K_2}$ by Point (iv) and hence $(x_{k+1}, x_{k+1}) \in [\![\pi]\!]_{K_1}$ by Point (iii). We obtain $x_{k+1} \in [\![\psi]\!]_{K_1}$ since $y_j \in [\![\psi]\!]_{K_2}$ if and only if $x_{k+1} \in [\![\psi]\!]_{K_1}$ by combining Points (ii) and (i). Hence $x_{k+1} \in [\![\langle\pi\rangle\psi]\!]_{K_1}$.

For Point (ii) observe that Point (ii) holds for ψ and Points (iii) and (iv) hold for π; hence we obtain the following equivalences

$$\begin{aligned} y_{k+1} \in [\![\langle\pi\rangle\psi]\!]_{K_2} &\overset{(3.9)}{\iff} \exists j \in \{1, 2\} : (y_{k+1}, y_{k+j}) \in [\![\pi]\!]_{K_2} \wedge y_{k+j} \in [\![\psi]\!]_{K_2}\\ &\overset{\text{(ii),(iv)}}{\iff} (y_{k+1}, y_{k+1}) \in [\![\pi]\!]_{K_2} \wedge (y_{k+2}, y_{k+2}) \in [\![\pi]\!]_{K_2} \wedge\\ &\qquad y_{k+1} \in [\![\psi]\!]_{K_2} \wedge y_{k+2} \in [\![\psi]\!]_{K_2}\\ &\overset{\text{(ii),(iii)}}{\iff} y_{k+2} \in [\![\langle\pi\rangle\psi]\!]_{K_2}.\end{aligned}$$

(7) Let $i \in [k]$ and $\psi \in \Phi$. Then Points (i) and (ii) for $@_{p_i}\psi$ follow easily from $[\![p_i]\!]_{K_1} = \{x_i\}$, $[\![p_i]\!]_{K_2} = \{y_i\}$ and from Points (i) and (ii) for ψ.

(8) Let $\pi = a \in \mathbb{A} \cup \{\overline{a} \mid a \in \mathbb{A}\}$. Then Points (iii) and (iv) hold directly by definition of K_1 and K_2.

(9) Let $\psi \in \Phi$. It is clear that Points (iii) and (iv) hold for $\varphi?$ due to Point (i) and (ii) for ψ.

(10) Let $\pi_1, \pi_2 \in \Pi$.
To prove $\pi_1 \cap \pi_2 \in \Phi$, let us first prove Point (iii). For this, let $i \in [k+1]$. Then, we have

$$\begin{aligned} (x_i, x_i) \in [\![\pi_1 \cap \pi_2]\!]_{K_1} &\Leftrightarrow (x_i, x_i) \in [\![\pi_1]\!]_{K_1} \wedge (x_i, x_i) \in [\![\pi_2]\!]_{K_1} \\ &\Leftrightarrow (y_i, y_i) \in [\![\pi_1]\!]_{K_2} \wedge (y_i, y_i) \in [\![\pi_2]\!]_{K_2} \\ &\Leftrightarrow (y_i, y_i) \in [\![\pi_1 \cap \pi_2]\!]_{K_2}. \end{aligned}$$

For Point (iv) let $i, j \in \{1, 2\}$ and assume $(y_{k+i}, y_{k+j}) \in [\![\pi_1 \cap \pi_2]\!]_{K_2}$. Then we have $(y_{k+i}, y_{k+j}) \in [\![\pi_1]\!]_{K_1} \cap [\![\pi_2]\!]_{K_1}$. Since $\pi_1, \pi_2 \in \Pi$ we have $(y_{k+1}, y_{k+1}) \in [\![\pi_1]\!]_{K_2}$, $(y_{k+2}, y_{k+2}) \in [\![\pi_1]\!]_{K_2}$, $(y_{k+1}, y_{k+1}) \in [\![\pi_2]\!]_{K_2}$, and $(y_{k+2}, y_{k+2}) \in [\![\pi_2]\!]_{K_2}$. Hence $(y_{k+1}, y_{k+1}) \in [\![\pi_1 \cap \pi_2]\!]_{K_2}$ and $(y_{k+2}, y_{k+2}) \in [\![\pi_1 \cap \pi_2]\!]_{K_2}$.

The proof of $\pi_1 \cup \pi_2 \in \Pi$ can be proven in analogy to $\pi_1 \cap \pi_2 \in \Pi$.

Let us prove $\pi_1 \circ \pi_2 \in \Pi$. For Point (iii) let $i \in [k+1]$. The equality $(x_i, x_i) \in [\![\pi_1 \circ \pi_2]\!]_{K_1} \Leftrightarrow (y_i, y_i) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$ for each $i \in [k]$ follows immediately from (3.8), (3.9), and $\pi_1, \pi_2 \in \Pi$. Let us prove the equality for $i = k+1$. On the one hand, assume $(x_{k+1}, x_{k+1}) \in [\![\pi_1 \circ \pi_2]\!]_{K_1}$. Then by (3.8) we have $(x_{k+1}, x_{k+1}) \in [\![\pi_1]\!]_{K_1}$ and $(x_{k+1}, x_{k+1}) \in [\![\pi_2]\!]_{K_1}$. Since $\pi_1, \pi_2 \in \Pi$ we have $(y_{k+1}, y_{k+1}) \in [\![\pi_1]\!]_{K_2}$ and $(y_{k+1}, y_{k+1}) \in [\![\pi_2]\!]_{K_2}$ by Point (iii). Hence $(y_{k+1}, y_{k+1}) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$. On the other hand, assume $(y_{k+1}, y_{k+1}) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$. Then by (3.9) there exists some $j \in \{1, 2\}$ such that $(y_{k+1}, y_{k+j}) \in [\![\pi_1]\!]_{K_2}$ and $(y_{k+j}, y_{k+1}) \in [\![\pi_2]\!]_{K_2}$. Since $\pi_1, \pi_2 \in \Pi$ Point (iii) implies $(y_{k+1}, y_{k+1}) \in [\![\pi_1]\!]_{K_2}$ and $(y_{k+2}, y_{k+2}) \in [\![\pi_1]\!]_{K_2}$. By Point (iii) it follows $(x_{k+1}, x_{k+1}) \in [\![\pi_1]\!]_{K_1}$ and $(x_{k+1}, x_{k+1}) \in [\![\pi_2]\!]_{K_1}$. Thus $(x_{k+1}, x_{k+1}) \in [\![\pi_1 \circ \pi_2]\!]_{K_1}$.
For Point (iv), assume $(y_{k+i}, y_{k+j}) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$ for some $i, j \in \{1, 2\}$. Then $(y_{k+i}, y_{k+l}) \in [\![\pi_1]\!]_{K_2}$ and $(y_{k+l}, y_{k+j}) \in [\![\pi_2]\!]_{K_2}$ for some $l \in \{1, 2\}$ by (3.9). Since $\pi_1, \pi_2 \in \Pi$ we have $(y_{k+1}, y_{k+1}), (y_{k+2}, y_{k+2}) \in [\![\pi_1]\!]_{K_2}$ and $(y_{k+1}, y_{k+1}), (y_{k+2}, y_{k+2}) \in [\![\pi_2]\!]_{K_2}$ by Point (iv). Hence $(y_{k+1}, y_{k+1}) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$ and $(y_{k+2}, y_{k+2}) \in [\![\pi_1 \circ \pi_2]\!]_{K_2}$.

(11) Let $\pi \in \Pi$. Points (iii) and (iv) hold trivially for π^*. ■

Since we proved the inexpressiveness of the recovery world property in the previous proposition "from scratch", we discuss possibilities of obtaining inexpressiveness results for μ-ICPDL$_{\texttt{Nom}}$ in a more elegant way.

Note that we introduced the operator $\#$ on Kripke structures from Section 3.2 in order to show that each satisfiable μ-ICPDL$_{\texttt{Nom}}$ formula that uses k different nominals has a countable model of tree width at most $k+2$ (Corollary 3.6). It would be nice to prove inexpressiveness results in μ-ICPDL$_{\texttt{Nom}}$ by the usage of the operator $\#$. For example, to show e.g. that the recovery world property is inexpressible in μ-ICPDL$_{\mathtt{Nom}}$ it is sufficient to find some Kripke structure K and some world x of K such that x is a recovery world of K if and only if $x^{\#}$ is *not* a recovery world in $K^{\#}$. But unfortunately, the following claim states that our operator $\#$ preserves the recovery world property.

Claim 3.39 (without proof) *For every Kripke structure K and every world x of K we have that x is a recovery world in K if and only if $x^{\#}$ is a recovery world in $K^{\#}$.* □

So the question is if one can restrict the operator $\#$ to gain inexpressiveness results in the way described above? Another elegant way to obtain inexpressiveness results in μ-ICPDL$_{\mathtt{Nom}}$ would be to introduce some appropriate notion of bisimilarity. So whenever one can (easily) show that two Kripke structures are μ-ICPDL$_{\mathtt{Nom}}$-bisimilar this would imply that they can not be distinguished by any μ-ICPDL$_{\mathtt{Nom}}$ formula. For example, it is well known that two (pointed) Kripke structures are bisimilar (in the classical sense) if and only if they are indistinguishable in infinitary modal logic [114]. So can we find a notion of bisimilarity that characterizes our infinitary logic ICPDL$^{\infty}_{\mathtt{Nom}}$ from Section 3.1?

4

Dynamic Logics of Permission

The main result of this chapter is that satisfiability in PDL's extension $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ is in EXP and thus EXP-complete by the EXP-hardness that $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ inherits from PDL. The previously best known upper bound for satisfiability in $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ was $2\mathsf{EXP}$. It has been proven by Demri via a natural translation to PDL [24]. From Demri's translation it follows that satisfiability of formulas, where nestings of $\mathbf{grant}$ and $\mathbf{revoke}$ operators are bounded by some constant, lies in EXP. For $\mathrm{DLP}^{+}_{\mathrm{dyn}}$'s fragment $\mathrm{DLP}_{\mathrm{dyn}}$, it is pointed out but not fully proved in [95] that satisfiability is in NEXP.

An approach proposed in [24] to improve the complexity status of full $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ is a polynomial time computable reduction to an extension of PDL that allows an additional operator $\oplus$ (we call the resulting logic PDL$\oplus$ from now on) and then proving that satisfiability of PDL$\oplus$ lies in EXP. A program $\oplus(\pi, \varphi_1, \varphi_2)$, where π is a program and both φ_1 and φ_2 are formulas, relates pairs of worlds (x, y) of a Kripke structure, that are related via π such that additionally φ_1 holds in x or φ_2 holds in y. As we will remark later, PDL$\oplus$ does not add expressiveness to PDL but succinctness, i.e. the size of the resulting PDL formula may grow exponentially in the size of original PDL$\oplus$ formula. Alas, it turns out that a translation from $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ to PDL$\oplus$ will not lead to an improvement of the complexity of $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ – we prove in this chapter that satisfiability in PDL$\oplus$ is $2\mathsf{EXP}$-complete. Yet to obtain an EXP upper bound for satisfiability in $\mathrm{DLP}^{+}_{\mathrm{dyn}}$, we accurately identify a syntactic fragment PDL$\oplus[\mathbb{A}]$ of PDL$\oplus$, into which we translate $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ and that we prove to lie in EXP. Our translation from $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ to PDL$\oplus[\mathbb{A}]$ consists of a concise examination of how applied $\mathbf{grant}$ and $\mathbf{revoke}$ operators influence the truth of subformulas. For proving that PDL$\oplus[\mathbb{A}]$ lies in EXP, we translate an input formula of PDL$\oplus[\mathbb{A}]$ φ into a two-way alternating parity tree automaton over infinite trees $\mathcal{T}(\varphi)$ and check $\mathcal{T}(\varphi)$ for non-emptiness.

Section 4.1 introduces $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ and its fragments $\mathrm{DLP}_{\mathrm{dyn}}$ and DLP. Known complexity results for comparable logics and difficulties of reasoning in $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ are discussed in Section 4.2. In

Section 4.3, we give a satisfiability preserving translation from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to $\mathrm{PDL}\oplus[\mathbb{A}]$. An EXP upper bound for satisfiability in $\mathrm{PDL}\oplus[\mathbb{A}]$ is proven in Section 4.4. In Section 4.5, we provide a 2EXP lower bound for satisfiability in $\mathrm{PDL}\oplus$. Section 4.6 discusses a variant of $\mathrm{DLP}^+_{\mathrm{dyn}}$ which satisfiability is 2EXP-hard in. We conclude this chapter with open problems in Section 4.7.

4.1 $\mathrm{DLP}^+_{\mathrm{dyn}}$ and its fragments $\mathrm{DLP}_{\mathrm{dyn}}$ and DLP

Let $w = a_1a_2\cdots a_n$ be a string over some alphabet Σ and $a_i \in \Sigma$ for each $i \in [n]$. Then $|w| = n$ denotes the *length* of w. Moreover, $w^{(j)} = a_1\cdots a_j$ denotes the *prefix of* w *of length* j for each $j \in [n]$ and by convention $w^{(0)} = \varepsilon$. Recall that $\mathbb{P}$ denotes some countable set of *atomic propositions* and $\mathbb{A}$ some countable set of *atomic programs*.

The set of *formulas* Φ and the set of *programs* Π of the logic $\mathrm{DLP}^+_{\mathrm{dyn}}$ are the smallest sets that satisfy the following nine conditions,

(i) $\mathbb{P} \subseteq \Phi$,

(ii) if $\varphi \in \Phi$, then $\neg\varphi \in \Phi$,

(iii) if $\varphi_1, \varphi_2 \in \Phi$, then $\varphi_1 \vee \varphi_2 \in \Phi$,

(iv) if $\pi \in \Pi$ and $\varphi \in \Phi$, then $\langle\pi\rangle\varphi,\ \mathbf{perm}(\pi)\varphi,\ \mathbf{fperm}(\pi)\varphi \in \Phi$,

(v) if $\varphi_1, \varphi_2, \varphi_3 \in \Phi$, then $\mathbf{grant}(\varphi_1, \varphi_2)\varphi_3,\ \mathbf{revoke}(\varphi_1, \varphi_2)\varphi_3 \in \Phi$,

(vi) if $\varphi \in \Phi$, then $\varphi? \in \Pi$,

(vii) $\mathbb{A} \subseteq \Pi$,

(viii) if $\pi_1, \pi_2 \in \Pi$, then $\pi_1 \cup \pi_2,\ \pi_1 \circ \pi_2 \in \Pi$, and

(ix) if $\pi \in \Pi$, then $\pi^* \in \Pi$.

We denote by $\mathrm{Tests} = \{\varphi? \mid \varphi \in \Phi\}$ the set of *test programs*. Moreover, we introduce the usual abbreviations: $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\texttt{true} = p \vee \neg p$ for some $p \in \mathbb{P}$, $\texttt{false} = \neg\texttt{true}$, $\varphi_1 \to \varphi_2 = \neg\varphi_1 \vee \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$, and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$.
The logic $\mathrm{DLP}_{\mathrm{dyn}}$ is the fragment of $\mathrm{DLP}^+_{\mathrm{dyn}}$, where (vi) is dropped and where (v) is replaced by

(v)′ if β_1 and β_2 are boolean combinations of atomic propositions and $\varphi \in \Phi$, then $\mathbf{grant}(\beta_1, \beta_2)\varphi,\ \mathbf{revoke}(\beta_1, \beta_2)\varphi \in \Phi$.

The logic DLP is the fragment of $\mathrm{DLP}_{\mathrm{dyn}}$, where the operators $\mathbf{grant}$ and $\mathbf{revoke}$ do not occur.
A *Kripke structure* is a tuple $(W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\})$, where W is a set of *worlds*, $\to_a \subseteq W \times W$ is a binary relation for each $a \in \mathbb{A}$, and $W_p \subseteq W$ for each $p \in \mathbb{P}$. An *extended Kripke structure* is a tuple $(W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$, where $(W, \{\to_a \mid a \in$

$\mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\})$ is a Kripke structure and $P \subseteq W \times W$ is a binary relation that we call *policy set*. Let $\mathrm{op} \in \{\cup, \setminus\}$ be a binary operation, let $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ be an extended Kripke structure, and let $A, B \subseteq W$. Then, let us define the extended Kripke structure $K \restriction (A, B, \mathrm{op}) = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P \;\mathrm{op}\; (A \times B))$.

Let $K = (W, \{\to_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ be an extended Kripke structure. Then, for each atomic/test program π, we define a binary relation $[\![\pi]\!]_K \subseteq W \times W$ and for each formula $\varphi \in \Phi$ we define a subset $[\![\varphi]\!]_K \subseteq W$ inductively as follows,

$$
\begin{aligned}
[\![a]\!]_K &= \to_a \quad \text{for each } a \in \mathbb{A}, \\
[\![\varphi?]\!]_K &= \{(x, x) \mid x \in [\![\varphi]\!]_K\}, \\
[\![p]\!]_K &= W_p \quad \text{for each } p \in \mathbb{P}, \\
[\![\neg\varphi]\!]_K &= W \setminus [\![\varphi]\!]_K, \\
[\![\varphi_1 \vee \varphi_2]\!]_K &= [\![\varphi_1]\!]_K \cup [\![\varphi_2]\!]_K, \\
[\![\mathbf{grant}(\varphi_1, \varphi_2)\varphi_3]\!]_K &= [\![\varphi_3]\!]_{K \restriction ([\![\varphi_1]\!]_K, [\![\varphi_2]\!]_K, \cup)}, \text{ and} \\
[\![\mathbf{revoke}(\varphi_1, \varphi_2)\varphi_3]\!]_K &= [\![\varphi_3]\!]_{K \restriction ([\![\varphi_1]\!]_K, [\![\varphi_2]\!]_K, \setminus)}.
\end{aligned}
$$

It remains to define the semantics for formulas of the kind $\langle\pi\rangle\varphi$, $\mathbf{perm}(\pi)\varphi$, and $\mathbf{fperm}(\pi)\varphi$. For every program $\pi \in \Pi$ let $L(\pi)$ denote the *regular language* when π is interpreted as a regular expression over some finite subset from $\mathbb{A} \cup \mathrm{Test}$. A sequence $x_0 \xrightarrow{A_1} x_1 \cdots \xrightarrow{A_n} x_n$ is called *π-path from x_0 to x_n* if

- $A_1, \ldots, A_n \in \mathrm{Tests} \cup \mathbb{A}$ and $A_1 \cdots A_n \in L(\pi)$,
- $A_i = a$ for some $a \in \mathbb{A}$ implies $x_{i-1} \to_a x_i$ for each $i \in [n]$, and
- $A_i \in \psi? \in \mathrm{Tests}$ implies $x_{i-1} = x_i$ and $x_i \in [\![\psi]\!]_K$ for each $i \in [n]$.

If moreover $A_i \in \mathbb{A}$ implies $(x_{i-1}, x_i) \in P$ for each $i \in [n]$, we call the π-path *permitted*. We define

$$
\begin{aligned}
[\![\langle\pi\rangle\varphi]\!]_K &= \{x \in W \mid \text{there exists a } \pi\text{-path from } x \text{ to a } y \in [\![\varphi]\!]_K\}, \\
[\![\mathbf{perm}(\pi)\varphi]\!]_K &= \{x \in W \mid \text{there exists a permitted } \pi\text{-path from } x \text{ to a } y \in [\![\varphi]\!]_K\}, \text{ and} \\
[\![\mathbf{fperm}(\pi)\varphi]\!]_K &= \{x \in W \mid \text{each } \pi\text{-path from } x \text{ to each } y \in [\![\varphi]\!]_K \text{ is permitted}\}.
\end{aligned}
$$

If for some world $x \in W$ we have $x \in [\![\varphi]\!]_K$, then K is a *model* for φ. A formula φ is *satisfiable* if there exists some model for φ.

Example 4.1 *Let us consider a website where the users should enter their name and their age before the actual content is displayed. When the users fill out text fields advertisement is displayed. The website manager thinks about permitting the users to skip the introduction. We abstract the website by the Kripke structure K_{ws} shown in Figure 4.1 over the atomic propositions*

$$\{\texttt{start intro}, \texttt{ad}_1, \texttt{ad}_2, \texttt{end intro}, \texttt{content}\}$$

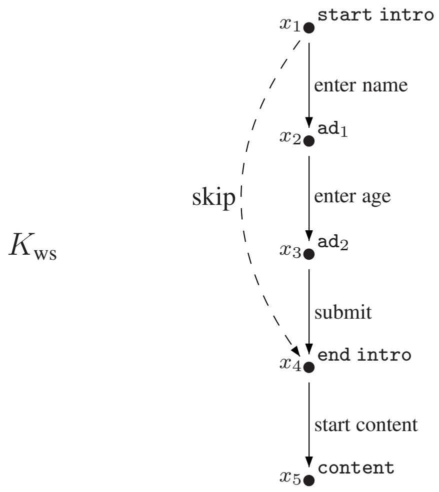


Figure 4.1: The extended Kripke structure K_{ws}.

and over the atomic programs

$$\mathsf{A} \quad = \quad \{\mathtt{enter\ name}, \mathtt{enter\ age}, \mathtt{submit}, \mathtt{start\ content}, \mathtt{skip}\}$$

and with policy set $P = \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_5)\}$*: Note that the dashed line in Figure 4.1 from* x_1 *to* x_4 *denotes the fact that* $(x_1, x_4) \notin P$ *and drawn-through lines denote members of* P*. The formula*

$$\varphi \quad = \quad \neg\mathbf{perm}\left(((\neg\mathtt{ad}_1 \wedge \neg\mathtt{ad}_2)?; \mathsf{A})^*\right)\mathtt{content}$$

states that there is no way to legally access the content without having seen either $\mathtt{ad}_1$ *or* $\mathtt{ad}_2$*. Hence we have* $x_1 \in [\![\varphi]\!]_{K_{ws}}$*. Before activating a skip-button, the website manager might want to know: Although granting a skip-button, is it true that always at least one advertisement is shown before the actual content is displayed? This can be expressed by the formula*

$$\psi \quad = \quad \mathbf{grant}(\mathtt{start\ intro}, \mathtt{end\ intro})\,\varphi.$$

However, $x_1 \notin [\![\psi]\!]_{K_{ws}}$*, since in* $K_{ws} \restriction ([\![\mathtt{start\ intro}]\!]_{K_{ws}}, [\![\mathtt{end\ intro}]\!]_{K_{ws}}, \cup)$ *neither* $\mathtt{ad}_1$ *nor* $\mathtt{ad}_2$ *occurs on the permitted path* $x_1 \xrightarrow{\mathtt{skip}} x_4 \xrightarrow{\mathtt{start\ content}} x_5$ *.* □

The size $|\varphi|$ of a $\mathrm{DLP}^+_{\mathrm{dyn}}$ formula φ and the size $|\pi|$ of a $\mathrm{DLP}^+_{\mathrm{dyn}}$ program π is inductively defined as follows,

- $|p| = |a| = 1$ for every $p \in \mathbb{P}$ and for every $a \in \mathbb{A}$,
- $|\neg\varphi| = |\varphi| + 1$,

- $|\varphi_1 \vee \varphi_2| = |\varphi_1| + |\varphi_2| + 1$,
- $|\langle\pi\rangle\varphi| = |\mathbf{perm}(\pi)\varphi| = |\mathbf{fperm}(\pi)\varphi| = |\pi| + |\varphi| + 1$,
- $|\mathbf{grant}(\varphi_1, \varphi_2)\varphi_3| = |\mathbf{revoke}(\varphi_1, \varphi_2)\varphi_3| = |\varphi_1| + |\varphi_2| + |\varphi_3| + 1$,
- $|\varphi?| = |\varphi| + 1$,
- $|\pi_1 \cup \pi_2| = |\pi_1 \circ \pi_2| = |\pi_1| + |\pi_2| + 1$, and
- $|\pi^*| = |\pi| + 1$.

4.2 Difficulties of DLP$^{+}_{\mathrm{dyn}}$ reasoning

In this section, we state known results and explain some difficulties of determining the complexity of reasoning in DLP$^{+}_{\mathrm{dyn}}$ and logics related to it.

Recall that due to Fischer/Ladner and Pratt satisfiability in PDL is EXP-complete, see Theorem 2.6. Focusing on a natural translation from DLP$^{+}_{\mathrm{dyn}}$ to PDL, Demri has recently shown the following result.

Theorem 4.2 ([24]) *There exists a satisfiability preserving reduction from DLP$^{+}_{\mathrm{dyn}}$ to PDL computable in exponential time.* □

Since the size of the resulting PDL formula in the proof of Theorem 4.2 is exponential in the size of the original DLP$^{+}_{\mathrm{dyn}}$ formula, the following theorem holds.

Theorem 4.3 ([24]) *Satisfiability in DLP$^{+}_{\mathrm{dyn}}$ is in* 2EXP. □

For the fragment DLP$_{\mathrm{dyn}}$ of DLP$^{+}_{\mathrm{dyn}}$, the following upper bound is known.

Theorem 4.4 ([95]) *Satisfiability in DLP$_{\mathrm{dyn}}$ is in* NEXP . □

Let us summarize the difficulties of identifying the exact complexity of DLP$^{+}_{\mathrm{dyn}}$.

Sabotage-likeness

By the presence of the operators **grant** and **revoke**, the truth of a formula may depend on the truth of a subformula in some modified extended Kripke structure. This is very much in the flavor of sabotage modal logic SML of van Benthem [115]. SML extends modal logic by formulas of the kind $\langle -\rangle_a\varphi$ – where $a \in \mathbb{A}$ is some atomic program and φ is an SML formula – and holds in those worlds x for which one can remove an a-labeled transition and φ holds in x of the resulting modified Kripke structure. As stated in Theorem 2.25 satisfiability in SML is undecidable. At first glance, one could think that the situation for DLP$^{+}_{\mathrm{dyn}}$ is even worse, since transitions can be removed *and* added and since moreover these transitions can be specified in the logic itself. As we will see, precisely the fact that the updated transitions are specified in the logic itself allows translations to some other logic for which satisfiability is more manageable.

Presence of intersection and negation

Another promising approach was a translation into PDL with negation of atomic programs and intersection given in [24]. This translation has the property that the *intersection width* (see also Section 3.4.2 for a precise definition) of the resulting formulas is bounded by some constant. Firstly, Theorem 3.28 implies that satisfiability formulas of PDL with intersection of constant intersection width is in EXP. Secondly, Theorem 2.16 states that PDL with negation of atomic programs is in EXP too. But unfortunately, by Theorem 3.37, PDL with intersection *and* negation of atomic programs is highly undecidable and the intersection width of the formula in the proof of Theorem 3.37 is two. Hence, we can not hope for a tight upper complexity bound for $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ by a translation into PDL with negation of atomic programs and intersection.

Reduction to PDL with the $\oplus$ operator

Let us introduce the logic PDL$\oplus$ which is basically PDL extended by the operator $\oplus$. Formally, *formulas* φ and *programs* π of the logic PDL$\oplus$ are given by the following grammar, where p ranges over $\mathbb{P}$ and a ranges over $\mathbb{A}$:

$$\begin{array}{lcl} \varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\,\varphi \\ \pi & ::= & a \mid \pi \cup \pi \mid \pi \circ \pi \mid \pi^* \mid \varphi? \mid \oplus(\pi, \varphi, \varphi) \end{array}$$

Formulas and programs of PDL$\oplus$ are interpreted in Kripke structures (hence policy sets do not occur). If K is a Kripke structure with world set W, then then for each program π, we can assign a binary relation $[\![\pi]\!]_K \subseteq W \times W$, which is defined homomorphic for $\cup, \circ, *$ and ? as for PDL, and where the semantics of the program operator $\oplus$ is defined as

$$[\![\oplus(\pi, \varphi_1, \varphi_2)]\!]_K \quad = \quad [\![\pi]\!]_K \cap \Big(([\![\varphi_1]\!]_K \times W) \cup (W \times [\![\varphi_2]\!]_K) \Big).$$

The size of $\oplus(\pi, \varphi_1, \varphi_2)$ is defined as $|\oplus(\pi, \varphi_1, \varphi_2)| = 1+|\pi|+|\varphi_1|+|\varphi_2|$. As mentioned above, a proposal of Demri [24] to improve the 2EXP upper bound for $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ is to give a polynomial time computable translation from $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ to PDL$\oplus$ and to try to decide satisfiability of PDL$\oplus$ in EXP. Unfortunately, it turns out that satisfiability in PDL$\oplus$ is 2EXP-complete, which we prove in Section 4.5.

Wrapping up, all mentioned translations have the drawback that either the target logic was too hard with respect to complexity or the translations have a blowup in formula size. Combining the EXP-hardness that $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ inherits from PDL [36], let us state the main result of this chapter.

Theorem 4.5 *Satisfiability in $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ is* EXP*-complete.* □

For proving Theorem 4.5, we first find a syntactic fragment PDL$\oplus[\mathbb{A}]$ of PDL$\oplus$ and give a satisfiability preserving polynomial time computable translation from $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ to PDL$\oplus[\mathbb{A}]$ in Section 4.3. The translation is based on a precise analysis of how $\mathbf{grant}$ and $\mathbf{revoke}$ operators can influence subformulas. In a second step, we prove in Section 4.4 that PDL$\oplus[\mathbb{A}]$ is in EXP via a reduction to the emptiness problem of two-way alternating parity tree automata over infinite trees.

4.3 A translation from DLP$^+_{\text{dyn}}$ to PDL$\oplus[\mathbb{A}]$

The logic PDL$\oplus[\mathbb{A}]$ is the syntactic fragment of PDL$\oplus$, where the program arguments of each $\oplus$ program must either be an atomic program or a $\oplus$ program itself. More formally, *formulas* φ, *basic programs* α and *programs* π of PDL$\oplus[\mathbb{A}]$ are given by the following grammar, where p ranges over $\mathbb{P}$ and a ranges over $\mathbb{A}$:

$$\begin{array}{lll} \varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\,\varphi \\ \alpha & ::= & a \mid \oplus(\alpha, \varphi, \varphi) \\ \pi & ::= & \alpha \mid \pi \cup \pi \mid \pi \circ \pi \mid \pi^* \mid \varphi? \end{array}$$

Remark 4.6 *Note that each PDL$\oplus$ program $\oplus(\pi, \varphi_1, \varphi_2)$ can be translated into an equivalent PDL program $\oplus(\pi, \varphi_1, \varphi_2)'$ as follows, where π', φ_1', and φ_2' are inductively the translations for π, φ_1, and φ_2 respectively:*

$$\oplus(\pi, \varphi_1, \varphi_2)' \quad = \quad \left(\varphi_1'? \circ \pi' \;\cup\; \pi' \circ \varphi_2'?\right)$$

Hence, the logics PDL$\oplus$, PDL$\oplus[\mathbb{A}]$, and PDL are equi-expressive. However, by the presence of $\oplus$ programs, the translation causes an exponential blowup. Thus, the three logics may differ in succinctness. In fact, our 2EXP-completeness result for PDL$\oplus$ shown in Section 4.5 implies that there does *not* even exist a *satisfiability preserving* translation from PDL$\oplus$ to PDL that is computable in polynomial time.

In parts our translation combines some ideas from [24] and [72]. First, we introduce a notion of certain modified Kripke structures. Recall that in this chapter Φ denotes the set of all DLP$^+_{\text{dyn}}$ formulas. Let $\Sigma = (\Phi \times \Phi \times \{\cup, \setminus\})$. For each $\sigma = (\theta_1, \theta_1', \text{op}_1) \cdots (\theta_k, \theta_k', \text{op}_k) \in \Sigma^*$, where $k \geq 0$, let $U_\sigma = \{m \in [k] \mid \text{op}_m = \cup\}$ be the positions in σ that contain $\cup$ and let $M_\sigma = \{m \in [k] \mid \text{op}_m = \setminus\}$ be the positions in σ that contain $\setminus$. If K is an extended Kripke structure, then for every $\sigma \in \Sigma^*$ define the extended Kripke structure $K \restriction \sigma$, inductively by the length of σ as follows,

$$\begin{array}{rcll} K \restriction \varepsilon & = & K & \text{and} \\ K \restriction \sigma(\theta_1, \theta_2, \text{op}) & = & (K \restriction \sigma) \restriction ([\![\theta_1]\!]_{K\restriction\sigma}, [\![\theta_2]\!]_{K\restriction\sigma}, \text{op}) & \text{for each } (\theta_1, \theta_2, \text{op}) \in \Sigma \end{array}$$

Remark 4.7 *If K is an extended Kripke structure and $\sigma \in \Sigma^*$, then the extended Kripke structures K and $K \restriction \sigma$ only differ in their policy set. Thus, for all $a \in \mathbb{A}$ we have $[\![a]\!]_K = [\![a]\!]_{K\restriction\sigma}$ and for all $p \in \mathbb{P}$ we have $[\![p]\!]_K = [\![p]\!]_{K\restriction\sigma}$.* $\square$

For a formula $\varphi \in \Phi$, let $\text{Occ}(\varphi)$ denote the set of all occurrences of subformulas of φ. As it simplifies notation, we will sometimes treat each $\psi \in \text{Occ}(\varphi)$ as a DLP$^+_{\text{dyn}}$ formula instead of as an occurrence of a subformula of φ. Moreover, for each $\psi \in \text{Occ}(\varphi)$, define the unique sequence $\sigma(\psi) \in \Sigma^*$ that we get by considering the $\mathbf{grant}$ and $\mathbf{revoke}$ operators that occur along the path from φ to ψ in the syntax tree of φ. Formally, the mapping $\sigma : \text{Occ}(\varphi) \to \Sigma$ is defined in a top down manner as follows,

- $\sigma(\varphi) = \varepsilon$,
- if $\psi = \neg\chi$, then $\sigma(\chi) = \sigma(\psi)$,
- if $\psi = \psi_1 \vee \psi_2$, then $\sigma(\psi_1) = \sigma(\psi_2) = \sigma(\psi)$,
- if $\psi = \langle\pi\rangle\chi$, $\psi = \mathbf{perm}(\pi)\chi$, or $\psi = \mathbf{fperm}(\pi)\chi$, then $\sigma(\chi) = \sigma(\theta) = \sigma(\psi)$ for every test program $\theta?$ of π,
- if $\psi = \mathbf{grant}(\psi_1, \psi_2)\chi$, then $\sigma(\psi_1) = \sigma(\psi_2) = \sigma(\psi)$, and $\sigma(\chi) = \sigma(\psi)(\psi_1, \psi_2, \cup)$, and
- if $\psi = \mathbf{revoke}(\psi_1, \psi_2)\chi$, then $\sigma(\psi_1) = \sigma(\psi_2) = \sigma(\psi)$, and $\sigma(\chi) = \sigma(\psi)(\psi_1, \psi_2, \setminus)$.

Before giving the translation from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to $\mathrm{PDL}\oplus[\mathbb{A}]$, the following lemma characterizes the policy set of $K \restriction \sigma$ for every $\sigma \in \Sigma^*$.

Lemma 4.8 *Let $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ be an extended Kripke structure, $\sigma = (\theta_1, \theta'_1, \mathrm{op}_1) \cdots (\theta_k, \theta'_k, \mathrm{op}_k) \in \Sigma^*$ and let P_σ be the policy set of $K \restriction \sigma$. Firstly, for each $a \in \mathbb{A}$, we have*

$$\begin{aligned}
[\![a]\!]_{K\restriction\sigma} \cap P_\sigma \quad = \quad & [\![a]\!]_K \cap \Bigg[\Bigg(P \cap \bigcap_{j \in M_\sigma} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}}\Bigg) && (4.1)\\
& \cup \Bigg(\bigcup_{l \in U_\sigma} \Big([\![\theta_l]\!]_{K\restriction\sigma^{(l-1)}} \times [\![\theta'_l]\!]_{K\restriction\sigma^{(l-1)}}\Big) && (4.2)\\
& \cap \bigcap_{j \in M_\sigma \cap [l,k]} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}}\Bigg)\Bigg]. && (4.3)
\end{aligned}$$

Secondly, for each $a \in \mathbb{A}$, we have

$$\begin{aligned}
[\![a]\!]_{K\restriction\sigma} \setminus P_\sigma \quad = \quad & [\![a]\!]_K \cap \Bigg[\Bigg((W \times W) \setminus P \cap \bigcap_{j \in U_\sigma} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}}\Bigg) && (4.4)\\
& \cup \Bigg(\bigcup_{l \in M_\sigma} \Big([\![\theta_l]\!]_{K\restriction\sigma^{(l-1)}} \times [\![\theta'_l]\!]_{K\restriction\sigma^{(l-1)}}\Big) && (4.5)\\
& \cap \bigcap_{j \in U_\sigma \cap [l,k]} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}}\Bigg)\Bigg]. && (4.6)
\end{aligned}$$

$\square$

Before sketching the proof for it, let us summarize the intuition behind Lemma 4.8. Let $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ be some extended Kripke structure, let $\sigma \in \Sigma$, and let $a \in \mathbb{A}$. Moreover, let $(x, y) \in W \times W$ be a pair of worlds. Assume that P_σ is the policy set of $K \restriction \sigma$. Having $(x, y) \in [\![a]\!]_{K\restriction\sigma} \cap P_\sigma$ is equivalent to having $(x, y) \in [\![a]\!]_K$ such that either (x, y) is already in the policy set P and is never removed by any element of σ at a position from M_σ, see also (4.1), or that there is some witness $(\theta_l, \theta'_l, \cup)$ in σ such that $x \in [\![\theta_l]\!]_{K\restriction\sigma^{(l-1)}}$ and $y \in [\![\theta'_l]\!]_{K\restriction\sigma^{(l-1)}}$, see also (4.2), and moreover the pair (x, y) is then never removed by any element from σ at position $j \in M_\sigma$ with $j \geq i$, see also (4.3). Let us analogously describe the set $[\![a]\!]_{K\restriction\sigma} \setminus P_\sigma$. Having $(x, y) \in [\![a]\!]_{K\restriction\sigma} \setminus P_\sigma$ is equivalent to having $(x, y) \in [\![a]\!]_K$ such that (x, y) is *not* in the policy set P and is never added by any element of σ at a position from U_σ, see also (4.4), or there is some witness $(\theta_l, \theta'_l, \setminus)$ in σ such that $x \in [\![\theta_l]\!]_{K\restriction\sigma^{(l-1)}}$ and $y \in [\![\theta'_l]\!]_{K\restriction\sigma^{(l-1)}}$, see also (4.5), and moreover the pair (x, y) is then never added by any element from σ at position $j \in U_\sigma$ with $j \geq l$, see also (4.6).

PROOF (OF LEMMA 4.8,SKETCH) We proceed by induction on the length k of σ. So fix some $a \in \mathbb{A}$. If $k = 0$, then $\sigma = \varepsilon$, and $M_\sigma = U_\sigma = \emptyset$. Hence $[\![a]\!]_K \cap P_\sigma = [\![a]\!]_K \cap P$ by definition of $K \restriction \sigma$. Now assume the lemma holds for all $\tau \in \Sigma^k$. Let $\sigma = \tau(\theta_{k+1}, \theta'_{k+1}, \mathrm{op}_{k+1})$ for some $\tau = (\theta_1, \theta'_1, \mathrm{op}_1) \cdots (\theta_k, \theta'_k, \mathrm{op}_k) \in \Sigma^k$. Let P_τ denote the policy set of $K \restriction \tau$. We only consider the case when $\mathrm{op}_{k+1} = \setminus$ and show the lemma for $[\![a]\!]_{K\restriction\sigma} \cap P_\sigma$. The following equalities hold then:

$$
\begin{aligned}
[\![a]\!]_{K\restriction\sigma} \cap P_\sigma &= [\![a]\!]_{K\restriction\tau} \cap (P_\tau \setminus ([\![\theta_{k+1}]\!]_{K\restriction\tau} \times [\![\theta'_{k+1}]\!]_{K\restriction\tau})) \\
&= ([\![a]\!]_{K\restriction\tau} \cap P_\tau) \cap [\![\oplus(a, \neg\theta_{k+1}, \neg\theta'_{k+1})]\!]_{K\restriction\tau} \\
&\overset{\text{IH}}{=} [\![a]\!]_K \cap \Bigg[\Bigg(P \cap \bigcap_{j \in M_\tau} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\tau^{(j-1)}} \cap [\![\oplus(a, \neg\theta_{k+1}, \neg\theta'_{k+1})]\!]_{K\restriction\tau} \Bigg) \\
&\qquad \cup \Bigg(\bigcup_{l \in U_\tau} \Big([\![\theta_l]\!]_{K\restriction\tau^{(l-1)}} \times [\![\theta'_l]\!]_{K\restriction\tau^{(l-1)}} \Big) \\
&\qquad\qquad \cap \bigcap_{j \in M_\tau \cap [l,k]} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\tau^{(j-1)}} \\
&\qquad\qquad \cap [\![\oplus(a, \neg\theta_{k+1}, \neg\theta'_{k+1})]\!]_{K\restriction\tau} \Bigg) \Bigg] \\
&= [\![a]\!]_K \cap \Bigg[\Bigg(P \cap \bigcap_{j \in M_\sigma} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}} \Bigg) \\
&\qquad \cup \Bigg(\bigcup_{l \in U_\sigma} \Big([\![\theta_l]\!]_{K\restriction\sigma^{(l-1)}} \times [\![\theta'_l]\!]_{K\restriction\sigma^{(l-1)}} \Big) \\
&\qquad\qquad \cap \bigcap_{j \in M_\sigma \cap [l,k+1]} [\![\oplus(a, \neg\theta_j, \neg\theta'_j)]\!]_{K\restriction\sigma^{(j-1)}} \Bigg) \Bigg]
\end{aligned}
$$

The other cases can be proven analogously. ■

Let us turn to our translation. For this, fix some DLP$^+_{\mathrm{dyn}}$ formula φ over atomic programs A and over atomic propositions P for the rest of this section. Let $\psi_1, \ldots, \psi_n$ be an enumeration of

$\mathrm{Occ}(\varphi)$ and assume $\psi_n = \varphi$. Let $\mathsf{A}' = \mathsf{A} \uplus \overline{\mathsf{A}}$, where $\overline{\mathsf{A}} = \{\overline{a} \mid a \in \mathsf{A}\}$ is a disjoint copy of A. Define some set of atomic propositions $\mathsf{P}' = \mathsf{P} \uplus \{p_1, \ldots, p_n\}$. Below, we also write $p(\psi_i)$ for p_i whenever $\psi_i \in \mathrm{Occ}(\varphi)$. Let $\sigma = (\theta_1, \theta'_1, \mathrm{op}_1) \cdots (\theta_k, \theta'_k, \mathrm{op}_k) \in \sigma(\mathrm{Occ}(\varphi))$. Then for every subset $S = \{j_1, \ldots, j_l\} \subseteq [k]$ of positions in σ, where $j_1 < j_2 < \cdots < j_l$, and every atomic program $b \in \mathsf{A}'$, let us define the PDL$\oplus[\mathbb{A}]$ program $b^S = b^S_l$, where

$$b^S_0 = b \qquad \text{and} \qquad b^S_h = \oplus(b^S_{h-1}, \neg p(\theta_{j_h}), \neg p(\theta'_{j_h})) \quad \text{for all } h \in [l].$$

Note that the equality

$$[\![b^S]\!]_K \quad = \quad \bigcap_{j \in S} [\![\oplus(b, \neg p(\theta_j), \neg p(\theta'_j))]\!]_K$$

holds for each Kripke structure K.

We will construct a PDL$\oplus[\mathbb{A}]$ formula φ' over the atomic programs A' and over the atomic propositions P' such that φ is satisfiable if and only if φ' is satisfiable. Intuitively, if K is a model for φ with policy set P and K' is a model of φ', think of the relation $[\![a]\!]_{K'}$ as $[\![a]\!]_K \cap P$ and of $[\![\overline{a}]\!]_{K'}$ as $[\![a]\!]_K \setminus P$.

Let us first, for every $\psi_i \in \mathrm{Occ}(\varphi)$, define the PDL$\oplus[\mathbb{A}]$ formula $\widehat{\psi_i}$ over the atomic propositions P' and over the atomic programs A' inductively as follows:

- If $\psi_i = p$ for some $p \in \mathsf{P}$, then $\widehat{\psi_i} = p$.
- If $\psi_i = \neg\psi_j$, then $\widehat{\psi_i} = \neg p_j$.
- If $\psi_i = \psi_j \vee \psi_k$, then $\widehat{\psi_i} = p_j \vee p_k$.
- If $\psi_i = \mathbf{grant}(\psi_j, \psi_k)\psi_l$ or $\psi_i = \mathbf{revoke}(\psi_j, \psi_k)\psi_l$, then $\widehat{\psi_i} = p_l$.
- If $\psi_i = \langle\pi\rangle\psi_j$, then $\widehat{\psi_i} = \langle T(\pi)\rangle p_j$, where $T(\pi)$ is defined homomorphic on $\cup, \circ$, and on $*$, $T(\psi_k?) = p_k?$ for every test program $\psi_k?$, and $T(a) = a \cup \overline{a}$ for every $a \in \mathsf{A}$.
- Assume $\psi_i = \mathbf{perm}(\pi)\psi_j$ and $\sigma = \sigma(\psi_i) = (\theta_1, \theta'_1, \mathrm{op}_1) \cdots (\theta_k, \theta'_k, \mathrm{op}_k)$. Recall that $U_\sigma = \{l \in [k] \mid \mathrm{op}_l = \cup\}$ and $M_\sigma = \{l \in [k] \mid \mathrm{op}_l = \setminus\}$. Then, we define $\widehat{\psi_i} = \langle T^\forall(\pi)\rangle p_j$, where $T^\forall(\pi)$ is defined homomorphic on $\cup, \circ$, and on $*$, $T^\forall(\psi_l?) = p_l?$ for every test program $\psi_l?$. For every $a \in \mathsf{A}$ we define:

$$T^\forall(a) \qquad = \qquad a^{M_\sigma} \cup \bigcup_{b \in \{a, \overline{a}\}} \bigcup_{l \in U_\sigma} p(\theta_l)? \circ b^{M_\sigma \cap [l,k]} \circ p(\theta'_l)?$$

  The intention behind $T^\forall(a)$ is to define $[\![a]\!]_{K \restriction \sigma} \cap P_\sigma$ by usage of Lemma 4.8 for extended Kripke structures K with P_σ denoting the policy set of $K \restriction \sigma$.

- Assume $\psi_i = \mathbf{fperm}(\pi)\psi_j$ and $\sigma = \sigma(\psi_i) = (\theta_1, \theta'_1, \mathrm{op}_1) \cdots (\theta_k, \theta'_k, \mathrm{op}_k)$. Then, we define $\widehat{\psi_i} = \neg\langle T^\exists(\pi)\rangle p_j$, where $T^\exists(\pi)$ is inductively defined as follows:
  - $T^\exists(\pi_1 \cup \pi_2) = T^\exists(\pi_1) \cup T^\exists(\pi_2)$
  - $T^\exists(\pi_1 \circ \pi_2) = T^\exists(\pi_1) \circ T(\pi_2) \ \cup \ T(\pi_1) \circ T^\exists(\pi_2)$, where T is defined as above.

- $T^{\exists}(\pi^*) = T(\pi^*) \circ T^{\exists}(\pi) \circ T(\pi^*)$, where T is defined as above.
- $T^{\exists}(\psi_l?) = \texttt{false}?$
- For every $a \in \mathsf{A}$ we define:

$$T^{\exists}(a) \quad = \quad \overline{a}^{U_\sigma} \cup \bigcup_{b \in \{a, \overline{a}\}} \bigcup_{l \in M_\sigma} p(\theta_l)? \circ b^{U_\sigma \cap [l,k]} \circ p(\theta'_l)?$$

Similarly as $T^{\forall}(a)$ corresponds to $[\![a]\!]_{K\restriction\sigma} \cap P_\sigma$, the intention behind $T^{\exists}(a)$ is to define $[\![a]\!]_{K\restriction\sigma} \setminus P_\sigma$ by usage of Lemma 4.8 for extended Kripke structures K with P_σ denoting the policy set of $K \restriction \sigma$.

Recall $\varphi = \psi_n$. We are now ready to define φ'.

$$\varphi' \quad = \quad p_n \wedge [(\mathsf{A} \cup \overline{\mathsf{A}})^*] \bigwedge_{i \in [n]} (p_i \leftrightarrow \widehat{\psi_i}).$$

Before proving that φ and φ' are equi-satisfiable, let us generously estimate the size of φ'.

Lemma 4.9 $|\varphi'| \in \mathcal{O}(|\varphi|^4)$. □

Proof First of all, it is easy to see that $|T(\pi)| \leq 2 \cdot |\pi| + 1$ for every program π. Next, observe that

$$|b^S| \quad \leq \quad 1 + 5 \cdot |S| \ \leq \ 1 + 5 \cdot |\varphi|$$

for every index set S and every $b \in \mathsf{A} \cup \overline{\mathsf{A}}$. Moreover it is easy to see that there exists some constant c such that for every $b \in \mathsf{A} \cup \overline{\mathsf{A}}$ we have

$$T^{\forall}(b), T^{\exists}(b) \quad \leq \quad c \cdot |\varphi|^2.$$

Let us assume that $c \geq 9$. An easy induction on the structure of π yields $T^{\forall}(\pi) \leq c \cdot |\pi|^2$ for every program π that occurs in φ. Finally we prove by structural induction that

$$T^{\exists}(\pi) \quad \leq \quad c \cdot |\varphi|^2 \cdot |\pi|$$

for every program π that appears in φ. The cases $\pi \in \mathsf{A}$, $\pi = \psi?$, and $\pi = \pi_1 \cup \pi_2$ are straightforward. It remains to prove the cases $\pi = \pi_1 \circ \pi_2$ and $\pi = \gamma^*$. If $\pi = \pi_1 \circ \pi_2$, then

$$\begin{aligned}
|T^{\exists}(\pi)| &= |T^{\exists}(\pi_1) \circ T(\pi_2) \cup T(\pi_1) \circ T^{\exists}(\pi_2)| \\
&\overset{\mathrm{IH}}{\leq} c \cdot |\varphi|^2 \cdot (|\pi_1| + |\pi_2|) + 2 \cdot |\pi_1| + 1 + 2 \cdot |\pi_2| + 1 + 3 \\
&\leq c \cdot |\varphi|^2 \cdot (|\pi_1| + |\pi_2| + 1) \\
&= c \cdot |\varphi|^2 \cdot |\pi|.
\end{aligned}$$

If $\pi = \gamma^*$, then

$$\begin{aligned}
T^{\exists}(\pi) &= |T(\pi) \circ T^{\exists}(\gamma) \circ T(\pi)| \\
&= 2 \cdot |T(\pi)| + 2 + |T^{\exists}(\gamma)| \\
&\overset{\mathrm{IH}}{\leq} 4 \cdot |\pi| + 4 + c \cdot |\varphi|^2 \cdot |\gamma| \\
&\leq c \cdot |\varphi|^2 \cdot (|\gamma| + 1) \\
&= c \cdot |\varphi|^2 \cdot |\pi|.
\end{aligned}$$

Thus, it follows that $|\widehat{\psi}_i| \in \mathcal{O}(|\varphi|^3)$ for each $i \in [n]$. Since

$$\varphi' \quad = \quad p_n \wedge [(\mathsf{A} \cup \overline{\mathsf{A}})^*] \bigwedge_{i \in [n]} (p_i \leftrightarrow \widehat{\psi}_i),$$

it follows that $|\varphi'| \in \mathcal{O}(|\varphi|^4)$. ■

The correctness of the above translation follows from the following lemma.

Lemma 4.10 *The formula φ is satisfiable if and only if φ' is satisfiable.* □

PROOF "Only-if": Assume φ is satisfiable. Hence there exists an extended Kripke structure $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\}, P)$ over the atomic propositions P and over the atomic programs A such that for some world $x_0 \in W$ we have $x_0 \in [\![\varphi]\!]_K$. By Lemma 3.3. in [24], we can assume that for each $a, b \in \mathsf{A}$ with $a \neq b$, we have $\rightarrow_a \cap \rightarrow_b = \emptyset$. Recall that $\psi_1, \ldots, \psi_n$ is an enumeration of $\mathrm{Occ}(\varphi)$ with $\varphi = \psi_n$. Define the Kripke structure $K' = (W, \{\rightarrow'_a \mid a \in \mathsf{A} \cup \overline{\mathsf{A}}\}, \{W'_p \mid p \in \mathsf{P}'\})$ over the atomic propositions $\mathsf{P}' = \mathsf{P} \uplus \{p_1, \ldots, p_n\}$ and over the atomic programs $\mathsf{A}' = \mathsf{A} \uplus \overline{\mathsf{A}}$ as follows,

$$\begin{aligned}
\rightarrow'_a &= \rightarrow_a \cap P \quad \text{for each } a \in \mathsf{A}, \\
\rightarrow'_{\overline{a}} &= \rightarrow_a \setminus P \quad \text{for each } \overline{a} \in \overline{\mathsf{A}}, \\
W_p &= W_p \quad \text{for each } p \in \mathsf{P}, \text{ and} \\
W_{p_i} &= [\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} \quad \text{for each } i \in [n].
\end{aligned}$$

So note that we have $[\![p_i]\!]_{K'} = [\![\psi_i]\!]_{K \restriction \sigma(\psi_i)}$ for each $i \in [n]$. Now, the following claim holds:

Claim: We have $[\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} = [\![\widehat{\psi}_i]\!]_{K'}$ for each $i \in [n]$.

The claim is proven below. Let us show that the claim implies that K' is a model of φ'. Recall that $\varphi = \psi_n$ and $K \restriction \sigma(\psi_n) = K$. Since $x_0 \in [\![\varphi]\!]_K$, it follows $x_0 \in [\![p_n]\!]_{K'}$ by definition of K'. Moreover by the above claim and by definition of K' it follows that for each $i \in [n]$ we have

$$[\![p_i]\!]_{K'} \quad = \quad [\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} \quad = \quad [\![\widehat{\psi}_i]\!]_{K'}.$$

Hence $x_0 \in [\![p_n \wedge [(\mathsf{A} \cup \overline{\mathsf{A}})^*] \bigwedge_{i \in [n]} (p_i \leftrightarrow \widehat{\psi}_i)]\!]_{K'}$.

We prove the above claim by a case distinction of the structure of ψ_i.

- If ψ_i for some $p \in \mathsf{P}$, then

$$[\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} = [\![p]\!]_K = [\![p]\!]_{K'} = [\![\widehat{\psi}_i]\!]_{K'}.$$

- If $\psi_i = \neg \psi_j$, then

$$[\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} \;=\; W \setminus [\![\psi_j]\!]_{K \restriction \sigma(\psi_j)} \;=\; W \setminus [\![p_j]\!]_{K'} \;=\; [\![\neg p_j]\!]_{K'} \;=\; [\![\widehat{\psi}_i]\!]_{K'}.$$

- If $\psi_i = \psi_j \vee \psi_k$, then

$$[\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} \;=\; [\![\psi_j]\!]_{K \restriction \sigma(\psi_j)} \cup [\![\psi_k]\!]_{K \restriction \sigma(\psi_k)} \;=\; [\![p_j \vee p_k]\!]_{K'} \;=\; [\![\widehat{\psi}_i]\!]_{K'}.$$

- If $\psi_i = \mathbf{grant}(\psi_j, \psi_k)\psi_l$, then

$$[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} \;=\; [\![\psi_l]\!]_{(K\restriction\sigma(\psi_i))\restriction(\psi_j,\psi_k,\cup)} \;=\; [\![\psi_l]\!]_{K\restriction\sigma(\psi_l)} \;=\; [\![p_l]\!]_{K'} \;=\; [\![\widehat{\psi_i}]\!]_{K'}.$$

- If $\psi_i = \mathbf{revoke}(\psi_j, \psi_k)\psi_l$, then

$$[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} \;=\; [\![\psi_l]\!]_{(K\restriction\sigma(\psi_i))\restriction(\psi_j,\psi_k,\setminus)} \;=\; [\![\psi_l]\!]_{K\restriction\sigma(\psi_l)} \;=\; [\![p_l]\!]_{K'} \;=\; [\![\widehat{\psi_i}]\!]_{K'}.$$

- If $\psi_i = \langle\pi\rangle\psi_j$, then by definition of K' for every test program $\psi_k?$ that occurs in π we have $[\![\psi_k]\!]_{K\restriction\sigma(\psi_k)} = [\![p_k]\!]_{K'}$ and moreover $\sigma(\psi_k) = \sigma(\psi_i)$. Analogously, we have $[\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} = [\![p_j]\!]_{K'}$ and $\sigma(\psi_j) = \sigma(\psi_i)$. Now let $a \in \mathsf{A}$ be arbitrary. By Remark 4.7, we have $[\![a]\!]_{K\restriction\sigma(\psi_i)} = [\![a]\!]_K$ and by construction of K' we have $[\![a]\!]_K = [\![a]\!]_{K'} \cup [\![\overline{a}]\!]_{K'}$, hence $[\![a]\!]_{K\restriction\sigma(\psi_i)} = [\![a]\!]_{K'} \cup [\![\overline{a}]\!]_{K'} = [\![T(a)]\!]_{K'}$. Since $T(\pi)$ is defined homomorphic on $\cup, \circ$, and on $*$, and $T(\psi_k?) = p_k?$ for test programs $\psi_k?$, it is clear that $[\![\langle\pi\rangle\psi_j]\!]_{K\restriction\sigma(\psi_i)} = [\![T(\pi)p_j]\!]_{K'} = [\![\widehat{\psi_i}]\!]_{K'}$.

- Assume $\psi_i = \mathbf{perm}(\pi)\psi_j$, $\sigma = \sigma(\psi_i) = (\theta_1, \theta_1', \mathrm{op}_1)\cdots(\theta_k, \theta_k', \mathrm{op}_k)$ and P_σ is the policy set of $K \restriction \sigma$. By definition of K' for every $l \in [n]$ such that either $\psi_l?$ occurs in π or ψ_l occurs in σ we have $[\![\psi_l]\!]_{K\restriction\sigma(\psi_l)} = [\![p_l]\!]_{K'}$ and analogously $[\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} = [\![p_j]\!]_{K'}$. Recall that $[\![a]\!]_{K\restriction\tau} = [\![a]\!]_{K'} \cup [\![\overline{a}]\!]_{K'}$ for each $a \in \mathsf{A}$ and each prefix τ of σ. Also recall that by definition of K' we have $[\![a]\!]_K \cap P = [\![a]\!]_{K'}$ for each $a \in \mathsf{A}$. Thus, by Lemma 4.8 one can verify that for each $a \in \mathsf{A}$ we have

$$[\![a]\!]_{K\restriction\sigma} \cap P_\sigma \quad = \quad [\![a^{M_\sigma} \cup \bigcup_{b\in\{a,\overline{a}\}} \bigcup_{l\in U_\sigma} p(\theta_l)? \circ b^{M_\sigma\cap[l,k]} \circ p(\theta_l')?]\!]_{K'}.$$

Hence $[\![a]\!]_{K\restriction\sigma} \cap P_\sigma = [\![T^\forall(\pi)]\!]_{K'}$ for each $a \in \mathsf{A}$. Since $T^\forall$ is defined homomorphic on $\cup, \circ$, and on $*$ and since $T^\forall(\psi_l?) = p_l?$ for test programs $\psi_l?$, it easily follows

$$[\![T^\forall(\pi)]\!]_{K'} = \{(x, y) \in W \times W \mid \exists \text{ permitted } \pi\text{-path from } x \text{ to } y \text{ in } K \restriction \sigma\}.$$

Finally, we obtain

$$[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} \quad = \quad [\![\mathbf{perm}(\pi)\psi_j]\!]_{K\restriction\sigma(\psi_i)} \quad = \quad [\![\langle T^\forall(\pi)\rangle p_j]\!]_{K'} \quad = \quad [\![\widehat{\psi_i}]\!]_{K'}.$$

- Assume $\psi_i = \mathbf{fperm}(\pi)\psi_j$, $\sigma = \sigma(\psi_i) = (\theta_1, \theta_1', \mathrm{op}_1)\cdots(\theta_k, \theta_k', \mathrm{op}_k)$ and P_σ is the policy set of $K \restriction \sigma$. By definition of K' for every $l \in [n]$ such that either $\psi_l?$ occurs in π or ψ_l occurs in σ we have $[\![\psi_l]\!]_{K\restriction\sigma(\psi_l)} = [\![p_l]\!]_{K'}$ and analogously $[\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} = [\![p_j]\!]_{K'}$. Recall that $[\![a]\!]_{K\restriction\tau} = [\![a]\!]_{K'} \cup [\![\overline{a}]\!]_{K'}$ for each $a \in \mathsf{A}$ and each prefix τ of σ. Also recall that by definition of K' we have $[\![a]\!]_K \setminus P = [\![\overline{a}]\!]_{K'}$. Thus, by Lemma 4.8 one can verify that for each $a \in \mathsf{A}$ we have

$$[\![a]\!]_{K\restriction\sigma} \setminus P_\sigma \quad = \quad [\![\overline{a}^{U_\sigma} \cup \bigcup_{b\in\{a,\overline{a}\}} \bigcup_{l\in M_\sigma} p(\theta_l)? \circ b^{U_\sigma\cap[l,k]} \circ p(\theta_l')?]\!]_{K'}.$$

Hence $[\![a]\!]_{K\restriction\sigma} \setminus P_\sigma = [\![T^{\exists}(a)]\!]_{K'}$ for each $a \in \mathsf{A}$. By definition of $T^{\exists}$ it easily follows

$$[\![T^{\exists}(\pi)]\!]_{K'} = \{(x,y) \in W \times W \mid \exists \text{ non-permitted } \pi\text{-path from } x \text{ to } y \text{ in } K \restriction \sigma\}.$$

Hence, we obtain

$$[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} \quad = \quad [\![\mathbf{fperm}(\pi)\psi_j]\!]_{K\restriction\sigma(\psi_i)} \quad = \quad [\![\neg\langle T^{\exists}(\pi)\rangle p_j]\!]_{K'} \quad = \quad [\![\widehat{\psi_i}]\!]_{K'}.$$

"If": Assume φ' is satisfiable. Hence, there exists a Kripke structure $K' = (W, \{\rightarrow'_a \mid a \in \mathsf{A}'\}, \{W_p \mid p \in \mathsf{P}'\})$ over the atomic propositions $\mathsf{P}' = \mathsf{P} \uplus \{p_1, \ldots, p_n\}$ and over the atomic programs $\mathsf{A}' = \mathsf{A} \uplus \overline{\mathsf{A}}$ such that for some $x_0 \in W$ we have $x_0 \in [\![\varphi']\!]_{K'}$. Since PDL$\oplus[\mathbb{A}]$ is equally expressive as PDL we can assume that for all $a, b \in \mathsf{A}'$ with $a \neq b$ we have $\rightarrow'_a \cap \rightarrow'_b = \emptyset$ by Lemma 3.3 from [24].

Let $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\}, P)$ be the extended Kripke structure over the atomic propositions P and over the atomic programs A, where for all $a \in \mathsf{A}$ we have $\rightarrow_a = \rightarrow'_a \cup \rightarrow'_{\overline{a}}$ and $P = \bigcup_{a\in\mathsf{A}} \rightarrow'_a$. Let $R = \left\{x \in W \mid (x_0, x) \in \left(\bigcup_{a\in\mathsf{A}} \rightarrow'_a \cup \rightarrow'_{\overline{a}}\right)^*\right\}$ denote the set of worlds that are *reachable* from x_0. We can assume that $W = R$ without loss of generality since one can prove by a simple induction on the structure of φ that $x_0 \in [\![\varphi]\!]_K$ if and only if $x_0 \in [\![\varphi]\!]_{K_R}$, where K_R denotes the Kripke structure K restricted to the world set R. The following claim is proven below.

Claim: We have $[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} = [\![p_i]\!]_{K'}$ for every $i \in [n]$.

Before proving the above claim, let us show that it implies that K is a model of φ. By assumption, we have $x_0 \in [\![\varphi']\!]_{K'}$ and recall that

$$\varphi' \quad = \quad p_n \wedge [(\mathsf{A} \cup \overline{\mathsf{A}})^*] \bigwedge_{i\in[n]} (p_i \leftrightarrow \widehat{\psi_i}).$$

Thus also $x_0 \in [\![p_n]\!]_{K'}$. Hence by application of the above claim, we have $x_0 \in [\![\psi_n]\!]_{K\restriction\sigma(\psi_n)}$ and finally $x_0 \in [\![\varphi]\!]_K$ since $\psi_n = \varphi$ and $K \restriction \sigma(\psi_n) = K$.

It remains to prove the above claim. For this, let us define the binary relation $< \subseteq \mathrm{Occ}(\varphi) \times \mathrm{Occ}(\varphi)$ as follows:

$$\psi_i < \psi_j \quad \Leftrightarrow \quad \psi_i \neq \psi_j \text{ and } (\psi_i \text{ occurs as subformula of } \psi_j \text{ or } \psi_i \text{ occurs in } \sigma(\psi_j)).$$

Observe that the relation $<$ is acyclic. Define $\sqsubset = <^+$. Hence, the relation $\sqsubset$ is a strict partial order on $\mathrm{Occ}(\varphi)$.

Before proving the above claim by induction on $\sqsubset$, keep in mind that $[\![p_i]\!]_{K'} = [\![\widehat{\psi_i}]\!]_{K'}$ for each $i \in [n]$ due to definition of φ' and due to $W = R$.

Base. Let ψ_i be minimal with respect to $\sqsubset$. Then $\psi_i = p$ for some $p \in \mathsf{P}$. Hence,

$$[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} \quad = \quad [\![p]\!]_{K'} \quad = \quad [\![p_i]\!]_{K'}.$$

Step.

- If $\psi_i = \neg\psi_j$, then

$$\begin{array}{rcl} [\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} & = & W \setminus [\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} \\ & \overset{\text{IH}}{=} & W \setminus [\![p_j]\!]_{K'} \\ & = & [\![\widehat{\psi_i}]\!]_{K'} \\ & = & [\![p_i]\!]_{K'}. \end{array}$$

- If $\psi_i = \psi_j \vee \psi_k$, then

$$\begin{array}{rcl} [\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} & = & [\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} \cup [\![\psi_k]\!]_{K\restriction\sigma(\psi_k)} \\ & \overset{\text{IH}}{=} & [\![p_j]\!]_{K'} \cup [\![p_k]\!]_{K'} \\ & = & [\![p_j \vee p_k]\!]_{K'} \\ & = & [\![\widehat{\psi_i}]\!]_{K'} \\ & = & [\![p_i]\!]_{K'}. \end{array}$$

- If $\psi_i = \mathbf{grant}(\psi_j, \psi_k)\psi_l$, then

$$\begin{array}{rcl} [\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} & = & [\![\psi_l]\!]_{K\restriction\sigma(\psi_l)} \\ & \overset{\text{IH}}{=} & [\![p_l]\!]_{K'} \\ & = & [\![\widehat{\psi_i}]\!]_{K'} \\ & = & [\![p_i]\!]_{K'}. \end{array}$$

- If $\psi_i = \mathbf{revoke}(\psi_j, \psi_k)\psi_l$, then $[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} = [\![p_i]\!]_{K'}$ can be proven in analogy to the previous case.

- Assume $\psi_i = \langle\pi\rangle\psi_j$. By induction hypothesis $[\![\psi_k]\!]_{K\restriction\sigma(\psi_k)} = [\![p_k]\!]_{K'}$ for each $k \in [n]$ such that $\psi_k?$ occurs as a test program in π . Moreover, by induction hypothesis, $[\![\psi_j]\!]_{K\restriction\sigma(\psi_j)} = [\![p_j]\!]_{K'}$. Also recall that by definition of K and Remark 4.7 we have $[\![a]\!]_{K\restriction\sigma} = [\![a]\!]_K = [\![a]\!]_{K'} \cup [\![\overline{a}]\!]_{K'} = [\![T(a)]\!]_{K'}$. Since moreover T is defined homomorphic on $\cup$, on $\circ$, and on $*$ and since $T(\psi_k?) = p_k?$ for test programs $\psi_k?$ it is clear that

$$\begin{array}{rcl} [\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} & = & [\![\langle\pi\rangle\psi_j]\!]_{K\restriction\sigma(\psi_i)} \\ & \overset{\text{IH}}{=} & [\![T(\pi)p_j]\!]_{K'} \\ & = & [\![\widehat{\psi_i}]\!]_{K'} \\ & = & [\![p_i]\!]_{K'}. \end{array}$$

- If $\psi_i = \mathbf{perm}(\pi)\psi_j$, then we can prove $[\![\psi_i]\!]_{K\restriction\sigma(\psi_i)} = [\![p_i]\!]_{K'}$ similarly as the case $\mathbf{fperm}$ below.

- Assume $\psi_i = \mathbf{fperm}(\pi)\psi_j$, $\sigma = \sigma(\psi_i) = (\theta_1, \theta_1', \mathrm{op}_1) \cdots (\theta_k, \theta_k', \mathrm{op}_k)$ and P_σ is the policy set of $K \restriction \sigma$. By induction hypothesis for every $l \in [n]$ such that either $\psi_l?$ occurs as a test program in π or ψ_l occurs in σ we have $[\![\psi_l]\!]_{K \restriction \sigma(\psi_l)} = [\![p_l]\!]_{K'}$. Moreover, by induction hypothesis, $[\![\psi_j]\!]_{K \restriction \sigma(\psi_j)} = [\![p_j]\!]_{K'}$. Also recall that by definition of K we have $[\![a]\!]_K \setminus P = [\![\overline{a}]\!]_{K'}$ for each $a \in \mathsf{A}$. Thus, by Lemma 4.8 one can verify that for each $a \in \mathsf{A}$ we have

$$[\![a]\!]_{K \restriction \sigma} \setminus P_\sigma \;=\; [\![\overline{a}^{U_\sigma} \cup \bigcup_{b \in \{a, \overline{a}\}} \bigcup_{l \in M_\sigma} p(\theta_l)? \circ b^{U_\sigma \cap [l,k]} \circ p(\theta_l')?]\!]_{K'}.$$

Hence $[\![a]\!]_{K \restriction \sigma \setminus P_\sigma} = [\![T^\exists(a)]\!]_{K'}$ for each $a \in \mathsf{A}$. From the definition of $T^\exists$ it can easily be seen that $[\![T^\exists(\pi)]\!]_{K'}$ equals

$$\{(x, y) \in W \times W \mid \exists \text{ non-permitted } \pi\text{-path from } x \text{ to } y \text{ in } K \restriction \sigma\}.$$

Hence we obtain

$$\begin{aligned} [\![\psi_i]\!]_{K \restriction \sigma(\psi_i)} &= [\![\mathbf{fperm}(\pi)\psi_j]\!]_{K \restriction \sigma(\psi_i)} \\ &\overset{\mathrm{IH}}{=} [\![\neg\langle T^\exists(\pi)\rangle p_j]\!]_{K'} \\ &= [\![\widehat{\psi_i}]\!]_{K'} \\ &= [\![p_i]\!]_{K'}. \end{aligned}$$ ■

By combining Lemma 4.9 and Lemma 4.10, we can deduce the following theorem.

Theorem 4.11 *There exists a satisfiability preserving translation from* $\mathrm{DLP}^+_{\mathrm{dyn}}$ *to* PDL$\oplus[\mathbb{A}]$ *that is computable in polynomial time.* □

4.4 PDL$\oplus[\mathbb{A}]$ is in EXP

In this section, our goal is to prove that satisfiability of PDL$\oplus[\mathbb{A}]$ is in EXP. So for the rest of this section, fix some PDL$\oplus[\mathbb{A}]$ formula φ_0 over atomic propositions P and over atomic programs A. To decide satisfiability of φ_0, we translate φ_0 into a two-way alternating parity tree automaton over infinite trees (TWAPTA) $\mathcal{T}(\varphi_0)$ that accepts exactly the set of so called *Hintikka trees* for φ_0. Roughly speaking, Hintikka trees for φ_0 summarize relevant information of models of φ_0. So satisfiability of φ_0 reduces to non-emptiness of the language accepted by $\mathcal{T}(\varphi_0)$. For the formal definition of two-way alternating parity tree automata, we refer the reader to Section 3.3. We follow a similar approach as in [80].

Although, in the following, negations may occur in front of non-atomic formulas, we implictly assume without loss of generality that for every occurring PDL$\oplus[\mathbb{A}]$ formula negations occur only in front of atomic propositions. This can be achieved by introducing the operators $\wedge$ and $[\,]$ and by abbreviating $\neg(\psi_1 \vee \psi_2)$ by $\neg\psi_1 \wedge \neg\psi_2$ and $\neg\langle\pi\rangle\psi$ by $[\pi]\neg\psi$. Thus, we identify $\neg\neg\psi$ with ψ. Recall that we denote formulas of the kind $\langle\pi\rangle\psi$ diamond formulas and formulas of the kind $[\pi]\psi$ box formulas. For every PDL$\oplus[\mathbb{A}]$ program π we can associate a regular language $L(\pi)$ over the alphabet $\Sigma(\pi)$, where the latter consists of the basic programs and the test programs that occur in π. Moreover, we assume that the programs π that occur in φ_0 are given by NFAs $\mathcal{A}(\pi)$ over the alphabet $\Sigma(\pi)$, i.e. $L(\mathcal{A}(\pi)) = L(\pi)$. The size of such formulas is defined as usual, but with $|\pi|$ replaced by $|\mathcal{A}(\pi)|$ for each program π. Moreover, if K is a Kripke structure and $w = w_1 \cdots w_n$ with $w_i \in \Sigma(\mathcal{A}(\pi))$ for all $i \in [n]$, then $[\![w]\!]_K = [\![w_1]\!]_K \circ \cdots \circ [\![w_n]\!]_K$.

Let γ be either a basic program, a test program, or a formula. Then the set subf(γ) of subformulas of γ is inductively defined as follows:

- subf$(a) = \emptyset$ for each $a \in \mathbb{A}$,
- subf$(p) = \{p\}$ and subf$(\neg p) = \{\neg p, p\}$ for each $p \in \mathbb{P}$,
- subf$(\psi?) =$ subf(ψ),
- subf$(\oplus(\alpha, \psi_1, \psi_2)) =$ subf$(\alpha) \cup$ subf$(\psi_1) \cup$ subf(ψ_2),
- subf$(\psi_1 \text{ op } \psi_2) = \{\psi_1 \text{ op } \psi_2\} \cup$ subf$(\psi_1) \cup$ subf(ψ_2) for each op $\in \{\vee, \wedge\}$,
- if $\psi = \langle \mathcal{A} \rangle \theta$ or $\psi = [\mathcal{A}]\theta$, then subf$(\psi) = \{\psi\} \cup \bigcup_{\alpha \in \Sigma(\mathcal{A})}$ subf$(\alpha) \cup$ subf(θ)

Next, we introduce the *closure* of a formula ψ analogously as in [80, 122, 36].

Definition 4.12 *The* closure $\text{cl}(\psi)$ *of a PDL⊕[$\mathbb{A}$] formula ψ is the smallest set that satisfies the following five conditions,*

- $\psi \in \text{cl}(\psi)$,
- *if $\theta \in$ subf(χ) for some $\chi \in \text{cl}(\psi)$, then $\theta \in \text{cl}(\psi)$,*
- *if $\theta \in \text{cl}(\psi)$, then $\neg\theta \in \text{cl}(\psi)$,*
- *if $\langle \mathcal{A} \rangle \theta \in \text{cl}(\psi)$, then for all $q \in Q(\mathcal{A})$ we have $\langle \mathcal{A}_q \rangle \theta \in \text{cl}(\psi)$, and*
- *if $[\mathcal{A}]\theta \in \text{cl}(\psi)$, then for all $q \in Q(\mathcal{A})$ we have $[\mathcal{A}_q]\theta \in \text{cl}(\psi)$.* □

It is straightforward to verify, that the size of $\text{cl}(\psi)$ is polynomial in the size of ψ. Let us now introduce Hintikka sets. Roughly speaking, Hintikka sets are subsets of $\text{cl}(\psi)$ that worlds can satisfy at all.

Definition 4.13 *A subset $M \subseteq \text{cl}(\psi)$ is a* Hintikka set for a PDL⊕[$\mathbb{A}$] formula ψ*, if the following five closure properties are satisfied,*

(C1) if $\psi_1 \wedge \psi_2 \in M$, then $\psi_1, \psi_2 \in M$,

(C2) if $\psi_1 \vee \psi_2 \in M$, then $\psi_1 \in M$ or $\psi_2 \in M$,

(C3) $\theta \in M$ if and only if $\neg\theta \notin M$ for all $\theta \in \text{cl}(\psi)$,

(C4) if $[\mathcal{A}]\theta \in M$ and $q_0(\mathcal{A}) \in F(\mathcal{A})$, then $\theta \in M$, and

(C5) if $[\mathcal{A}]\theta \in M$, then for all $q \in Q(\mathcal{A})$ and all $\chi? \in \Sigma(\mathcal{A})$ with $q_0(\mathcal{A}) \xrightarrow{\chi?}_{\mathcal{A}} q$, we have $\neg\chi \in M$ or $[\mathcal{A}_q]\theta \in M$. □

Recall that φ_0 is the PDL$\oplus[\mathbb{A}]$ formula we want to decide satisfiability for. Let $\mathcal{B}$ denote the set of all basic programs that occur in some diamond formula or in some box formula from $\mathrm{cl}(\varphi_0)$. For each $\alpha \in \mathcal{B}$ let $\mathrm{At}(\alpha)$ denote the underlying atomic program, formally $\mathrm{At}(\alpha) = a$ if $\alpha = a \in \mathsf{A}$ and $\mathrm{At}(\alpha) = \mathrm{At}(\beta)$ if $\alpha = \oplus(\beta, \psi_1, \psi_2)$. For handling nestings of applied $\oplus$-operators in basic programs, we introduce, for basic programs $\alpha \in \mathcal{B}$, an appropriate notion for whether α holds between two subsets of $\mathrm{cl}(\varphi_0)$.

Definition 4.14 *For all $\alpha \in \mathcal{B}$ and all subsets $S, T \subseteq \mathrm{cl}(\varphi_0)$, we define the relation $(S,T) \models \alpha$ inductively by the syntactic structure of α as follows:*

- *$(S,T) \models a$ for all subsets $S, T \subseteq \mathrm{cl}(\varphi_0)$ and all $a \in \mathsf{A}$.*
- *If $\alpha = \oplus(\beta, \psi_1, \psi_2)$, then $(S,T) \models \alpha$ whenever $(S,T) \models \beta$ and ($\psi_1 \in S$ or $\psi_2 \in T$).* □

Let us introduce infinite trees and infinite paths in them. Let Γ be a node labeling set and let $k \in \mathbb{N}$. Then a *Γ-labeled k-tree* is a mapping $T : [k]^* \to \Gamma$. An *infinite path of T* is a mapping $\tau : \mathbb{N}_+ \to [k]$.

Fix an enumeration $\psi_1, \ldots, \psi_k$ of all diamond formulas in $\mathrm{cl}(\varphi_0)$, i.e. $\psi_i = \langle\mathcal{A}\rangle\theta$ for some NFA $\mathcal{A}$ and some formula θ. Let us now define a concrete node labeling set Γ that we will comment on below in more detail. Define the labeling set

$$\Gamma \quad = \quad 2^{\mathrm{cl}(\varphi_0)} \times (\mathcal{B} \uplus \{\bot\}) \times [0,k].$$

Intuitively, each model K of φ_0 with world set W corresponds to some Γ-labeled k-tree T that contains the necessary information about how diamond formulas are satisfied in K. We can think of T as additionally assigning each node $u \in [k]^*$ some world x of K. The first component of $T(u)$ contains exactly the set of formulas of $\mathrm{cl}(\varphi_0)$ that x satisfies. The second component of $T(u)$ either contains the information by which witnessing basic program the world x is connected with its parent or whether this node is not connected with its parent (then it equals $\bot$). If the third component of $T(u)$ contains the information $i \in [k]$, then u has the obligation, in its subtree, to show that the diamond formula ψ_i is true in x. But if the third component of $T(u)$ equals 0, then u is a witness that the obligation of some diamond formula $\psi_i = \langle\mathcal{A}\rangle\theta \in \mathrm{cl}(\varphi_0)$ in some ancestor $v \in [k]^*$ of u has just been realized by u - more precisely, let $y \in W$ be the world that is assigned to v, then $x \in [\![\theta]\!]_K$ and $(y,x) \in [\![w]\!]_K$ for some $w \in L(\mathcal{A})$.

For every $\gamma \in \Gamma$ and every $j \in \{1,2,3\}$, let γ_j denote the j-th component of γ. Before defining what a Hintikka tree is, let us first, for Γ-labeled k-trees, introduce a notion of local consistency of the labeling $T(u) \in \Gamma$ of nodes $u \in [k]^*$. For a string w over some alphabet Σ let $\mathrm{Occ}(w) \subseteq \Sigma$ denote the set of those letters that occur in w.

Definition 4.15 *Let $T : [k]^* \to \Gamma$ be a Γ-labeled k-tree, and let $u \in [k]^*$ be a node. We say that T is* locally consistent *in u, if the following two conditions hold:*

(L1) For every diamond formula $\psi_i = \langle\mathcal{A}\rangle\theta \in T(u)_1$ there exists some sequence of test programs $w \in (\Sigma(\mathcal{A}) \setminus \mathcal{B})^*$ with $\mathrm{Occ}(w) \subseteq T(u)_1$ and for some state $q' \in Q(\mathcal{A})$ we have $q_0(\mathcal{A}) \xrightarrow{w}_{\mathcal{A}} q'$ such that at least one of the following two conditions holds,

a)
- $q' \in F(\mathcal{A})$,
- $\theta \in T(u)_1$,
- $T(ui)_2 = \bot$, *and*
- $T(ui)_3 = 0$, *or*

b) for some basic program $\alpha \in \Sigma(\mathcal{A}) \cap \mathcal{B}$ and some state $q \in Q(\mathcal{A})$ we have
- $q' \xrightarrow{\alpha}_{\mathcal{A}} q$,
- $\psi_j = \langle \mathcal{A}_q \rangle \theta \in T(ui)_1$,
- $T(ui)_2 = \alpha$,
- $T(ui)_3 = j$, *and*
- $(T(u)_1, T(ui)_1) \models \alpha$. □

(L2) For every box formula $[\mathcal{A}]\theta \in T(u)_1$ and every transition $q_0(\mathcal{A}) \xrightarrow{\alpha}_{\mathcal{A}} q$, where $\alpha \in \Sigma(\mathcal{A}) \cap \mathcal{B}$ is some basic program and $q \in Q(\mathcal{A})$ is some state, the following implication holds for all $j \in [k]$ with $T(uj)_2 = \beta \in \mathcal{B}$ such that $\mathrm{At}(\alpha) = \mathrm{At}(\beta)$,

$$(T(u)_1, T(uj)_1) \models \alpha \qquad \Rightarrow \qquad [\mathcal{A}_q]\theta \in T(uj)_1.$$

Let us summarize the intention of local consistency of Γ-labeled k-trees in a node $u \in [k]^*$. As indicated above, condition (L1) ensures that whenever u obliges to prove that some diamond formula ψ_i holds, then this proof is provided in the subtree of u: Either we directly prove that ψ_i holds in u, this corresponds to (L1) a), or we delay this proof by obliging an appropriate successor of u to prove an appropriate diamond formula ψ_j, this corresponds to (L1) b). Condition (L2) ensures that if a box formula ψ holds in some node u , then certain successors of u must realize certain other box formulas in order to avoid inconsistencies to the fact that u satisfies ψ.

We call a diamond formula ψ_i *infinitely delaying in a node* $u \in [k]^*$ of some Γ-labeled k-tree $T : [k]^* \to \Gamma$, if there exists an infinite path $\tau : \mathbb{N}_+ \to [k]$ such that $\tau(1) = i$ and $T(u\tau(1) \cdots \tau(n))_3 = \tau(n+1)$ for all $n \geq 1$.

Definition 4.16 *A* Hintikka tree for φ_0 *is a Γ-labeled k-tree $T : [k]^* \to \Gamma$ such that the following four conditions hold,*

(H1) $\varphi_0 \in T(\varepsilon)_1$,

(H2) $T(u)_1$ is a Hintikka set for φ_0 for all $u \in [k]^*$,

(H3) T is locally consistent in all $u \in [k]^*$, and

(H4) the diamond formula ψ_i is not infinitely delaying in u for all $u \in [k]^*$ and all $i \in [k]$ such that $\psi_i \in T(u)_1$, . □

The following lemma relates the existence of models of φ_0 with the existence of Hintikka trees for φ_0.

Lemma 4.17 *The formula φ_0 is satisfiable if and only if there exists a Hintikka tree for φ_0.* □

PROOF
"Only-if": Assume φ_0 is satisfiable. Then, there exists a Kripke structure $K = (W, \{\rightarrow_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$ and some world $z_0 \in W$ with $z_0 \in [\![\varphi_0]\!]_K$. Recall that $\psi_1, \ldots, \psi_k$ is an enumeration of all diamond formulas of $\mathrm{cl}(\varphi_0)$.
For each NFA $\mathcal{A}$ that appears in some diamond formula $\langle\mathcal{A}\rangle\theta \in \mathrm{cl}(\varphi_0)$, fix an arbitrary total order $\sqsubset_{\mathcal{A}}$ on $\Sigma(\mathcal{A})$. We extend the total order $\sqsubset_{\mathcal{A}}$ to words over $\Sigma(\mathcal{A})$ as the corresponding length lexicographic order with respect to $\sqsubset_{\mathcal{A}}$. Formally, for all $u, v \in \Sigma(\mathcal{A})^*$, we define $u \sqsubset_{\mathcal{A}} v$ if and only if either (i) $|u| < |v|$, or (ii) $|u| = |v|$ and for some $w \in \Sigma(\mathcal{A})^*$ and some $b, c \in \Sigma(\mathcal{A})$ with $b \sqsubset_{\mathcal{A}} c$ we have that wb is a prefix of u and wc is a prefix of v.
For each diamond formula $\langle\mathcal{A}\rangle\theta \in \mathrm{cl}(\varphi_0)$ or box formula $[\mathcal{A}]\theta \in \mathrm{cl}(\varphi_0)$ and $b \in \Sigma(\mathcal{A})$ define

$$\stackrel{b}{\Rightarrow}_{\mathcal{A}} \quad = \quad \{((x,q),(y,q')) \in (W \times Q) \times (W \times Q) \mid q \xrightarrow{b}_{\mathcal{A}} q' \text{ and } (x,y) \in [\![b]\!]_K\}.$$

Let us for each $\psi_i = \langle A\rangle\theta$, and each $x \in [\![\psi_i]\!]_K$ fix the $\sqsubset_{\mathcal{A}}$-minimal sequence $\sigma_{x,i} = b_1 \cdots b_n$ ($n \geq 0$) together with some *involved worlds* $x_0, x_1, \ldots, x_n \in W$ and some *involved states* $q_0, q_1, \ldots, q_n \in Q(\mathcal{A})$ such that

$$(x_0, q_0(\mathcal{A})) = (x_0, q_0) \stackrel{b_1}{\Rightarrow} (x_1, q_1) \cdots (x_{n-1}, q_{n-1}) \stackrel{b_n}{\Rightarrow} (x_n, q_n) \in [\![\theta]\!]_K \times F(\mathcal{A}).$$

Recall that $\mathcal{B}$ denotes the set of all basic programs that occur in φ_0 and $\Gamma = 2^{\mathrm{cl}(\varphi_0)} \times (\mathcal{B} \uplus \{\bot\}) \times [0,k]$. Our goal is to construct a Hintikka tree for φ_0. For this, inductively, by the length of the elements from $[k]^*$, we simultaneously define a Γ-labeled k-tree $T : [k]^* \rightarrow \Gamma$ and a mapping $\Psi : [k]^* \rightarrow W$. We set

$$\begin{array}{lcl} \Psi(\varepsilon) & = & z_0, \\ T(\varepsilon)_1 & = & \{\psi \in \mathrm{cl}(\varphi_0) \mid z_0 \in [\![\psi]\!]_K\}, \\ T(\varepsilon)_2 & = & \bot, \text{ and} \\ T(\varepsilon)_3 & = & 0. \end{array}$$

The definition of T and Ψ will have the property that for all $u \in [k]^*$ we have

$$T(u)_1 \quad = \quad \{\psi \in \mathrm{cl}(\varphi_0) \mid \Psi(u) \in [\![\psi]\!]_K\}. \tag{4.7}$$

Now assume that $\Psi(u)$ and $T(u)$ have been defined for $u \in [k]^*$ but yet $\Psi(ui)$ and $T(ui)$ have not been defined for all $i \in [k]$. Assume $\Psi(u) = x$ for some $x \in W$. Then, for each $i \in [k]$, we do the following:

- If $\psi_i = \langle\mathcal{A}\rangle\theta \in T(u)_1$, then we know by equation (4.7) that $x \in [\![\psi_i]\!]_K$. We distinguish two cases:
  - $\sigma_{x,i}$ is either the empty word or only consists of test programs. Then, we put

$$\begin{array}{lcl} \Psi(ui) & = & z_0, \\ T(ui)_1 & = & \{\psi \in \mathrm{cl}(\varphi_0) \mid z_0 \in [\![\psi]\!]_K\}, \\ T(ui)_2 & = & \bot, \text{ and} \\ T(ui)_3 & = & 0. \end{array}$$

– $\sigma_{x,i} = b_1 \cdots b_n$ contains a basic program. Let $q_0, q_1, \ldots, q_n \in Q(\mathcal{A})$ be the involved states and let $x_0, x_1, \ldots, x_n \in W$ be the involved worlds for $\sigma_{x,i}$. Let $l \in [n]$ be minimal such that $b_l = \alpha \in \mathcal{B}$ is a basic program and let $\psi_j = \langle \mathcal{A}_{q_l} \rangle \theta$. Then, we put

$$\begin{array}{rcl} \Psi(ui) & = & x_l, \\ T(ui)_1 & = & \{\psi \in \mathrm{cl}(\varphi_0) \mid x_l \in [\![\psi]\!]_K\}, \\ T(ui)_2 & = & \alpha, \text{ and} \\ T(ui)_3 & = & j. \end{array}$$

- If $\psi_i = \langle \mathcal{A} \rangle \theta \notin T(u)_1$, then we put

$$\begin{array}{rcl} \Psi(ui) & = & z_0, \\ T(ui)_1 & = & \{\psi \in \mathrm{cl}(\varphi_0) \mid z_0 \in [\![\psi]\!]_K\}, \\ T(ui)_2 & = & \bot, \text{ and} \\ T(ui)_3 & = & 0. \end{array}$$

Note that equation (4.7) holds for each ui ($u \in [k]^*, i \in [k]$) directly by definition of T. Before proving that T is indeed a Hintikka tree, we prove the following claim.

Claim A: For all $u \in [k]^*$ and all $i \in [k]$ such that $T(ui)_2 = \beta$ for some $\beta \in \mathcal{B}$, the following equivalence holds for all $\alpha \in \mathcal{B}$ with $\mathrm{At}(\alpha) = \mathrm{At}(\beta)$:

$$(\Psi(u), \Psi(ui)) \in [\![\alpha]\!]_K \qquad \Leftrightarrow \qquad (T(u)_1, T(ui)_1) \models \alpha.$$

Let us prove Claim A. Let $\mathrm{At}(\alpha) = \mathrm{At}(\beta) = a$ for some $a \in \mathsf{A}$. We prove Claim A by induction on the syntactic structure of α, which is possible since $\mathrm{At}(\gamma) = \mathrm{At}(\alpha)$ for all basic subprograms γ of α.
*B*ase. Assume $\alpha = a$ for some $a \in \mathbb{A}$. Since $T(ui)_2 = \beta$, we have $(\Psi(u), \Psi(ui)) \in [\![\beta]\!]_K$ by definition of T. Since α is atomic, it is clear that also $(\Psi(u), \Psi(ui)) \in [\![\alpha]\!]_K$. Too, $(T(u)_1, T(ui)_1) \models \alpha$ holds since α is atomic.
Step. Now assume $\alpha = \oplus(\gamma, \theta_1, \theta_2)$. By induction hypothesis Claim A holds for γ. We have

$$\begin{array}{cl} & (\Psi(u), \Psi(ui)) \in [\![\alpha]\!]_K \\ \Leftrightarrow & (\Psi(u), \Psi(ui)) \in \oplus([\![\gamma]\!]_K, [\![\theta_1]\!]_K, [\![\theta_2]\!]_K) \\ \Leftrightarrow & (\Psi(u), \Psi(ui) \in [\![\gamma]\!]_K \text{ and } (\Psi(u) \in [\![\theta_1]\!]_K \text{ or } \Psi(ui) \in [\![\theta_2]\!]_K) \\ \overset{\text{IH}}{\Leftrightarrow} & (T(u)_1, T(ui)_1) \models \gamma \text{ and } (\Psi(u) \in [\![\theta_1]\!]_K \text{ or } \Psi(ui) \in [\![\theta_2]\!]_K) \\ \overset{(4.7)}{\Leftrightarrow} & (T(u)_1, T(ui)_1) \models \gamma \text{ and } (\theta_1 \in T(u)_1 \text{ or } \theta_2 \in T(ui)_1) \\ \overset{\text{Def. of} \models}{\Leftrightarrow} & (T(u)_1, T(ui)_1) \models \alpha. \end{array}$$

This completes the proof of Claim A.

It remains to prove that T is indeed a Hintikka tree for φ_0.
Point (H1), i.e. $\varphi_0 \in T(\varepsilon)_1$, follows directly from the definition of T. .

It follows directly from the semantics of PDL$\oplus[\mathbb{A}]$ that (H2) holds, i.e. $T(u)_1$ is a Hintikka set for φ_0 for every $u \in [k]^*$: Points (C1)-(C5) of Definition 4.13 are easily verified. .

Let us prove (H3) of Definition 4.16, i.e. T is locally consistent in all $u \in [k]^*$. Let $\Psi(u) = x$ for some $x \in W$.

Firstly, let us prove that (L1) of Definition 4.15 holds. So let $i \in [k]$ and assume $\psi_i = \langle \mathcal{A} \rangle \theta \in T(u)_1$. Then, by equation (4.7), we have $x \in [\![\psi_i]\!]_K$. Assume $\sigma_{x,i} = b_1 \cdots b_n$, let $q_0, q_1, \ldots, q_n \in Q(\mathcal{A})$ be its involved states, and let $x_0, x_1, \ldots, x_n \in W$ be its involved worlds. Furthermore, let $w = \theta_1? \cdots \theta_l?$ $(l \geq 0)$ be the longest prefix of $\sigma_{x,i}$ that only consists of test programs. Thus, we have $q_0(\mathcal{A}) \xrightarrow{w}_{\mathcal{A}} q_l$ and $x \in [\![\theta_j]\!]_K$ for all $j \in [l]$. Hence, $\mathrm{Occ}(w) = \{\theta_1, \ldots, \theta_l\} \subseteq T(u)_1$ by equation (4.7). We distinguish the following two cases:

a) If $\sigma_{x,i}$ is the empty word or consists of test programs only, then $l = n$, $x_l = x$, and $q_l \in F(\mathcal{A})$. Since the latter implies $x \in [\![\theta]\!]_K$, we get $[\![\theta]\!] \in T(u)_1$ by equation (4.7). By construction of T, we have $T(ui)_2 = \bot$ and $T(ui)_3 = 0$. Altogether, this shows (L1) a) of Definition 4.15.

b) Assume $\sigma_{x,i}$ contains at least one basic program. This implies $b_{l+1} = \alpha$ for some $\alpha \in \mathcal{B}$ and we have $q_l \xrightarrow{\alpha}_{\mathcal{A}} q_{l+1}$. Let $\psi_j = \langle \mathcal{A}_{q_{l+1}} \rangle \theta$. Clearly, we have $x_{l+1} \in [\![\psi_j]\!]_K$ and due to construction of T, we have $\Psi(ui) = x_{l+1}$. Thus, equation (4.7) yields $\psi_j \in T(ui)_1$. By construction of T, we also have $T(ui)_2 = \alpha$ and $T(ui)_3 = j$. Since $(\Psi(u), \Psi(ui)) \in [\![\alpha]\!]_K$ and $T(ui)_2 = \alpha$, Claim A implies $(T(u)_1, T(ui)_1) \models \alpha$. Altogether, this shows (L1) b) of Definition 4.15.

Let us now prove that (L2) holds. So assume $[\mathcal{A}]\theta \in T(u)_1$ and $q_0(\mathcal{A}) \xrightarrow{\alpha}_{\mathcal{A}} q$ for some $\alpha \in \Sigma(\mathcal{A}) \cap \mathcal{B}$ and some $q \in Q(\mathcal{A})$. Furthermore, assume, by contradiction, that for some $j \in [k]$ with $T(uj)_2 = \beta \in \mathcal{B}$ and $\mathrm{At}(\alpha) = \mathrm{At}(\beta)$, we have $(T(u)_1, T(uj)_1) \models \alpha$ and $[\mathcal{A}_q]\theta \notin T(uj)_1$. By equation (4.7), $[\mathcal{A}_q]\theta \notin T(uj)_1$ is equivalent to $\Psi(uj) \in [\![\langle \mathcal{A}_q \rangle \neg\theta]\!]_K$. Since $(T(u)_1, T(uj)_1) \models \alpha$, it follows $(\Psi(u), \Psi(uj)) \in [\![\alpha]\!]_K$ by Claim A. Hence

$$(q_0(\mathcal{A}), \Psi(u)) \stackrel{\alpha}{\Rightarrow}_{\mathcal{A}} (q, \Psi(uj)) \text{ and } \Psi(uj) \in [\![\langle \mathcal{A}_q \rangle \neg\theta]\!]_K$$

and therefore $\Psi(u) \in [\![\langle \mathcal{A} \rangle \neg\theta]\!]_K$ contradicting $\Psi(u) \in [\![[\mathcal{A}]\theta]\!]_K$ and thus also $[\mathcal{A}]\theta \in T(u)_1$ according to equation (4.7).

Before finally proving (H4), the following claim holds:

Claim B: For all $u \in [k]^*$, $i, j \in [k]$ with $\psi_i \in T(u)_1$ and $T(ui)_3 = j$ we have

$$|\sigma_{\Psi(u),i}| > |\sigma_{\Psi(ui),j}|.$$

Claim B follows directly by construction of T and by definition of $\sigma_{\Psi(u),i}$ and $\sigma_{\Psi(ui),j}$.

To prove (H4), assume by contradiction, that for some $i \in [k]$ the diamond formula ψ_i is infinitely delaying in some $u \in [k]^*$. Furthermore, let $\Psi(u) = x$ for some $x \in W$. But by Claim B the existence of an infinite path $\tau : \mathbb{N}_+ \to [k]$ with $\tau(1) = i$ and $T(u\tau(1) \cdots \tau(n))_3 = \tau(n+1)$ for all $n \geq 1$ clearly contradicts the finiteness of $|\sigma_{x,i}|$.

"If": Let $T : [k]^* \to \Gamma$ be a Hintikka tree for φ_0. We will prove that there exists a model for φ_0. Define the Kripke structure $K = (W, \{\to_a \mid a \in \mathsf{A}\}, \{W_p \mid p \in \mathsf{P}\})$ as follows,

$$
\begin{array}{rcl}
W & = & [k]^*, \\
\to_a & = & \{(u, ui) \mid u \in [k]^*, i \in [k], T(ui)_2 = \alpha \in \mathcal{B} \text{ and } \mathrm{At}(\alpha) = a\} \text{ for each } a \in \mathsf{A}, \text{ and} \\
W_p & = & \{u \in [k]^* \mid p \in T(u)_1\} \text{ for each } p \in \mathsf{P}.
\end{array}
$$

Let $\prec \subseteq \mathrm{cl}(\varphi_0) \times \mathrm{cl}(\varphi_0)$ be defined as follows,

$$\prec \quad = \quad \{(\psi, \theta) \in \mathrm{cl}(\varphi_0) \times \mathrm{cl}(\varphi_0) \mid \psi, \neg\psi \neq \theta \wedge \psi \in \mathrm{cl}(\theta)\}.$$

Note that $\prec$ is a strict partial order on $\mathrm{cl}(\varphi_0)$ and the minimal elements are those formulas ψ such that either $\psi = p$ or $\psi = \neg p$ for some $p \in \mathsf{P}$. Our goal is to prove that for all $\psi \in \mathrm{cl}(\varphi_0)$ and for all $u \in [k]^*$ the following implication holds:

$$\psi \in T(u)_1 \quad \Rightarrow \quad u \in [\![\psi]\!]_K. \tag{4.8}$$

Having shown implication (4.8) immediately proves that K is a model of φ_0, since, by (H1), we know $\varphi_0 \in T(\varepsilon)_1$ and thus, by implication (4.8), we have $\varepsilon \in [\![\varphi_0]\!]_K$. We prove implication (4.8) by induction on $\prec$.
Base. Assume ψ is minimal with respect to $\prec$. Hence $\psi = p$ or $\psi = \neg p$ for some $p \in \mathsf{P}$. It follows directly from definition of W_p and from (C3) of Definition 4.13 that $\psi \in T(u)_1$ implies $u \in [\![\psi]\!]_K$.
Step.

- If $\psi = \theta_1 \wedge \theta_2$ or $\psi = \theta_1 \vee \theta_2$, then $\psi \in T(u)_1 \Rightarrow u \in [\![\psi]\!]_K$ follows immediately from (C1) and (C2) of Definition 4.13 and from induction hypothesis.

- Let $\psi = \langle \mathcal{A} \rangle \theta$ or $\psi = [\mathcal{A}]\theta$. Now the following claim holds.

  Claim: Let $u \in [k]^*$ and $i \in [k]$ and assume $u \to_a ui$ for some $a \in \mathsf{A}$. Then, for all basic programs $\alpha \in \Sigma(\mathcal{A}) \cap \mathcal{B}$ such that $\mathrm{At}(\alpha) = a$, we have

  $$(T(u)_1, T(ui)_1) \models \alpha \quad \Leftrightarrow \quad (u, ui) \in [\![\alpha]\!]_K.$$

  Let us prove the above claim. Since $\mathrm{At}(\gamma) = \mathrm{At}(\alpha)$ for all basic subprograms γ of α, we can prove the claim by induction on the syntactic structure of α.
  Base. Assume α is atomic. By definition of $\models$, we have $(T(u)_1, T(ui)_1) \models \alpha$. By assumption, we also have $(u, ui) \in [\![\alpha]\!]_K$.
  Step. Let $\alpha = \oplus(\gamma, \theta_1, \theta_2)$ and assume the claim is true for γ. Let us first prove that the following two statements are equivalent:

  (i) $\theta_1 \in T(u)_1$ or $\theta_2 \in T(ui)_1$.
  (ii) $u \in [\![\theta_1]\!]_K$ or $ui \in [\![\theta_2]\!]_K$.

The implication from (i) to (ii) follows directly from the outer induction hypothesis. To prove that (ii) implies (i), assume by contradiction and without loss of generality that $u \in [\![\theta_1]\!]_K$ and $\theta_1 \notin T(u)_1$. Then, by (C3) of Definition 4.13, we have $\neg\theta_1 \in T(u)_1$. By the outer induction hypothesis we have $u \notin [\![\theta_1]\!]_K$, contradicting the assumption. Thus, we get

$$\begin{array}{cl}
 & (T(u)_1, T(ui)_1) \models \alpha \\
\overset{\text{Def. of} \models}{\Leftrightarrow} & (T(u)_1, T(ui)_1) \models \gamma \text{ and } (\theta_1 \in T(u)_1 \text{ or } \theta_2 \in T(ui)_1) \\
\overset{\text{(i)} \Leftrightarrow \text{(ii)}}{\Leftrightarrow} & (T(u)_1, T(ui)_1) \models \gamma \text{ and } (u \in [\![\theta_1]\!]_K \text{ or } ui \in [\![\theta_2]\!]_K) \\
\overset{\text{inner IH}}{\Leftrightarrow} & (u, ui) \in [\![\gamma]\!]_K \text{ and } (u \in [\![\theta_1]\!]_K \text{ or } ui \in [\![\theta_2]\!]_K) \\
\Leftrightarrow & (u, ui) \in \oplus([\![\gamma]\!]_K, [\![\theta_1]\!]_K, [\![\theta_2]\!]_K) \\
\Leftrightarrow & (u, ui) \in [\![\alpha]\!]_K .
\end{array}$$

This completes the proof of the above claim.

Let us first prove that if $\psi = \langle\mathcal{A}\rangle\theta$, then $\psi \in T(u)_1$ implies $u \in [\![\psi]\!]_K$. Let $\psi = \psi_i$ for some $i \in [k]$ and assume that $\psi_i \in T(u)_1$. Since T is a Hintikka tree for φ_0, and thus the diamond formula ψ_i is not infinitely delaying in u, there exists a finite sequence $\tau = j_1 \cdots j_n \in [k]^*$, $n \geq 1$, such that $j_1 = i$, $T(u\tau)_3 = 0$, and $T(u\tau^{(m)})_3 = j_{m+1}$ for all $m \in [n-1]$. Let $\Sigma_? = \Sigma(\mathcal{A}) \setminus \mathcal{B}$ denote the set of all test programs that occur in $\Sigma(\mathcal{A})$. Firstly, since $T(u\tau^{(m)})_3 = j_{m+1} \neq 0$ and T is locally consistent in $u\tau^{(m)}$ for each $m \in [n-1]$, condition (L1) b) of Definition 4.15 holds in $u\tau^{(m)}$ for each $m \in [n-1]$. Secondly, since $T(u\tau)_3 = 0$ and T is also locally consistent in $u\tau$, we know that condition (L1) a) of Definition 4.15 holds in $u\tau$. Taking these facts together, we conclude that there exist states $q_1, q_1', \ldots, q_n, q_n' \in Q(\mathcal{A})$, words $w_1, \ldots, w_{n-1} \in \Sigma_?^*$, and basic programs $\alpha_1, \ldots, \alpha_{n-1} \in \Sigma(\mathcal{A}) \cap \mathcal{B}$ such that the following seven conditions are satisfied:

(1) $q_1 = q_0(\mathcal{A})$,
(2) $\mathrm{Occ}(w_m) \subseteq T(u\tau^{(m-1)})_1$ for all $m \in [n-1]$,
(3) $q_m \xrightarrow{w_m}_{\mathcal{A}} q_m'$ for all $m \in [n]$,
(4) $q_m' \xrightarrow{\alpha_m}_{\mathcal{A}} q_{m+1}$ for all $m \in [n-1]$,
(5) $(T(u\tau^{(m-1)})_1, T(u\tau^{(m)})_1) \models \alpha_m$ for all $m \in [n-1]$,
(6) $q_n' \in F(\mathcal{A})$, and
(7) $\theta \in T(u\tau^{(n-1)})_1$.

Taking the transitions from Points (3) and (4) together, we obtain the run

$$q_0(\mathcal{A}) = q_1 \xrightarrow{w_1}_{\mathcal{A}} q_1' \xrightarrow{\alpha_1}_{\mathcal{A}} q_2 \quad \cdots \quad q_{n-1} \xrightarrow{w_{n-1}}_{\mathcal{A}} q_{n-1}' \xrightarrow{\alpha_{n-1}}_{\mathcal{A}} q_n \xrightarrow{w_n}_{\mathcal{A}} q_n'.$$

Moreover, Point (5) is equivalent to $(u\tau^{(m-1)}, u\tau^{(m)}) \in [\![\alpha_m]\!]_K$ for all $m \in [n-1]$ by the above claim. Point (2) implies $(u\tau^{(m-1)}, u\tau^{(m-1)}) \in [\![w_m]\!]_K$ for all $m \in [n-1]$ by induction hypothesis. Altogether, we obtain $(u, u\tau^{(n-1)}) \in [\![w_1\alpha_1 \cdots w_{n-1}\alpha_{n-1}w_n]\!]_K$.

Moreover, Point (7) implies $u\tau^{(n-1)} \in [\![\theta]\!]_K$ by induction hypothesis. Since $q_1 = q_0(\mathcal{A})$ and $q'_n \in F(\mathcal{A})$ by Points (1) and (6), we conclude $u \in [\![\langle\mathcal{A}\rangle\theta]\!]_K$.

It remains to prove that $[\mathcal{A}]\theta \in T(u)_1$ implies $u \in [\![[\mathcal{A}]\theta]\!]_K$. For this, assume $[\mathcal{A}]\theta \in T(u)_1$. Let $u_0, u_1, \ldots, u_n \in [k]^*$ be worlds, let $q_0, \ldots, q_n \in Q(\mathcal{A})$ be states, and let $b_1, \ldots, b_n \in \Sigma(\mathcal{A})$ be such that

$$(u, q_0(\mathcal{A})) = (u_0, q_0) \overset{b_1}{\Rightarrow}_{\mathcal{A}} (u_1, q_1) \quad \cdots \quad (u_{n-1}, q_{n-1}) \overset{b_n}{\Rightarrow}_{\mathcal{A}} (u_n, q_n) \in W \times F(\mathcal{A}).$$

We will prove that then $u_n \in [\![\theta]\!]_K$. It suffices to prove that $[\mathcal{A}_{q_m}]\theta \in T(u_m)_1$ for each $m \in [0, n]$ since $q_n \in F(\mathcal{A})$ and $[\mathcal{A}_{q_n}]\theta \in T(u_n)_1$ implies $\theta \in T(u_n)_1$ by (C4) of Definition 4.13 and thus $u_n \in [\![\theta]\!]_K$ by induction hypothesis. We prove $[\mathcal{A}_{q_m}]\theta \in T(u_m)_1$ by induction on m. The induction base, i.e. $[\mathcal{A}_{q_0}]\theta \in T(u_0)_1$, holds by assumption. For the induction step assume $[\mathcal{A}_{q_m}]\theta \in T(u_m)_1$ already holds for $m \in [0, n-1]$. To prove $[\mathcal{A}_{q_{m+1}}]\theta \in T(u_{m+1})_1$, we distinguish the following two cases:

- b_{m+1} is a test program, i.e. $b_{m+1} = \chi? \in \Sigma(\mathcal{A}) \setminus \mathcal{B}$. Hence $u_m = u_{m+1}$ and $u_{m+1} \in [\![\chi]\!]_K$. The outer induction hypothesis and (C3) of Definition 4.13 imply $\neg\chi \notin T(u_{m+1})_1$. By the inner induction hypothesis it holds $[\mathcal{A}_{q_m}]\theta \in T(u_m)_1$. Since $q_m \xrightarrow{\chi?}_{\mathcal{A}} q_{m+1}$ we have $[\mathcal{A}_{q_{m+1}}]\theta \in T(u_{m+1})_1$ by (C5) of Definition 4.13.
- b_{m+1} is a basic program, i.e. $b_{j+1} = \alpha \in \Sigma(\mathcal{A}) \cap \mathcal{B}$. Since $(q_m, u_m) \overset{\alpha}{\Rightarrow}_{\mathcal{A}} (q_{m+1}, u_{m+1})$ there exists some $j \in [k]$ such that $u_{m+1} = u_m j$ and there exists some $\beta \in \mathcal{B}$ such that $T(u_{m+1})_2 = \beta$ and $\mathrm{At}(\alpha) = \mathrm{At}(\beta)$ by definition of K. Applying the above claim yields $(T(u_m)_1, T(u_{m+1})_1) \models \alpha$. Since $[\mathcal{A}_{q_m}]\theta \in T(u_m)_1$ holds by the inner induction hypothesis, we get $[\mathcal{A}_{q_{m+1}}]\theta \in T(u_m j)_1 = T(u_{m+1})_1$ by (L2) of Definition 4.15. ■

Recall that TWAPTAs were introduced in Definition 3.10 of Section 3.3. The following lemma is straightforward and left to the reader.

Lemma 4.18 *From φ_0 one can compute in time exponential in $|\varphi_0|$ a TWAPTA $\mathcal{T}(\varphi_0) = (S, \delta, s_0, \mathrm{Acc})$ over Γ-labeled k-trees such that $L(\mathcal{T}(\varphi_0))$ equals the set of all Hintikka trees for φ_0. Moreover, $|\mathcal{T}(\varphi_0)| \in poly(|\varphi_0|)$, $i(\mathcal{T}) \in \mathcal{O}(1)$, and $|\delta| \in \exp(|\varphi_0|)$.* □

Note that actually a Büchi acceptance condition suffices to check whether some diamond formula is not infinitely delaying in some node. Clearly, we need alternation to check $(S, T) \models \alpha$ for subsets $S, T \subseteq \mathrm{cl}(\varphi_0)$ and $\alpha \in \mathcal{B}$ and thus for checking local consistency, more precisely (H3) of Definition 4.16. By Theorem 3.11 and Lemma 4.18, the existence of a Hintikka tree for φ_0 can be decided in time exponential in $|\varphi_0|$. By finally applying Lemma 4.17, we get the desired theorem.

Theorem 4.19 *Satisfiability for PDL$\oplus[\mathbb{A}]$ is in* EXP. □

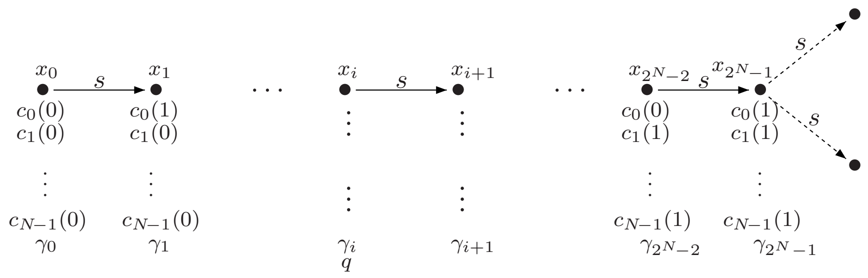


Figure 4.2: Representation of a configuration of $\mathcal{M}$.

4.5 PDL⊕ is 2EXP-complete

In this section, we prove that satisfiability in PDL⊕ is complete for 2EXP.

Theorem 4.20 *Satisfiability in PDL⊕ is* 2EXP*-complete.* □

PROOF The 2EXP upper bound follows easily from a translation from PDL⊕ into PDL with an exponential blowup in formula size (see also Remark 4.6) and by the EXP upper bound for satisfiability in PDL (see Theorem 2.6). The proof of the lower bound is an adaption of the 2EXP lower bound for IPDL from [72] with a little technical wit that allows to simulate intersection via the $\oplus$ operator. Fix some ATM $\mathcal{M} = (Q, \Gamma, \Sigma, q_0, \delta, \Box)$ with a 2EXP-hard membership problem, that, on an input $w = w_1 \cdots w_n \in \Sigma^*$ where $w_i \in \Sigma$ for each $i \in [n]$, uses at most $2^{p(n)}$ space for some polynomial p. We will give a translation of w and $\mathcal{M}$ into a PDL⊕ formula $\varphi = \varphi(\mathcal{M}, w)$ such that $w \in L(\mathcal{M})$ if and only if φ is satisfiable. Moreover the translation can be computed in time polynomial in $n = |w|$.

Let $N = p(n)$ and assume that Q and Γ are disjoint. A *configuration* of $\mathcal{M}$ is a word from the language $\bigcup_{i=0}^{2^N-1} \Gamma^i Q \Gamma^{2^N-i}$. Let $u = \gamma_0 \cdots \gamma_{i-1} q \gamma_i \cdots \gamma_{2^N-1}$ be a configuration of $\mathcal{M}$, where $i \in [0, 2^N - 1]$, $q \in Q$, and $\gamma_j \in \Gamma$ for each $j \in [0, 2^N - 1]$. Figure 5.1 illustrates how we will represent u.

Each world x_j ($j \in [0, 2^N - 1]$) contains the content γ_j of cell j of u together with a binary representation of j that we comment on below in more detail. In Figure 5.1 the symbols below each world denote the atomic propositions that hold in x_j. The atomic propositions $c_l(0)$ and $c_l(1)$ (where $l \in [0, N-1]$), represent the binary encoding of each position j. More precisely each x_j satisfies either $c_l(0)$ or $c_l(1)$ ($l \in [0, N-1]$) such that moreover

$$j = \sum_{l=0}^{N-1} 2^l \cdot b_l \qquad \text{where } b_l = \begin{cases} 0 & \text{if } x_j \text{ satisfies } c_l(0) \\ 1 & \text{if } x_j \text{ satisfies } c_l(1) \end{cases}.$$

Furthermore for each $j \in [0, 2^N - 1]$ there exists exactly one $\gamma \in \Gamma$ such that the atomic proposition γ holds in x_j, denoting that the content of cell j is γ. Furthermore, there exists exactly one $j \in [0, 2^N - 1]$ and exactly one $q \in Q$ such that proposition q holds in x_j, denoting that in configuration u the ATM $\mathcal{M}$ is currently in state q and scans cell j. World x_j is connected to world x_{j+1} via the atomic program s for each $j \in [0, 2^N - 1]$. Finally x_{2^N-1} may be connected to the first cell of one or both representations of successor configurations of u via s. Formally, the formula φ will be over the atomic propostions $\mathsf{P} = \{c_j(0), c_j(1) \mid j \in [0, N-1]\} \cup Q \cup \Gamma$ and over the singleton atomic program set $\mathsf{A} = \{s\}$.

Let us define some auxiliary formulas and programs. First, for every subset $B \subseteq Q \cup \Gamma$, we define φ_B as

$$\varphi_B \quad = \quad \bigvee_{b \in B} b.$$

The formula φ_{sound} checks if the atomic propositions in the current world describe sound representations of cells of configurations of $\mathcal{M}$. More precisely, for each $i \in [0, N-1]$ exactly one of the atomic propositions $c_i(0)$ and $c_i(1)$ is true. Moreover, for exactly one $\gamma \in \Gamma$ we have that γ holds and for at most one $q \in Q$ the atomic proposition q holds,

$$\begin{aligned}\varphi_{\text{sound}} \quad = \quad & \bigwedge_{i \in [0,N-1]} (c_i(0) \leftrightarrow \neg c_i(1)) \wedge \bigvee_{\gamma \in \Gamma} \left(\gamma \wedge \bigwedge_{\gamma' \in \Gamma : \gamma' \neq \gamma} \neg \gamma' \right) \\ & \wedge \bigwedge_{q \in Q} \left(q \rightarrow \bigwedge_{q' \in Q : q' \neq q} \neg q' \right).\end{aligned}$$

The formula φ_{first} (resp. φ_{last}) checks if the current world represents the first (resp. last) cell of some configuration:

$$\begin{aligned}\varphi_{\text{first}} \quad &= \quad \bigwedge_{i \in [0,N-1]} c_i(0) \\ \varphi_{\text{last}} \quad &= \quad \bigwedge_{i \in [0,N-1]} c_i(1)\end{aligned}$$

Formula φ_{+1} checks that the current world represents a cell at position $j \in [0, 2^N - 2]$ and all successor worlds represent some cell at position $j+1$,

$$\begin{aligned}\varphi_{+1} \quad = \quad & \bigvee_{i \in [0,N-1]} \Bigg[c_i(0) \wedge [s] c_i(1) \wedge \bigwedge_{k \in [0,i-1]} c_k(1) \wedge [s] c_k(0) \\ & \wedge \bigwedge_{k \in [i+1,N-1]} \left(\bigwedge_{b \in \{0,1\}} c_k(b) \rightarrow [s] c_k(b) \right) \Bigg].\end{aligned}$$

The formula φ_{glob} guarantees that all reachable worlds describe a sound representation of cells. Moreover, it is checked that all reachable worlds that are not last cells have some successor cell and furthermore all successors of last cells are first cells. Finally, the formula φ_{glob} prohibits that a configuration contains two cells that both contain some state symbol

$$\begin{aligned}\varphi_{\text{glob}} \quad = \quad & [s^*] \Big(\varphi_{\text{sound}} \wedge (\neg \varphi_{\text{last}} \rightarrow (\varphi_{+1} \wedge \langle s \rangle \mathtt{true})) \wedge (\varphi_{\text{last}} \rightarrow [s] \varphi_{\text{first}}) \\ & \wedge \big((\varphi_Q \wedge \neg \varphi_{\text{last}}) \rightarrow \neg \langle (s \circ \neg \varphi_{\text{first}}?)^+ \rangle \varphi_Q \big) \, \mathtt{true} \Big).\end{aligned}$$

Recall that $w = w_1 \cdots w_n$ is the input of $\mathcal{M}$. Next, we give a formula φ_w that guarantees that the configuration $q_0 w \Box^{2^N - n}$ is the first configuration seen,

$$\varphi_w \quad = \quad \varphi_{\text{first}} \wedge q_0 \wedge w_1 \\ \wedge [s] \left(w_2 \wedge [s] \left(w_3 \quad \cdots \quad \wedge \; [s](w_n \wedge [(s \circ \neg\varphi_{\text{first}}?)^+]\Box) \right) \cdots \right).$$

Let us now, for each formula ψ, define a program $\pi(\psi)$ that relates the current non-first (resp. first) cell c to some non-first (resp. the first) cell c' of some successor configuration and moreover, if c and c' are non-first cells, checks if the predecessor cell of c' satisfies the formula ψ,

$$\pi(\psi) \quad = \quad (\neg\varphi_{\text{first}}? \circ (s \circ \neg\varphi_{\text{first}}?)^* \circ s \circ \varphi_{\text{first}}? \circ (s \circ \neg\varphi_{\text{first}}?)^* \circ \psi? \circ s \circ \neg\varphi_{\text{first}}?) \\ \cup (\varphi_{\text{first}}? \circ (s \circ \neg\varphi_{\text{first}}?)^* \circ s \circ \varphi_{\text{first}}?).$$

Putting $\alpha_{-1}(\psi) = \pi(\psi)$, we define

$$\alpha_i(\psi) \quad = \quad \oplus\left(\oplus\left(\alpha_{i-1}(\psi), c_i(0), c_i(1)\right), c_i(1), c_i(0)\right) \qquad \text{for each } i \in [0, N-1].$$

Eventually, we define

$$\text{match}(\psi) = \alpha_{N-1}(\psi).$$

Since will enforce that each (reachable) world in a model of φ satisfies exactly one of the atomic propositions $c_i(0)$ and $c_i(1)$, the program $\text{match}(\psi)$ relates worlds that are related by $\pi(\psi)$ and that additionally agree on the same atomic propositions from $\{c_i(0), c_i(1) \mid i \in [0, N-1]\}$, i.e. the following equality holds:

$$[\![\text{match}(\psi)]\!]_K \quad = \quad \bigcap_{i=0}^{N-1} \bigcup_{b \in \{0,1\}} [\![c_i(b)? \circ \pi(\psi) \circ c_i(b)?]\!]_K.$$

Hence $\text{match}(\psi)$ precisely connects worlds that represent cells c and c' of consecutive configurations at *same positions* and in case c and c' both represent non-first cells, the predecessor of c' satisfies ψ.

Now, for every move $\mu = (q, \gamma, d) \in$ Moves of $\mathcal{M}$, where $q \in Q, \gamma \in \Gamma$, and $d \in \{\leftarrow, \rightarrow\}$, we define a program π_μ that relates the read-write cell of the current configuration u with some cell of some successor configuration that is obtained from u by making the move μ.
Firstly, assume $\mu = (q, \gamma, \rightarrow)$ for some $q \in Q$ and some $\gamma \in \Gamma$. As we move the read-write head of $\mathcal{M}$ to the right, we have to make sure that we do not scan the last cell of our current configuration and define

$$\pi_\mu \quad = \quad \neg\varphi_{\text{last}}? \circ \text{match}(\texttt{true}) \circ \gamma? \circ s \circ q?.$$

Secondly, assume $\mu = (q, \gamma, \leftarrow)$ for some $q \in Q$ and some $\gamma \in \Gamma$. Since we move the read-write head to the left, we have to make sure that we do not scan the first cell of our current configuration and define

$$\pi_\mu \quad = \quad \neg\varphi_{\text{first}}? \circ \text{match}(q) \circ \gamma?.$$

Let us now describe the formula φ_δ that assures that every model of φ contains a representation of a computation tree of $\mathcal{M}$.

$$\begin{aligned}\varphi_\delta \quad = \quad [s^*]\Bigg(& \bigwedge_{\substack{q\in Q_\exists,\gamma\in\Gamma:\\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \Bigg((q\wedge\gamma) \rightarrow \bigvee_{i\in\{1,2\}} \langle\pi_{\mu_i}\rangle\texttt{true}\Bigg)\\ & \wedge \bigwedge_{\substack{q\in Q_\forall,\gamma\in\Gamma:\\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \Bigg((q\wedge\gamma) \rightarrow \bigwedge_{i\in\{1,2\}} \langle\pi_{\mu_i}\rangle\texttt{true})\Bigg)\\ & \wedge \bigwedge_{\gamma\in\Gamma} ((\gamma\wedge\neg p_Q) \rightarrow \neg\langle\text{match}(\texttt{true})\rangle\neg\gamma)\Bigg)\end{aligned}$$

The formula φ_{acc} ensures that there does not exist a configuration in an existential state $q \in Q_\exists$ and scanning a symbol γ such that $\delta(q,\gamma) = \bot$ and that moreover there does not exist an infinite s-path. This corresponds to the existence of an acceptance tree of $\mathcal{M}$ on w,

$$\varphi_{\text{acc}} \quad = \quad \neg[s^*]\langle s\rangle\texttt{true} \wedge [s^*] \bigwedge_{\substack{q\in Q_\exists,\gamma\in\Gamma:\\ \delta(q,\gamma)=\bot}} \neg(q\wedge\gamma).$$

Eventually, we define

$$\varphi \quad = \quad \varphi_{\text{glob}} \;\wedge\; \varphi_w \;\wedge\; \varphi_\delta \;\wedge\; \varphi_{\text{acc}}.$$

It is tedious but not difficult to verify that $w \in L(\mathcal{M})$ if and only if φ is satisfiable. ■

4.6 A 2EXP-hard variant of DLP$^+_{\text{dyn}}$

The aim of this section is to point out that a mere modification of the semantics of DLP$^+_{\text{dyn}}$ yields a logic which satisfiability is 2EXP hard for.

Assume that we changed the semantics of **perm** and **fperm** formulas of DLP$^+_{\text{dyn}}$ as follows, where $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\}, P)$ is some extended Kripke structure:

$$[\![\mathbf{perm}(\pi)\varphi]\!]_K \quad = \quad \{x \in W \mid \exists y \in [\![\varphi]\!]_K\ \exists\pi\text{-path from } x \text{ to } y \text{ such that } (x,y) \in P\}$$

$$[\![\mathbf{fperm}(\pi)\varphi]\!]_K \quad = \quad \{x \in W \mid \forall y \in [\![\varphi]\!]_K\ \exists\pi\text{-path from } x \text{ to } y \text{ implies } (x,y) \in P\}$$

Let us call this modified logic DLP$^{++}_{\text{dyn}}$. Let us summarize the difference between DLP$^{++}_{\text{dyn}}$ and DLP$^+_{\text{dyn}}$. In DLP$^+_{\text{dyn}}$, for a world x satisfying $\mathbf{perm}(\pi)\varphi$ there has to be a π-path to some world y satisfying φ such that each non-test transition on this π-path lies in the policy set P. In DLP$^{++}_{\text{dyn}}$ however, we require that $(x,y) \in P$ and no further requirement on transitions on the π-path is imposed.

Analogously, in DLP$^+_{\text{dyn}}$ a world x satisfying the formula $\mathbf{fperm}(\pi)\varphi$ has the property that all non-test transitions on any π-path from x to any world y satisfying φ lie in P. In DLP$^{++}_{\text{dyn}}$

however, we require that $(x, y) \in P$ and no further requirements on transitions on the π-paths are imposed.

The following proposition shows that this change of semantics indeed increases the complexity of satisfiability. We sketch a 2EXP-hardness proof for satisfiability in $\mathrm{DLP}^{++}_{\mathrm{dyn}}$. It relies on the 2EXP-hardness proof of satisfiability in PDL$\oplus$ (Theorem 4.20).

Proposition 4.21 *Satisfiability in* $\mathit{DLP}^{++}_{\mathrm{dyn}}$ *is* 2EXP*-hard.* □

PROOF (SKETCH) We recycle the proof of Theorem 4.20. Starting with an ATM $\mathcal{M}$ and an input w we constructed a formula $\varphi = \varphi(\mathcal{M}, w)$ which was defined as follows:

$$\varphi \quad = \quad \varphi_{\mathrm{glob}} \wedge \varphi_w \wedge \varphi_\delta \wedge \varphi_{\mathrm{acc}}.$$

Note that only the formula φ_δ contains the $\oplus$ operator. We will give a $\mathrm{DLP}^{++}_{\mathrm{dyn}}$ formula φ' that is equi-satisfiable to φ. The formula φ' is defined as follows,

$$\varphi' \quad = \quad \varphi_{\mathrm{glob}} \wedge \varphi_w \wedge \varphi'_\delta \wedge \varphi_{\mathrm{acc}},$$

with φ'_δ defined as,

$$\begin{aligned}
\varphi'_\delta \quad = \quad & \mathbf{grant}(\mathtt{true}, \mathtt{true}) && (4.9)\\
& \mathbf{revoke}(c_0(0), c_0(1))\ \mathbf{revoke}(c_0(1), c_0(0)) && (4.10)\\
& \mathbf{revoke}(c_1(0), c_1(1))\ \mathbf{revoke}(c_1(1), c_1(0)) && (4.11)\\
& \vdots \\
& \mathbf{revoke}(c_{N-1}(0), c_{N-1}(1))\ \mathbf{revoke}(c_{N-1}(1), c_{N-1}(0)) && (4.12)\\
& [s^*]\Bigg(\bigwedge_{\substack{q \in Q_\exists, \gamma \in \Gamma:\\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \Big((q \wedge \gamma) \to \bigvee_{i \in \{1,2\}} \mathbf{perm}(\pi'_{\mu_i})\mathtt{true} \Big) \\
& \quad \wedge \bigwedge_{\substack{q \in Q_\forall, \gamma \in \Gamma:\\ \delta(q,\gamma)=(\mu_1,\mu_2)}} \Big((q \wedge \gamma) \to \bigwedge_{i \in \{1,2\}} \mathbf{perm}(\pi'_{\mu_i})\mathtt{true} \Big) \\
& \quad \wedge \bigwedge_{\gamma \in \Gamma} \big((\gamma \wedge \neg p_Q) \to \neg \mathbf{perm}(\mathrm{match}'(\mathtt{true}))\neg\gamma \big) \Bigg)
\end{aligned}$$

for appropriately adjusted programs π'_{μ_i} and match' that no longer have to contain the $\oplus$ operator to relate cells at same positions since the **grant** and **revoke** operations from (4.9) to (4.12) assure that precisely those pairs of worlds are added to the policy set that satisfy the same atomic propositions from $\{c_i(0), c_i(1) \mid i \in [0, N-1]\}$ and hence represent cells at same positions. ■

We conjecture that the previous lower bound given in Proposition 4.21 is tight and satisfiability in $\mathrm{DLP}^{++}_{\mathrm{dyn}}$ is 2EXP-complete.

4.7 Open problems

Non-regular and fixed point extensions

Recall that our translation from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to $\mathrm{PDL}\oplus[\mathbb{A}]$ exploits the idea that we define for each subformula ψ of a given $\mathrm{DLP}^+_{\mathrm{dyn}}$ formula some $\mathrm{PDL}\oplus[\mathbb{A}]$ formula that roughly defines those worlds of $K \upharpoonright \sigma(\psi)$ that satisfy ψ.

Firstly, one can extend $\mathrm{DLP}^+_{\mathrm{dyn}}$ by allowing non-regular programs. A prominent example for a non-regular extension of PDL that still has a decidable satisfiability problem [75] are visibly pushdown languages (VPL) introduced in [4]. The author conjectures that satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ where programs are given by visibly pushdown automata [4] is not harder than the VPL extension of PDL studied in [75], namely 2EXP. The reason for this is that it should not be hard, using our approach, to define VPL programs that correspond to the set of permitted and non-permitted paths by an appropriate manipulation of visibly pushdown automata.

Secondly, our method to decide satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is not directly applicable when enhancing $\mathrm{DLP}^+_{\mathrm{dyn}}$ by a fixed point formalism in the style of the modal μ-calculus since subformulas behave more complex in the presence of fixed points. Yet, it is rather straightforward to prove that the resulting logic is still invariant under bisimulation. However the decidability and complexity status of the satisfiability problem is worth investigating.

Finite sequence semantics

Recall that DLP is the fragment of $\mathrm{DLP}_{\mathrm{dyn}}$, where the operators $\mathbf{grant}$ and $\mathbf{revoke}$ do not occur. Orignally [117], the semantics of DLP was defined differently. Atomic programs were not interpreted as binary relations over the underlying world set W but rather as a set of finite sequences over W. Let us call this notion of semantics the *finite sequence semantics*. Clearly the semantics we introduced for $\mathrm{DLP}^+_{\mathrm{dyn}}$ is a special case of the finite sequence semantics since each binary relation over the world set is a set of sequences of length one. As shown in Lemma 2.1 of [24] a DLP formula is satisfiable in our semantics if and only if it is satisfiable in the finite sequence semantics. However, this is no longer the case for $\mathrm{DLP}^+_{\mathrm{dyn}}$. Hence the question arises what the complexity of satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$ is when formulas are interpreted by the finite sequence semantics.

Sabotage Modal Logic

As shown in [76] satisfiability in SML is undecidable. However, the decidability status is unclear if we restrict ourselves to SML formulas over a singleton set of atomic programs.

Let us shortly introduce world-deleting version SML' of SML. In SML' the sabotage operator is not parametrized by an atomic program, i.e. we only have the world deleting sabotage operator $\langle - \rangle$ with the following semantics, where $K = (W, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{W_p \mid p \in \mathbb{P}\})$ is a Kripke structure:

$$[\![\langle - \rangle \varphi]\!]_K \quad = \quad \{x \in W \mid \exists y \in W \setminus \{x\} : x \in [\![\varphi]\!]_{K \setminus y}\},$$

where for each world $y \in W$ the Kripke structure $K \setminus y$ is defined as follows:

$$K \setminus y \quad = \quad (W \setminus \{y\}, \{\rightarrow_a \cap (W \setminus \{y\})^2 \mid a \in \mathbb{A}\}, \{W_p \setminus \{y\} \mid p \in \mathbb{P}\}).$$

So far, the decidability status of satisfiability in SML' is unknown.

5 Infinite state model checking of PDL and fragments

In this chapter we investigate the complexity of infinite state model checking of PDL and fragments of it. As stated in Section 2.5, besides prefix-recognizable systems, we follow Mayr's classification of infinite state systems. However, in Section 5.1 we show, via a reduction from the undecidable model checking problem of EF over Petri nets, that already over basic parallel processes (a subclass of Petri nets) model checking test-free PDL becomes undecidable. In Section 5.2 we prove that over prefix-recognizable systems already reachability is hard for deterministic exponential time and thus EXP-complete. For the sake of completeness, we investigate in Section 5.3 (resp. Section 5.4) the complexity of infinite state model checking of test-free (resp. full) PDL via simple reductions from known results of infinite state model checking of EF, CTL, and the modal μ-calculus respectively. Table 5.1 summarizes the results. Section 5.5 concludes the chapter with the open problem of model checking EF over one-counter systems.

Since we are interested in infinite state model checking we restrict the considered the logics and Kripke structures not to contain any atomic propositions as already justified in Section 2.5. Since we deal with it several times in this chapter, we introduce the logic EF, a fragment of test-free PDL. Formally, formulas φ of EF are given by the following grammar, where a ranges over the countable set of atomic programs $\mathbb{A}$,

$$\varphi \quad ::= \quad \texttt{true} \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid \mathrm{EF}\varphi.$$

In the following, $\mathsf{A} \subseteq \mathbb{A}$ always denotes a finite set of atomic programs. Given a Kripke structure

<table>
<tr><th colspan="2"></th><th>BPA</th><th>PDS</th><th>PRS</th></tr>
<tr><td rowspan="3">EF, PDL$\setminus$?</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></td></tr>
<tr><td>combined</td><td></td></tr>
</table>

Table 5.1: Complexity of infinite state model checking (test-free) PDL.

$K = (W, \{\to_a \mid a \in \mathsf{A}\})$ for each EF formula φ, the semantics $[\![\varphi]\!]_K \subseteq W$ is defined as follows,

$$\begin{aligned}
[\![\mathtt{true}]\!]_K &= W, \\
[\![\neg\varphi]\!]_K &= W \setminus [\![\varphi]\!]_K, \\
[\![\varphi_1 \vee \varphi_2]\!]_K &= [\![\varphi_1]\!]_K \cup [\![\varphi_2]\!]_K, \\
[\![\langle a\rangle\varphi]\!]_K &= \{x \in W \mid \exists y \in [\![\varphi]\!]_K : x \to_a y\} \quad \text{for each } a \in \mathsf{A}, \text{ and} \\
[\![\mathrm{EF}\varphi]\!]_K &= \{x \in W \mid \exists y \in [\![\varphi]\!]_K : (x, y) \in (\cup_{a\in\mathsf{A}} \to_a)^*\}.
\end{aligned}$$

Clearly, EF is a fragment of test-free PDL since formulas of the kind EFφ can be translated to $\langle \mathsf{A}^*\rangle\varphi'$, where φ' is inductively the translation for φ.

5.1 Undecidability of test-free PDL over communication-free nets

In this section we give an undecidability proof of model checking test-free PDL over communication-free nets by reducing the undecidable model checking problem of EF over Petri nets to it.

A *Petri net* over a finite set of atomic programs $\mathsf{A} \subseteq \mathbb{A}$ is a tuple $\mathcal{N} = (S, T, \gamma, \rho)$, where S is a finite set of *places*, T is a finite set of *transitions*, $\gamma : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a *weight function*, and $\rho : T \to \mathsf{A}$ labels each transition with an atomic program. A *marking* is a mapping $M : S \to \mathbb{N}$. The underlying Kripke structure is defined as $K(\mathcal{N}) = (\mathbb{N}^S, \{\to_a \mid a \in \mathsf{A}\})$, where for each $a \in \mathsf{A}$ we define

$$\begin{aligned}
\to_a \quad = \quad \{(M_1, M_2) \in \mathbb{N}^S \times \mathbb{N}^S \mid \exists t \in \rho^{-1}(a) \forall s \in S : M_1(s) \geq \gamma(s,t) \\
\wedge\, M_2(s) = M_1(s) + \gamma(t,s) - \gamma(s,t)\}.
\end{aligned}$$

The *size* of a Petri net is defined as $|\mathcal{N}| = |\mathsf{A}| + \sum_{s\in S, t\in T} \gamma(s,t) + \gamma(t,s)$. Petri nets can also be described by transition systems which are generated by Mayr's process rewriting [84], where only parallel composition is allowed on both the left and the right hand side of each rewriting rule. A subclass of Petri nets, called *basic parallel processes* is defined by allowing parallel composition only on the right hand side of each rewriting rule. This class is also known as the class of communication-free nets, which we introduce next. A *communication-free net* is a Petri net $\mathcal{N} = (S, T, \gamma, \rho)$ such that for all $t \in T$ there exists a unique $s \in S$ with $\gamma(s, t) = 1$ such that $\gamma(s', t) = 0$ for all $s' \neq s$.

Let us state the undecidability of EF model checking over Petri nets.

Theorem 5.1 ([29, 30]) *There exists a fixed EF formula φ such that the following problem is undecidable:*
INPUT: A Petri net $\mathcal{N}$ and an initial marking M.
QUESTION: $M \in [\![\varphi]\!]_{K(\mathcal{N})}$? □

In the following corollary, we reduce EF model checking over Petri nets to test-free PDL model checking over communication-free nets. Note that the resulting test-free PDL formula of our reduction needs to be part of the input. Too, it is worth mentioning that the latter corollary stands in contrast with the PSPACE-completeness of the combined complexity of EF model checking over communication-free nets as shown in [85].

Corollary 5.2 *The following problem is undecidable:*
INPUT: A communication-free net $\mathcal{N}$, an initial marking M, and a test-free PDL formula φ.
QUESTION: $M \in [\![\varphi]\!]_{K(\mathcal{N})}$? □

Proof Let φ be the fixed EF formula of Theorem 5.1 over some set of atomic programs A, let $\mathcal{N} = (S, T, \gamma, \rho)$ be a Petri net and let $M \in \mathbb{N}^S$ be an initial marking. We prove that one can compute a test-free PDL formula φ', a communication-free net $\mathcal{N}' = (S', T', \gamma', \rho')$, and a marking M' such that

$$M \in [\![\varphi]\!]_{K(\mathcal{N})} \quad \Leftrightarrow \quad M' \in [\![\varphi']\!]_{K(\mathcal{N}')}.$$

Firstly, we enhance each transition $t \in T$ with several new transitions, namely with $(s, t) \in T'$ for each place $s \in S$. The idea is that firing a transition $t \in T$ in $\mathcal{N}$ corresponds in $\mathcal{N}'$ to first firing each transition (s, t) (which takes from place s precisely one weight unit) exactly $\gamma(s, t)$ times and then, by firing t, ensuring that each place s gets added $\gamma(t, s)$ more weight units. Our formula φ' actually takes care that the latter is accomplished. Thus, $T' = T \cup (S \times T)$. In addition to S, we introduce a fresh place s_0, i.e. $S' = S \uplus \{s_0\}$. The place s_0 ensures that in $\mathcal{N}'$ firing transitions from $t \in T$ is possible. The set of atomic programs A' that $\mathcal{N}'$ and φ' are defined over equals T'. We define ρ' to be the identity function, i.e. $\rho'(t') = t'$ for each $t' \in T'$. We define the weight function γ' as follows,

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

and

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

for each $s' \in S'$ and each $t' \in T'$. Observe that $\mathcal{N}'$ is a communication-free net. Our initial marking M' is defined as $M'(s) = M(s)$ for each $s \in S$ and $M'(s_0) = 1$. Let us assume that $S = \{s_1, \ldots, s_n\}$. We define the test-free PDL formula φ' to emerge from the EF formula φ by replacing each occurrence of a modality "$\langle a \rangle$" by

$$\left\langle \bigcup_{t \in \rho^{-1}(a)} \left(\bigcirc_{i=1}^{n} \underbrace{(s_i, t) \circ \cdots \circ (s_i, t)}_{\gamma(s_i,t) \text{ times}} \right) \circ t \right\rangle$$

and each occurrence of "EF" by

$$\left\langle \left(\bigcup_{a \in \mathsf{A}, t \in \rho^{-1}(a)} \left(\bigcirc_{i=1}^{n} \underbrace{(s_i, t) \circ \cdots \circ (s_i, t)}_{\gamma(s_i,t) \text{ times}} \right) \circ t \right)^{*} \right\rangle.$$

It is not difficult to see that we have

$$M \in [\![\varphi]\!]_{K(\mathcal{N})} \quad \Leftrightarrow \quad M' \in [\![\varphi']\!]_{K(\mathcal{N}')}.$$

5.2 EXP-hardness of reachability on prefix-recognizable graphs

In this section we no longer talk about Kripke structures but about graphs, which formally can be seen as Kripke structures over a singleton set of atomic programs.

The *reachability problem* asks, given some graph G and two vertices u and v of G, if there exists a path from u to v in G. The *alternating reachability problem* asks, given two vertices u and v of some game graph, i.e. a graph whose vertices are partitioned into those for player *Eve* and player *Adam*, if *Eve* has a winning strategy in the reachability game from u to v. On finite graphs it is well known that the reachability problem is complete for NL, whereas for the alternating reachability problem the complexity jumps to P-completeness.

It is natural to ask what the complexity status of the (alternating) reachability problem is when the input graph is potentially infinite but finitely presented. On *pushdown graphs* reachability can be solved in deterministic polynomial time [14]. A matching lower bound can easily be seen to hold already over BPAs via a reduction from the non-emptiness problem of context-free grammars, see also Corollary 5.7. Alternating reachability on pushdown graphs on the other hand is EXP-complete [107]. On *one-counter graphs*, i.e. transition graphs of pushdown automata over a singleton stack alphabet, reachability is NL-complete, whereupon recent results [62, 100] imply that alternating reachability is PSPACE-complete. Reachability on order-n pushdown graphs (which generalize pushdown graphs by allowing a nested stack structure) is $(n-1)$EXP-complete [28], whereas alternating reachability is nEXP-complete [51, 16]. Summarizing, for the above mentioned established classes of infinite graphs, there is an exponential complexity jump from reachability to alternating reachability. A comparable phenomenon can be observed on *regular ground tree rewriting graphs*: Reachability is decidable but already deciding whether all paths starting in some source vertex eventually reach some vertex of a regular set of vertices is undecidable [74]. Even worse, on the more general class of *automatic graphs*, where reachability is Σ^0_1-complete [9], alternating reachability is at least Π^1_1-hard following from a recent result [69], where it is shown inter alia that deciding if an automatic successor tree has an infinite path is Σ^1_1-complete. Prefix-recognizable graphs, introduced by Caucal [18], form a natural class of infinite graphs with numerous characterizations and a decidable monadic second order theory; [73] gives a nice overview. For instance, they can be characterized as the transition graphs of pushdown automata with ε-transitions [105] and strictly generalize (order-1) pushdown graphs. We prove that already the reachability problem on prefix-recognizable graphs is EXP-hard. Since solving parity games on prefix-recognizable is in EXP [15], it immediately follows that alternating reachability is in EXP too. Thus surprisingly, both reachability *and* alternating reachability on prefix-recognizable graphs turn out to be EXP-complete. So alternation on prefix-recognizable graphs comes for free.

As a consequence of our lower bound, the data complexity of model checking EF and thus of test-free PDL over prefix-recognizable systems is EXP-hard, see also Table 5.1. As we are interested in proving a lower bound, a *prefix-recognizable graph* G is simply a prefix-recognizable relation, i.e. $G = \cup_{i=1}^{k}(U_i \times V_i)W_i$, where $k \in \mathbb{N}$ and $U_1, V_1, W_1, \ldots, U_k, V_k, W_k$ are regular languages over some finite alphabet Σ. We assume that G is represented by a collection of triples of regular expressions $(\alpha_i, \beta_i, \gamma_i)_{1 \leq i \leq k}$ where $L(\alpha_i) = U_i$, $L(\beta_i) = V_i$, $L(\gamma_i) = W_i$. The *size* of G is defined as $|G| = \sum_{i=1}^{k} |\alpha_i| + |\beta_i| + |\gamma_i|$.

The *reachability problem* (on prefix-recognizable graphs) is stated as follows:

INPUT: A prefix-recognizable graph G and $u, v \in \Sigma^*$.
QUESTION: $(u, v) \in G^+$?

We proceed with the EXP lower bound for reachability on prefix-recognizable graphs. For this, fix some $p(n)$-space bounded ATM $\mathcal{M} = (Q, \Gamma, \Sigma_{\mathcal{M}}, q_0, \delta, \Box)$, where $p(n)$ is some polynomial, with an EXP-hard membership problem. Moreover, let $w \in \Sigma^*_{\mathcal{M}}$ be an input of length n and let $N = p(n)$. Configurations of $\mathcal{M}$ will be described by the regular language

$$U \quad = \quad \bigcup_{0 \leq i < N} \Gamma^i Q \Gamma^{N-i}.$$

In the following, configurations of $\mathcal{M}$ will be confused with members of U. We will construct from w and $\mathcal{M}$ a polynomial time computable (in $n = |w|$) prefix-recognizable graph G such that

$$w \in L(\mathcal{M}) \quad \Leftrightarrow \quad (q_0 w \Box^{N-n}, \varepsilon) \in G^+.$$

We may assume without loss of generality that $\mathcal{M}$ never attempts to move left (resp. right) scanning the left-most (resp. right-most) tape cell. Let Moves denote the moves that occur as components in $\delta(Q, \Gamma) \setminus \{\perp\}$. The alphabet of G will be

$$\Sigma \quad = \quad Q \cup \Gamma \cup \{l, r, e\} \cup \{\mu_{i,j} \mid \mu \in \text{Moves and } i, j \in [0, N+1]\}.$$

Before defining G, let us introduce a relation $\longrightarrow \subseteq \Sigma^* \times \Sigma^*$ that describes a single step of a depth-first left-to-right traversal of acceptance trees of $\mathcal{M}$. For this, let u range over U being in state $q \in Q$, scanning a symbol $a \in \Gamma$ (i.e. $u \in \Gamma^* q a \Gamma^*$), and where $v \in \Sigma^*$ is arbitrary,

$$uv \longrightarrow v \quad \text{if } q \in Q_\forall \text{ and } \delta(q,a) = \bot, \tag{5.1}$$

$$xuv \longrightarrow v \quad \text{if } x \in \{e, r\}, \tag{5.2}$$

$$uv \longrightarrow u'luv \quad \text{if } q \in Q_\forall, \delta(q,a) = (\mu,\nu) \in \text{Moves}^2, \text{ and } u' \text{ is the } \mu\text{-successor configuration of } u, \tag{5.3}$$

$$uv \longrightarrow u'euv \quad \text{if } q \in Q_\exists, \delta(q,a) = (\mu,\nu) \in \text{Moves}^2 \text{ and } u' \text{ is the } \mu\text{- or } \nu\text{-successor configuration of } u, \tag{5.4}$$

$$luv \longrightarrow u'ruv \quad \text{if } q \in Q_\forall, \delta(q,a) = (\mu,\nu) \in \text{Moves}^2, \text{ and } u' \text{ is the } \nu\text{-successor configuration of } u. \tag{5.5}$$

Next, we prove that the relation $\longrightarrow$ is as required.

Lemma 5.3 *For every $u \in U$ the following three statements are equivalent:*

(i) $u \longrightarrow^+ \varepsilon$,

(ii) $uv \longrightarrow^+ v$ *for every* $v \in \Sigma^*$,

(iii) u *is accepting.* □

PROOF
(ii) ⇒ (i): Trivial.
(i) ⇒ (iii): We prove $u \longrightarrow^k \varepsilon$ implies that u has an acceptance tree by induction on $k \geq 1$. The case $k = 1$ is clear since we can only apply (5.1) and u has an acceptance tree by (CT a) of Defintion 2.1. Now assume $u \longrightarrow^{k+1} \varepsilon$. Assume u is universal and let $u', u'' \in U$ be the two successor configurations of u. It follows that $u \longrightarrow^{k+1} \varepsilon$ is decomposed as

$$u \overset{(5.3)}{\longrightarrow} u'lu \longrightarrow^{k_1} lu \overset{(5.5)}{\longrightarrow} u''ru \longrightarrow^{k_2} ru \overset{(5.2)}{\longrightarrow} \varepsilon,$$

where $k_1 + k_2 < k$. The latter is due to the observation that every occurrence of the letter l (resp. r) in a word $z \in \Sigma^*$ can only be rewritten by $\longrightarrow$ if l (resp. r) is the first letter of z. Moreover it follows by the definition of $\longrightarrow$ that $u' \longrightarrow^{k_1} \varepsilon$ and $u'' \longrightarrow^{k_2} \varepsilon$. Hence, by induction hypothesis both for u' and u'' there exists an acceptance tree. Thus, by (CT b) of Definition 2.1, there exists an acceptance tree for u too. The case when u is existential can be proven analogously.
(iii) ⇒ (ii): Assume u is accepting. We prove $uv \longrightarrow^+ v$ for all $v \in \Sigma^*$ by induction on the smallest height h of some acceptance tree of u. Assume $h = 0$, hence u is in some state $q \in Q_\forall$ scanning some symbol $a \in \Gamma$ and $\delta(q,a) = \bot$ by (CT a) of Definition 2.1. Thus, by applying (5.1), we obtain $uv \longrightarrow v$ for all $v \in \Sigma^*$. Now let $h > 0$. In case u is existential, there exists a successor configuration u' of u for which the smallest height of some acceptance tree for u' is strictly less than h. Hence, by induction hypothesis, we have $u'v' \longrightarrow^+ v'$ for all $v' \in \Sigma^*$. By additionally applying (5.2) and (5.4) we obtain for every $v \in \Sigma^*$

$$uv \overset{(5.4)}{\longrightarrow} u'euv \longrightarrow^+ euv \overset{(5.2)}{\longrightarrow} v.$$

Hence $uv \longrightarrow^+ v$ for all $v \in \Sigma^*$. The case when u is universal can be proven analogously. ■

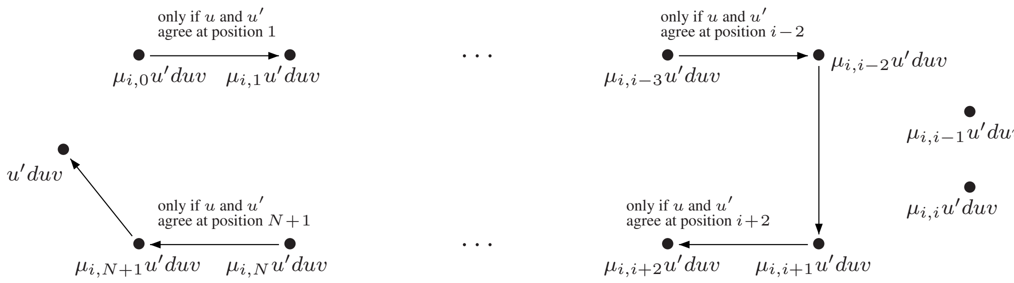


Figure 5.1: The graph $G_=$

Our prefix-recognizable graph G will be the union of several prefix-recognizable graphs we define below.
The prefix-recognizable graph G_1 corresponds to (5.1):

$$G_1 \quad = \quad \bigcup_{\substack{0\le i<N \\ q\in Q_\forall, a\in\Gamma:\delta(q,a)=\perp}} \left(\Gamma^i qa\Gamma^{N-i-1} \times \{\varepsilon\}\right)\Sigma^*$$

Similarly, the prefix-recognizable graph G_2 corresponds to (5.2):

$$G_2 \quad = \quad (\{e,r\}U \times \{\varepsilon\})\Sigma^*$$

Before defining prefix-recognizable graphs that correspond to (5.3), (5.4), and (5.5) respectively, we need an auxiliary graph $G_=$ that is often used later and has the following property: For each $\mu \in$ Moves and each $u, u' \in U$ we have $(\mu_{i,0}u'duv, u'duv) \in G_=^+$ only if u' agrees with u on all positions besides $\{i-1, i, i+1\}$, where $d \in \{l, r, e\}$ and $v \in \Sigma^*$. Figure 5.1 shows how this can be achieved. Formally, we define

$$\begin{aligned} G_= \quad = \quad & \bigcup_{\substack{1<i\le N, \\ \mu\in\text{Moves}}} \left(\{\mu_{i,i-2}\} \times \{\mu_{i,i+1}\}\right)\Sigma^* \;\cup\; \left(\mu_{i,N+1} \times \{\varepsilon\}\right)\Sigma^* \;\cup \\ & \bigcup_{\substack{0\le j<i-2 \text{ or } i+1\le j\le N \\ c\in\Gamma}} \left(\{\mu_{i,j}\} \times \{\mu_{i,j+1}\}\right)\Sigma^j c\Sigma^{N+1}c\Sigma^*. \end{aligned}$$

Let us turn to a graph G_3 that corresponds to (5.3). We make a case distinction. In case the move μ is from $Q \times \Gamma \times \{\leftarrow\}$, we realize (5.3) by

$$G_3^{\leftarrow} \quad = \quad \bigcup_{1<i\le N} \bigcup_{c\in\Gamma} \bigcup_{\substack{q\in Q_\forall, a\in\Gamma \\ \delta(q,a)=(\mu,\nu) \\ \mu=(q',b,\leftarrow)}} \left(\{\varepsilon\} \times \mu_{i,0}\Gamma^{i-2}q'cb\Gamma^{N-i}l\right)\Gamma^{i-2}cqa\Gamma^{N-i}\Sigma^*.$$

In case μ is from $Q \times \Gamma \times \{\rightarrow\}$, we realize (5.3) by

$$G_3^{\rightarrow} \quad = \quad \bigcup_{1<i\le N} \bigcup_{c\in\Gamma} \bigcup_{\substack{q\in Q_\forall, a\in\Gamma \\ \delta(q,a)=(\mu,\nu) \\ \mu=(q',b,\rightarrow)}} \left(\{\varepsilon\} \times \mu_{i,0}\Gamma^{i-2}bq'c\Gamma^{N-i}l\right)\Gamma^{i-2}qac\Gamma^{N-i}\Sigma^*.$$

Define $G_3 = G_3^{\leftarrow} \cup G_3^{\rightarrow}$. Note that both $G_3^{\rightarrow}$ and $G_3^{\leftarrow}$ allow to get from vertices uv to vertices $\mu_{i,0}u'luv$ where u' is not necessarily the μ-sucessor of u. But thanks to $G_=$ it is guaranteed that from $\mu_{i,0}u'luv$ the vertex $u'luv$ can only be reached if u' is indeed the μ-successor of u.

Analogously to (5.3), the graph G_4 that corresponds to (5.4) is the union of

$$\bigcup_{1<i\leq N}\bigcup_{c\in\Gamma}\bigcup_{\substack{q\in Q_\exists,a\in\Gamma\\ \delta(q,a)=(\nu,\nu')\\ \mu=(q',b,\leftarrow)\in\{\nu,\nu'\}}}\left(\{\varepsilon\}\times\mu_{i,0}\Gamma^{i-2}q'cb\Gamma^{N-i}e\right)\Gamma^{i-2}cqa\Gamma^{N-i}\Sigma^*$$

and

$$\bigcup_{1<i\leq N}\bigcup_{c\in\Gamma}\bigcup_{\substack{q\in Q_\exists,a\in\Gamma\\ \delta(q,a)=(\nu,\nu')\\ \mu=(q',b,\rightarrow)\in\{\nu,\nu'\}}}\left(\{\varepsilon\}\times\mu_{i,0}\Gamma^{i-2}bq'c\Gamma^{N-i}e\right)\Gamma^{i-2}qac\Gamma^{N-i}\Sigma^*.$$

Finally, as for (5.3) and (5.4), the graph G_5 that realizes (5.5) is the union of

$$\bigcup_{1<i\leq N}\bigcup_{c\in\Gamma}\bigcup_{\substack{q\in Q_\forall,a\in\Gamma\\ \delta(q,a)=(\mu,\nu)\\ \nu=(q',b,\leftarrow)}}\left(\{l\}\times\nu_{i,0}\Gamma^{i-2}q'cb\Gamma^{N-i}r\right)\Gamma^{i-2}cqa\Gamma^{N-i}\Sigma^*$$

and

$$\bigcup_{1<i\leq N}\bigcup_{c\in\Gamma}\bigcup_{\substack{q\in Q_\forall,a\in\Gamma\\ \delta(q,a)=(\mu,\nu)\\ \nu=(q',b,\rightarrow)}}\left(\{l\}\times\nu_{i,0}\Gamma^{i-2}bq'c\Gamma^{N-i}r\right)\Gamma^{i-2}qac\Gamma^{N-i}\Sigma^*.$$

Define $G = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_=$. It is easy to verify that G is computable in time polynomial in $n = |w|$. The following fact is not hard to check.

Fact 5.4 *Let $A = (U \cup \{\varepsilon\})(\{l, r, e\}U)^*$. Then*

$$\longrightarrow^+ \cap\, A \times A \quad = \quad G^+ \cap A \times A.$$

The following equivalences finish our EXP lower bound proof:

$$\begin{array}{rcl} w \in L(\mathcal{M}) & \iff & q_0 w\square^{N-n} \in U \text{ is accepting} \\ & \stackrel{\text{Lemma 5.3}}{\iff} & \forall v \in \Sigma^* : q_0 w\square^{N-n}v \longrightarrow^+ v \\ & \stackrel{\text{Lemma 5.3}}{\iff} & q_0 w\square^{N-n} \longrightarrow^+ \varepsilon \\ & \stackrel{\text{Fact 5.4}}{\iff} & (q_0 w\square^{N-n}, \varepsilon) \in G^+. \end{array}$$

By combining the EXP upper bound that even holds for solving parity games on prefix-recognizable graphs [15], we get the following theorem.

Theorem 5.5 *The reachability problem on prefix-recognizable graphs is* EXP*-complete.* □

It is a natural question to ask what the complexity of the reachability problem is on *prefix rewrite graphs*, where transitions are given as

$$\bigcup_{1\leq i\leq k}(U_i \times V_i)\Sigma^* \quad \text{for regular languages } U_1, V_1, \ldots, U_k, V_k. \qquad (*)$$

Note that in our lower bound proof only $G_=$ was not of the form $(*)$. The latter is inevitable, since there is an easy reduction from the reachability problem on prefix rewrite graphs to the reachability problem on pushdown graphs: Transitions of the kind $(U_i \times V_i)\Sigma^*$ are simulated by a *sequence* of transitions of two NFAs that simulate U_i (resp. V_i) by popping (resp. pushing) symbols from (resp. on) the stack. Thus, reachability on prefix rewrite graphs is P-complete.

5.3 Infinite state model checking test-free PDL

In this section, we fill the picture of infinite state model checking of test-free PDL. These results can be obtained from known results of infinite state model checking of EF.

Lower Bounds

A *context-free grammar* (CFG for short) is a tuple $G = (V, \Sigma, S, P)$, where V is a finite set of *variable symbols*, Σ is a finite *terminal alphabet*, $S \in V$ is the *starting variable*, and $P : V \to 2^{(V\cup\Sigma)^*}$ with $P(X)$ finite for each $X \in V$ is the set of *production rules*. We denote by $L(G)$ 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 5.6 ([63]) *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 that reachability and hence EF model checking on BPAs is P-hard:

Corollary 5.7 *The following problem is* P*-hard:*
INPUT: A BPA $\mathcal{X} = (\Gamma, \{\xrightarrow{a}_{\mathcal{X}} \mid a \in \mathsf{A}\})$ and two worlds $x, y \in \Gamma^*$ of $K(\mathcal{X})$.
QUESTION: Is y reachable from x, i.e. $(x, y) \in (\cup_{a\in\mathsf{A}} \to_a)^*$? □

PROOF We give a logspace reduction from the non-emptiness problem of context-free grammars to the above problem. Thus, P-hardness follows from Theorem 5.6. Let $G = (V, \emptyset, S, P)$ be a CFG over the empty terminal alphabet. We construct a BPA $\mathcal{X} = \mathcal{X}(G) = (\Gamma, \{\to_{\mathcal{X}}\})$, where $\Gamma = V$ and

$$\to_{\mathcal{X}} \quad = \quad \{(X, v) \mid X \in V \wedge v \in P(X)\}.$$

It is now obvious that we have $L(G) \neq \emptyset$ if and only if ε is reachable from S in $K(\mathcal{X})$. ■

As mentioned above, it follows that the data complexity of EF and hence of test-free PDL over basic process algebras is P-hard. Over pushdown systems however, Bouajjani et al. proved a PSPACE lower bound.

Theorem 5.8 ([14]) *There exists a fixed EF formula φ such that the following problem is* PSPACE-*hard:*
INPUT: A PDS $\mathcal{Y}$ and a state q_0 of $\mathcal{Y}$.
QUESTION: $q_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

Concerning expression complexity of EF over basic process algebras, a PSPACE lower bound was shown by Mayr.

Theorem 5.9 ([83]) *There exists a fixed BPA $\mathcal{X}$ such that the following problem is* PSPACE*-hard:*
INPUT: An EF formula φ.
QUESTION: $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$?.
Moreover, this lower bound even holds if the input φ does not contain an EF-operator, i.e. φ can be chosen to be a formula of modal logic. □

Upper bounds

In [124], Walukiewicz shows that the combined complexity of EF over pushdown systems is in PSPACE.

Theorem 5.10 ([124]) *The following problem is in* PSPACE*:*
INPUT: A PDS $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$, a world $q_0 w \in Q\Gamma^*$, and an EF formula φ.
QUESTION: $q_0 w \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

From the previous theorem we can deduce the following corollary.

Corollary 5.11 *The following problem is in* PSPACE*:*
INPUT: A PDS $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$, a world $q_0 w \in Q\Gamma^*$, and a test-free PDL formula φ.
QUESTION: $q_0 w \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

PROOF Given a PDS $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\}$, a world $q_0 w \in Q\Gamma^*$, and a test-free PDL formula φ, we construct in polynomial time a PDS $\mathcal{Y}' = (Q', \Gamma, \{\xrightarrow{a}_{\mathcal{Y}'} \mid a \in \mathsf{A}'\})$, a state $q_0' \in Q'$, and an EF-formula φ' such that $q_0 w \in [\![\varphi]\!]_{K(\mathcal{Y})}$ if and only if $q_0' w \in [\![\varphi']\!]_{K(\mathcal{Y}')}$. Thus, the corollary follows from Theorem 5.10. We assume that each occurring program in φ is given by an NFA over A. Let $\varphi_1, \varphi_2, \ldots, \varphi_n$ be an enumeration of the set of all diamond subformulas of φ, i.e. each formula φ_i is of the kind $\langle \mathcal{A}_i \rangle \psi_i$ for some NFA $\mathcal{A}_i = (Q_i, \mathsf{A}, r_i, \delta_i, F_i)$ and some subformula ψ_i, such that additionally for all $i, j \in [1, n]$ we have

$$\text{if } \varphi_i \text{ is a strict subformula of } \varphi_j, \text{ then } i < j.$$

We assume moreover without loss of generality that $\varphi = \varphi_n$ is a diamond formula. The simple idea is that the states of $\mathcal{Y}'$ have the information which state of $\mathcal{Y}$ and which of the subformulas φ_i is currently tested. We define $Q' = Q \times (\biguplus_{i=1}^{n} Q_i)$ and $\mathsf{A}' = Q \uplus [n] \uplus \{\#\}$. The rewriting rules of $\mathcal{Y}'$ are defined as follows, where $q, q' \in Q$ and $r, r' \in \uplus_{i=1}^{n} Q_i$

- $(q, r) \xrightarrow{r}_{\mathcal{Y}'} (q, r)$,
- $(q, r)\gamma \xrightarrow{\#}_{\mathcal{Y}'} (q', r')v$ whenever $q\gamma \xrightarrow{a}_{\mathcal{Y}} q'v$ and $r' \in \delta_i(q, a)$, and
- $(q, r) \xrightarrow{i}_{\mathcal{Y}'} (q, r_i)$ whenever either $r = r_i$ or $r \in F_j$ for some $j > i$.

We give the definition of φ' inductively,

$$\begin{aligned} \mathtt{true}' &= \mathtt{true}, \\ (\neg\varphi)' &= \neg\varphi', \\ (\varphi_1 \vee \varphi_2)' &= \varphi_1' \vee \varphi_2', \text{ and} \\ (\langle \mathcal{A}_i \rangle \varphi)' &= \langle i \rangle \, \mathrm{EF} \left(\bigvee_{r \in F_i} \langle r \rangle \mathtt{true} \wedge \varphi' \right) \text{ for each } i \in [n]. \end{aligned}$$

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

$$q_0 w \in [\![\varphi]\!]_{K(\mathcal{Y})} \quad \Leftrightarrow \quad (q_0, r_n) w \in [\![\varphi']\!]_{K(\mathcal{Y}')}. \qquad \blacksquare$$

In [105] 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 world in $K(\mathcal{Y})$ either has no outgoing ε-transitions or no outgoing non-ε-transitions, see also [8]. Moreover, one regains the prefix-recognizable system $\mathcal{Z}$ by factoring out the ε-transitions of $\mathcal{Y}$ as follows: One takes only those worlds in $K(\mathcal{Y})$ without outgoing ε-transitions (let us denote such worlds *ε-free*) and adds an a-transition between two worlds whenever there exists a path between them in $K(\mathcal{Y})$ consisting of one a-transition followed by arbitrary many ε-transitions. It is now easy to see that Corollary 5.11 implies a PSPACE upper bound for the expression complexity of test-free PDL over prefix-recognizable systems: Given a fixed PRS $\mathcal{Z} = (\Gamma, \phi)$ and a test-free PDL formula φ each over the set of atomic programs A, we effectively compute in constant time a pushdown system $\mathcal{Y}$ with ε-transitions and the properties from above. We add transitions to $\mathcal{Y}$ such that each ε-free world of $K(\mathcal{Y})$ has a self-loop labeled by a fresh symbol $\# \notin \mathsf{A}$. Let $\mathcal{Y}'$ denote the resulting pushdown system with ε-transitions. Finally, we replace each occurring atomic program $a \in \mathsf{A}$ in φ by $\# \circ a \circ \varepsilon^* \circ \#$ and hereby receive φ'. Hence, we get the following corollary.

Corollary 5.12 *For every fixed PRS $\mathcal{Z} = (\Gamma, \phi)$ and world $x \in \Gamma^*$ the following problem is in* PSPACE*:*
INPUT: A test-free PDL formula φ.
QUESTION: $x \in [\![\varphi]\!]_{K(\mathcal{Z})}$? $\square$

Table 5.1 summarizes the results for model checking test-free PDL over BPA, PDS, and PRS.

5.4 Model-checking full PDL

Lower Bounds

Before we proceed with lower bounds of model checking full PDL, let us introduce the logic CTL. Formulas φ of the logic CTL are given by the following grammar, where $a \in \mathbb{A}$,

$$\varphi \ ::= \ \mathtt{true} \ | \ \neg\varphi \ | \ \varphi_1 \vee \varphi_2 \ | \ \langle a \rangle \varphi \ | \ \exists(\varphi \, \mathcal{U} \, \varphi) \ | \ \exists(\varphi \, w\mathcal{U} \, \varphi).$$

Let $K = (W, \{\to_a \mid a \in \mathsf{A}\})$ be a Kripke structure over some finite set of atomic programs $\mathsf{A} \subseteq \mathbb{A}$ and let $\to$ denote $\bigcup_{a\in\mathsf{A}} \to_a$. We only define to semantics of the formulas $\exists(\varphi_1\, \mathcal{U}\, \varphi_2)$ and $\exists(\varphi\, w\mathcal{U}\, \varphi)$:

$$\begin{aligned}
[\![\exists(\varphi_1\, \mathcal{U}\, \varphi_2)]\!]_K &= \{x \in W \mid \exists x_1, \ldots, x_n \in W, n \geq 1 : x = x_1 \wedge x_n \in [\![\varphi_2]\!]_K \\
&\qquad \wedge \forall i \in [n-1] : x_i \to x_{i+1} \wedge x_i \in [\![\varphi_1]\!]_K\} \\
[\![\exists(\varphi_1\, w\mathcal{U}\, \varphi_2)]\!]_K &= [\![\exists(\varphi_1\, \mathcal{U}\, \varphi_2)]\!]_K \\
&\quad \cup \{x \in W \mid \exists x_1, x_2, \ldots \in [\![\varphi_1]\!]_K : x = x_1 \wedge \forall i \geq 1 : x_i \to x_{i+1}\}
\end{aligned}$$

Hence, EF is a fragment of CTL, since EFφ is equivalent to $\exists(\mathtt{true}\, \mathcal{U}\, \varphi)$. In [124], Walukiewicz has shown that the combined complexity of CTL over pushdown systems is EXP-hard. It is not too difficult to see that his proof can be adjusted in such a way to show that CTL model checking is EXP-hard already over a fixed pushdown system.

Theorem 5.13 ([124]) *There exists a fixed PDS $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$ and a state $q_0 \in Q$ such that the following problem is* EXP*-hard:*
INPUT: A CTL formula φ.
QUESTION: $q_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

Looking at the proof of Theorem 5.13 in [124], the structure of the CTL formulas φ (over some finite set of atomic programs A) establishing the EXP lower bound are of the form $\varphi = \exists(\varphi_1\, \mathcal{U}\, \varphi_2)$, where φ_1 and φ_2 are EF formulas. It is easy to see that the PDL formula $\varphi' = \langle(\psi_1'? \circ \mathsf{A})^*\rangle\psi_2'$ is equivalent to φ, where ψ_1' (resp. ψ_2') is a logarithmic space computable test-free PDL formula equivalent to ψ_1 (resp. ψ_2). Let $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$ be the fixed PDS of the proof of Theorem 5.13. Moreover, let φ'' be the PDL formula over the set of atomic programs $Q \cup \overline{Q} \cup \Gamma \cup \overline{\Gamma}$ that emerges from φ' by replacing each occurrence of an atomic program $a \in \mathsf{A}$ by

$$\bigcup_{q\gamma \xrightarrow{a}_{\mathcal{Y}} q'v} \overline{q} \circ \overline{\gamma} \circ \overleftarrow{v} \circ q'.$$

We finally have the following equivalence, where Tree$(Q \cup \Gamma)$ denotes the complete $Q \cup \Gamma$-tree with backwards edges, see also Example 2.34,

$$q_0 \in [\![\varphi]\!]_{K(\mathcal{Y})} \quad \Leftrightarrow \quad q_0 \in [\![\varphi'']\!]_{\mathrm{Tree}(Q\cup\Gamma)}.$$

Observe that in the formula φ'' there is only one occurrence of the test operator in contrast to Corollary 5.11. Hence, we obtain an EXP lower bound for the expression complexity of PDL over BPAs.

Corollary 5.14 *There exists a fixed BPA $\mathcal{X}$ and a fixed world x of $K(\mathcal{X})$ such that the following problem is* EXP*-hard:*
INPUT: A PDL formula φ with only one occurrence of the test operator.
QUESTION: $x \in [\![\varphi]\!]_{K(\mathcal{X})}$? □

From Walukiewicz's EXP-hardness proof of the combined complexity of CTL model checking over pushdown systems [124] it is not hard to see that, instead of fixing the pushdown system and having the formula as input, one can modify the proof having the CTL formula fixed and the pushdown systems as input.

Theorem 5.15 ([124]) *There exists a fixed CTL formula φ such that the following problem is* EXP*-hard:*
INPUT: A PDS $\mathcal{Y}$ and a state q of $\mathcal{Y}$.
QUESTION: $q \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

Since the fixed CTL formula φ establishing the EXP lower bound for model checking pushdown systems is of the form $\varphi = \exists(\varphi_1 \,\mathcal{U}\, \varphi_2)$ for two EF formulas ψ_1 and ψ_2 we can, as discussed above, translate this formula into an equivalent PDL formula. Hence, we obtain an EXP lower bound for the data complexity of PDL over pushdown systems.

Corollary 5.16 *There exists a fixed PDL formula φ such that the following problem is* EXP*-hard:*
INPUT: A PDS $\mathcal{Y}$ and a state q of $\mathcal{Y}$.
QUESTION: $q \in [\![\varphi]\!]_{K(\mathcal{Y})}$? □

Upper bounds

Since there is a polynomial time translation from PDL into the alternation-free fragment of the modal μ-calculus [53] (when μ-calculus formulas are given as a dag structure), upper bounds for the modal μ-calculus carry over to PDL. So let us state known results for model checking the modal μ-calculus.

The following upper bound is known for the combined complexity of the μ-calculus over BPAs.

Theorem 5.17 ([125]) *For every fixed μ-calculus formula φ the following problem is in* P*:*
INPUT: A BPA $\mathcal{X}$.
QUESTION: $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$?

The following upper bound is known for the combined complexity of the μ-calculus over prefix-recognizable systems.

Theorem 5.18 ([68, 15]) *The following problem is in* EXP*:*
INPUT: A PRS $\mathcal{Z}$, a world w of $K(\mathcal{Z})$, and μ-calculus formula φ.
QUESTION: $w \in [\![\varphi]\!]_{K(\mathcal{Z})}$?

Table 5.1 summarizes the results for model checking full PDL over BPA, PDS, and PRS.

5.5 Open problem: EF model checking over one-counter systems

In this section, we discuss the open problem of determining the precise complexity of EF model checking over one-counter systems. A *one-counter system* is a pushdown system

$$\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\}), \quad \text{where } |\Gamma| = 1,$$

i.e. where the stack alphabet is a singleton. A recent result of Serre states that solving parity games (and hence μ-calculus model checking) over one-counter systems is in PSPACE.

Theorem 5.19 ([100]) *The combined complexity of model checking the modal μ-calculus over one-counter systems is in* PSPACE. □

In a paper of Holzer [60], it is shown that the non-emptiness problem of alternating finite automata over finite words is PSPACE-hard already over a unary alphabet, see also [62] for a direct proof. Given an alternating finite automaton $\mathcal{A}$, one can compute in polynomial time an alternating reachability game played on the transition graph of a one-counter system as follows:

1. The game begins with the empty stack.
2. Eve pushes finitely many elements on the stack, say 1^n for some $n \in \mathbb{N}$ and claims that 1^n is some member of $L(\mathcal{A})$.
3. By popping elements from the stack Adam and Eve simulate the transition function of $\mathcal{A}$.
4. Eve wins if she can force the simulation game from step 3 to reach the empty stack when the current simulation of $\mathcal{A}$ is a final state.

Moreover, taking the above results together, one can easily show the following corollary – the details are left to the reader.

Corollary 5.20 *There exists a fixed alternation-free μ-calculus formula φ such that checking whether φ is satisfied in a given world of a one-counter system is* PSPACE*-hard.* □

Note that the above described lower bound heavily relied on the usage of a fixed point formalism. Hence, it is natural to ask what the complexity of model checking the weaker logics EF or PDL is over one-counter systems. Before we discuss the current known complexity status of EF model checking over one-counter systems, we recall some complexity classes located between P and PSPACE.

The complexity class coNP is the class of all complements of problems that are in NP. The complexity class DP denotes the class of all languages L for which there exists a language $L_1 \in \mathsf{NP}$ and a language $L_2 \in \mathsf{coNP}$ such that $L = L_1 \cap L_2$. The class Θ_2^p denotes the class of all languages that can be decided by some deterministic polynomially time bounded Turing machine $\mathcal{M}$ that has access to an NP-oracle, but whose oracle queries have to be asked in parallel, i.e. on an input w the machine $\mathcal{M}$ computes instances of some language in NP (e.g. SAT), receives the corresponding bit string of query answers and finally, decides whether $w \in L$ deterministically in polynomial time. Therefore Θ_2^p is often refered to as *parallel access to* NP. See [101] for an intensive treatment of Θ_2^p-complete problems. There are numerous characterizations of Θ_2^p, e.g. in [123] Wagner proved that $\Theta_2^p = \mathsf{L}^{\mathsf{NP}}$. Dropping the requirement that the queries to some NP oracle are performed in parallel yields Δ_2^p, i.e. Δ_2^p is the class of languages that can be decided by some deterministic polynomially time bounded Turing machine that has access to an NP oracle.

The *polynomial time hierarchy*, introduced and studied in [106], is defined as follows: $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathsf{P}$,

$$\begin{aligned} \Delta_{i+1}^p &= \mathsf{P}^{\Sigma_i^p}, \\ \Sigma_{i+1}^p &= \mathsf{NP}^{\Sigma_i^p}, \text{ and} \\ \Pi_{i+1}^p &= \mathsf{coNP}^{\Sigma_i^p} \end{aligned}$$

$$\begin{tikzcd}[column sep=small]
 & \mathsf{NP} \arrow[dr, phantom, "\subseteq", sloped] & & & & \Sigma_2^p \arrow[dr, phantom, "\subseteq", sloped] & \\
\mathsf{P} \arrow[ur, phantom, "\subseteq", sloped] \arrow[dr, phantom, "\subseteq", sloped] & & \mathsf{DP} \arrow[r, phantom, "\subseteq"] & \Theta_2^p \arrow[r, phantom, "\subseteq"] & \Delta_2^p \arrow[ur, phantom, "\subseteq", sloped] \arrow[dr, phantom, "\subseteq", sloped] & & \Delta_3^p \subseteq \cdots \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \\
 & \mathsf{coNP} \arrow[ur, phantom, "\subseteq", sloped] & & & & \Pi_2^p \arrow[ur, phantom, "\subseteq", sloped] &
\end{tikzcd}$$

Figure 5.2: Some complexity classes below PSPACE.

for each $i \geq 0$. By PH we denote the union of each of the levels of the polynomial time hierarchy, i.e. $\mathsf{PH} = \bigcup_{i \geq 0} \Sigma_i^p$. For a solid introduction into the above mentioned classes, we refer to [90].

In [61] a general technique was introduced for proving DP lower bounds for several verification problems and EF model checking over one-counter systemss. Although for several of the bisimulation problems discussed [61] the lower bound has been lifted from DP to PSPACE (see e.g. [103]), there has, to the knowledge of the author, not been any improvement of the complexity status of EF model checking over one-counter systems.

Theorem 5.21 ([100],[61]) *Model checking EF over one-counter systems is in* PSPACE *and* DP-*hard.* ☐

Using techniques developed in [61], our goal is to give a Θ_2^p lower bound for EF model checking over one-counter systems. Hence we lift the lower bound from DP to Θ_2^p. Moreover, we aim at pointing out difficulties that arise when further trying to improve the lower bound to Δ_2^p-hardness using the same technique. Firstly, let us give the idea how to reduce the NP-complete 3SAT (see [90] for more details), i.e. satisfiability of boolean formulas in conjuctive normal form where every clause consists of three literals (we say that such formulas are in 3CNF), to EF model checking over one-counter systems.

Let $F(X_1, \ldots, X_k)$ be a 3CNF formula with boolean variables $X_1, \ldots, X_k$. Let p_i denote the i-th prime number for each $i \in [k]$, i.e. $p_1 = 2$, $p_2 = 3$ and so on. Let $N = \prod_{i=1}^{k} p_i$ be the product of the first k primes. The idea is that each $n \in \mathbb{N}$ corresponds to the truth assignment $\alpha_n : \{X_1, \ldots, X_k\} \to \{0, 1\}$, namely

$$\alpha_n(X_i) \quad = \quad \begin{cases} 1 & \text{if } p_i | n \\ 0 & \text{else} \end{cases} \qquad \text{for each } i \in [k].$$

Let us fix some 3CNF formula $F(X_1, \ldots, F_l) = \bigwedge_{i=1}^{l} C_i$, where each C_i is a clause with literals $Y_{i,1}, Y_{i,2}, Y_{i,3}$, i.e. $C_i = (Y_{i,1} \vee Y_{i,2} \vee Y_{i,3})$. Consider the following game between two players Eve and Adam:

1. The game begins with the empty stack.

2. Eve pushes $n \in \mathbb{N}$ elements on the stack. Eve claims that the assignment α_n satisfies F.
3. Adam chooses some $i \in [l]$ and aims at disproving the truth of clause C_i under α_n.
4. Eve chooses some $j \in [3]$ and claims that literal $Y_{i,j}$ is true under α_n.
5. Both players verify that $Y_{i,j}$ is satisfied under α_n, i.e. if e.g. $Y_{i,j} = \overline{X_m}$ for some $m \in [k]$, then it is checked whether p_m does not divide n as follows: Consecutively blocks of p_m elements are popped from the stack until the stack is empty and Eve wins if and only if the current block is not completely processed.

It is not hard to verify that F is satisfiable if and only if Eve has a winning strategy in the above game. Furthermore, it is not hard to see that the above steps can easily established by some EF formula and some one-counter system both of size at most polynomial in the size of F. Also note that by switching roles, the game can be adapted to prove *unsatisfiability* of F. It should now be clear how DP-hardness of EF model checking of one-counter systems was established in [61] via a reduction from the DP-complete problem

$$\text{SAT-UNSAT} \quad = \quad \{(\varphi, \psi) \mid \varphi, \psi \in \text{3CNF} : \varphi \text{ is satisfiable and } \psi \text{ is unsatisfiable}\}$$

The following problem, called ODD-MAX-TRUE-3SAT, was shown to be complete for Θ_2^p in [101]:
INPUT: A boolean formula F in 3CNF.
QUESTION: Is the maximum number of 1's in satisfying truth assignments for F odd?

Now consider the following game:

(1) Eve mentions some odd number $m \in [k]$ and claims that m is the maximum number for which there exists a truth assignment that assigns precisely m variables to 1.

(2) Adam now has two choices:

a) He believes Eve's choice in step (1) but demands her to mention a satisfying truth assignment α_n that assigns 1 to precisely m variables. After Eve has comitted herself to α_n (by pushing n elements on the stack), Adam either tries to disprove that α_n satisfies F (see the above game) or to disprove that α_n assigns precisely m variables to 1.

b) He does not believe Eve's choice in step (1) but mentions a truth assignment α_n he claims to satisfy F and that assigns strictly more than m variables to 1, say m' many for some $m' > m$. After having mentioned α_n, it is now Eve's turn. She can now either try to disprove that α_n satisfies F (see previous game) or to disprove that α_n assigns m' many variables to 1.

Again, it is not hard to see that F is a member of ODD-MAX-TRUE-3SAT if and only if Eve has a winning strategy in the above game. Moreover it is quite easy to see that the game can be defined by a polynomial time computable one-counter system and EF formula. Finally, we get the following proposition.

Proposition 5.22 *Model checking EF over one-counter systems is Θ_2^p-hard.* □

Let us conclude with difficulties that arise when trying to lift the lower bound of EF model checking over one-counter systems to Δ_2^p-hardness.

The following problem, called DSAT, was introduced by Papadimitriou [89] and proven to be complete for Δ_2^p:

INPUT: Equations

$$\begin{array}{rcl} X_1 & = & \varphi_1(\mathbf{Y_1}) \\ X_2 & = & \varphi_2(X_1, \mathbf{Y_2}) \\ & \vdots & \\ X_k & = & \varphi_k(X_1, \cdots, X_{k-1}, \mathbf{Y_k}), \end{array}$$

where each φ_i is a 3CNF formula with variables $X_1, \ldots, X_{i-1}$ and a further *set* of variables $\mathbf{Y_i}$.

QUESTION: Is the value of X_k equal to 1, where the *value* of each X_i equals 1 if and only if there exists some truth assignment $\alpha : \mathbf{Y_i} \to \{0, 1\}$ that satisfies $\varphi_i(X_1, \ldots, X_{i-1}, \mathbf{Y_i})$? Observe that the value of each X_i is determined.

An ad-hoc idea for reducing DSAT to EF model checking over one-counter systems, could be to let Eve claim to know the values of each of the X_i. In doing so, she would again commit herself to some assignment $\alpha : \{X_1, \ldots, X_k\} \to \{0, 1\}$ representing the values of each of the variables $X_1, \ldots, X_n$. Now if Adam mistrusted her, it is sufficient for him to mention some $j \in [k]$ such that he believes that the value of X_l equals $\alpha(X_l)$ for each $l < j$, but simultaneously claims that the correct value of X_j should be $1 - \alpha(X_j)$ instead of $\alpha(X_j)$. If Adam for instance wanted to show that the value of X_j is 1 instead of 0 he would have to mention a truth assignment $\beta(\mathbf{Y_i}) \to \{0, 1\}$ satisfying φ_i. But now the question is how he can mention β by modifying the stack *without* destroying the information that corresponds to the values of the variables $X_1, \ldots X_{j-1}$ given by α (which are essential for evaluating φ_i).

Summarizing, either the complexity of EF model checking over one-counter systems lies somewhere in the polynomial time hierarchy or probably new techniques are required to improve the current Θ_2^p lower bound. The same comments apply to (test-free) PDL. Therefore the precise complexity status of model checking EF and (test-free) PDL over one-counter systems remains an interesting open problem.

6 Conclusion and open problems

In this thesis we have studied the computational complexity of the satisfiability problem and of the infinite state model checking problem for logics related to PDL. After having introduced PDL extensions and having proved some minor new results in Chapter 2, we studied the extension of ICPDL with fixed points and nominals (μ-ICPDL$_{\mathtt{Nom}}$) in Chapter 3. We proved that satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is 2EXP-complete (Theorem 3.28) in three steps. Firstly, we proved that every satisfiable μ-ICPDL$_{\mathtt{Nom}}$ formula containing k different nominals has a countable model of tree width at most $k + 2$ (Corollary 3.6). Secondly, we reduced general satisfiability to a special reasoning problem that we called ω-regular tree satisfiability (Theorem 3.18). Thirdly, we proved that ω-regular tree satisfiability in μ-ICPDL$_{\mathtt{Nom}}$ is in 2EXP (Theorem 3.27). For its fragment ICPDL the previously best known complexity upper bound for satisfiability was non-elementary (Theorem 2.22). Too, via a reduction to ω-regular tree satisfiability, we proved that model checking of μ-ICPDL$_{\mathtt{Nom}}$ over prefix-recognizable systems is in 2EXP (Theorem 3.31). We provided matching lower bounds for model checking loop-PDL and IPDL over basic process algebras (Theorems 3.33 and 3.35).

We have studied dynamic logics of permission and proved that satisfiability in the PDL extension $\mathrm{DLP}^+_{\mathrm{dyn}}$ is complete for EXP, thus improving a previously known 2EXP upper bound (Theorem 2.27) and answering an open problem formulated in [24]. By proving that satisfiability of PDL's extension PDL$\oplus$ is 2EXP-complete (Theorem 4.20), we showed that translations from $\mathrm{DLP}^+_{\mathrm{dyn}}$ to PDL$\oplus$ may not lead to success for improving upper bounds for satisfiability in $\mathrm{DLP}^+_{\mathrm{dyn}}$.

We determined the data complexity, the expression complexity, and the combined complexity of model checking (fragments of) PDL over basic process algebras, pushdown systems, and prefix-recognizable systems. We showed that already reachability on prefix-recognizable graphs is EXP-hard (Theorem 4.5) matching the EXP upper bound that holds even for model checking the modal μ-calculus (Theorem 5.18).

We conclude this thesis with a discussion of open problems the author would like to stress:

- Is the finite satisfiability problem for IPDL/μ-ICPDL$_{\mathtt{Nom}}$ decidable?
- What precisely are the language classes $\mathcal{L}$ for which the corresponding PDL extension PDL+$\mathcal{L}$ is decidable?
- Is satisfiability in PDL extended with converse, nominals and graded modalities[1] decidable? The latter problem is of a big interest in the description logic community since it corresponds to the logic OWL DL extended with transitive closure, see also [2]. It is worth mentioning that by [13] the μ-calculus with converse, nominals, and graded modalities is undecidable. Apparently it can be shown that already satisfiability in PDL with *well-founded* programs, converse, nominals, and graded modalities is undecidable [79].
- What is the precise complexity of Data Analysis Logic (lower bound: EXP, upper bound: 2EXP)?
- What is the precise complexity of model checking EF over one-counter systems (lower bound:Θ_2^p, upper bound: PSPACE)?

[1] A graded modal formula is a formula of the kind $\langle a\rangle_{\geq k}\varphi$ and holds in those worlds that have at least k different a-successors that all satisfy φ.

Bibliography

[1] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115–135, 2005.

[2] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.

[3] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems*, 27(4):786–818, 2005.

[4] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Conference Record of the 36th Annual ACM Symposium on Theory of Computing*, pages 202–211. ACM Press, 2004.

[5] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

[6] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2003.

[7] R. Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[8] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report 2001-06, RWTH Aachen, Department of Computer Science, 2001.

[9] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37(6):641–674, 2004.

[10] M. Bojańczyk. The finite graph problem for two-way alternating automata. *Theoretical Computer Science*, 3(298):511–528, 2003.

[11] M. Bojańczyk. A bounding quantifier. In *Proceedings of the 13th Annual Conference for Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[12] P. Bonatti, C. Lutz, A. Murano, and M. Y. Vardi. The complexity of enriched μ-calculi. In *Proccedings of the 33rd International Colloquium on Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 540–551. Springer, 2006.

[13] P. A. Bonatti and A. Peron. On the undecidability of logics with converse, nominals, recursion and counting. *Artificial Intelligence*, 158(1):75–96, 2004.

[14] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurreny*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[15] T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electronic Notes Theoretical Computer Science*, 68(6), 2002.

[16] T. Cachat and I. Walukiewicz. The Complexity of Games on Higher Order Pushdown Automata. *CoRR*, abs/0705.0262, 2007.

[17] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 84–89. Morgan Kaufmann, 1999.

[18] D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79–115, 2002.

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

[20] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[21] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal Computer and System Sciences*, 46(2):218–270, 1993.

[22] R. Danecki. Nondeterministic Propositional Dynamic Logic with Intersection is decidable. In *Proceedings of the 5th Symposium on Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 34–53, 1984.

[23] R. Danecki. Propositional Dynamic Logic with Strong Loop Predicate. In *Proccedings of the 11th International Symposium on Mathematical Foundations of Computer Science*, volume 176 of *Lecture Notes in Computer Science*, pages 573–581. Springer, 1984.

[24] S. Demri. A reduction from DLP to PDL. *Journal of Logic and Computation*, 15(5):767–785, 2005.

[25] S. Demri. *Logiques pour la spécification et vérification*. Habilitation thesis, Université Paris 7, 2007.

[26] R. Diestel. *Graph Theory, Third Edition*. Springer, 2005.

[27] A. Ehrenfeucht and P. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

[28] J. Engelfriet. Iterated pushdown automata and complexity classes. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 365–373. ACM Press, 1983.

[29] J. Esparza. On the Decidabilty of Model Checking for Several mu-calculi and Petri Nets. In *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 1994.

[30] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

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

[32] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[33] L. Farinas Del Cerro and E. Orłowska. DAL-a logic for data analysis. *Theoretical Computer Science*, 36(2-3):251–264, 1985.

[34] A. Ferman and D. Harel. ? in preparation, 2000.

[35] M. J. Fischer and N. Immerman. Interpreting Logics of Knowledge in Propositional Dynamic Logic with Converse. *Information Processing Letters*, 25(3):175–181, 1987.

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

[37] J. Flum. On the (infinite) model theory of fixed point logics. In *Models, Algebras, and Proofs: selected papers of the X Latin American symposium on mathematical logic*, volume 203 of *Lecture Notes in Pure and Applied Mathematics*, pages 67–75. Marcel Dekker, Inc., 1999.

[38] G. De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Universitá di Roma, 1995.

[39] G. De Giacomo. Eliminating "Converse" from Converse PDL. *Journal of Logic, Language and Information*, 5(2):193–208, 1996.

[40] G. De Giacomo and M. Lenzerini. Boosting the Correspondence between Description Logics and Propositional Dynamic Logics. In *Proceedings of the 12th National Conference on Artifical Intelligence*, pages 205–212, 1994.

[41] S. Göller. On the Complexity of Reasoning About Dynamic Policies. In *Proceedings of the 17th Annual Conference for Computer Science Logic*, volume 4646 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2007.

[42] S. Göller. Reachability on prefix-recognizable graphs. *Information Processing Letters*, 108(2):71–74, 2008.

[43] S. Göller and M. Lohrey. Infinite State Model-Checking of Propositional Dynamic Logics. In *Proceedings of the 16th Annual Conference for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2006.

[44] S. Göller and M. Lohrey. Infinite State Model-Checking of Propositional Dynamic Logics. Technical Report 2006/04, Universität Stuttgart, February 2006.

[45] S. Göller, M. Lohrey, and C. Lutz. PDL with Intersection and Converse is 2EXP-complete. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures*, volume 4423 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2007.

[46] S. Göller, M. Lohrey, and C. Lutz. PDL with Intersection and Converse: Satisfiability and Infinite-State Model Checking. *Journal of Symbolic Logic*, 74(1):279–314, 2009.

[47] S. Göller and D. Nowotka. A Note on an Extension of PDL. *Journal of Applied Logic*, 6(4):606–608, 2008.

[48] E. Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Computer Science*, 288(1):129–152, 2002.

[49] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Proceedings of the 14th Symposium on Logic in Computer Science*, pages 45–54. IEEE Computer Society, 1999.

[50] J. Friso Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, 1994.

[51] M. Hague and L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures*, volume 4423 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2007.

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

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

[54] D. Harel, A. R. Meyer, and V. R. Pratt. Computability and completeness in logics of programs (preliminary report). In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pages 261–268. ACM, 1977.

[55] D. Harel and M. Paterson. Undecidability of PDL with $L = \{a^{2^i} | i \geq 0\}$. *Journal of Computer and System Sciences*, 29(3):359–365, 1984.

[56] D. Harel, A. Pnueli, and J. Stavi. Propositional Dynamic Logic of Nonregular Programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[57] D. Harel, A. Pnueli, and M. Vardi. Two dimensional temporal logic and PDL with intersection. unpublished, 1982.

[58] D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM Journal of Computing*, 22(4):857–874, 1993.

[59] D. Harel and E. Singerman. More on Nonregular PDL: Finite Models and Fibonacci-Like Programs. *Inf. Comput.*, 128(2):109–118, 1996.

[60] M. Holzer. On emptiness and counting for alternating finite automata. In *Developments in Language Theory*, pages 88–97, 1995.

[61] P. Jancar, Antonín Kucera, F. Moller, and Z. Sawa. DP lower bounds for equivalence-checking and model-checking of one-counter automata. *Information and Computation*, 188(1):1–19, 2004.

[62] P. Jancar and Z. Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Information Processing Letters*, 104(5):164–167, 2007.

[63] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. In *Proceedings of the 6th annual ACM Symposium on Theory of Computing*, pages 40–46. ACM Press, 1974.

[64] D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266. IEEE Computer Society Press, 1977.

[65] D. Kozen. Results on the propositional μ-calculus. In *In Proceedings of the 9th International Conference on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1982.

[66] O. Kupferman, N. Piterman, and M. Y. Vardi. Model Checking Linear Properties of Prefix-Recognizable Systems. In *Proceedings of the 14th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[67] O. Kupferman, U. Sattler, and M. Y. Vardi. The Complexity of the Graded μ-Calculus. In *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2002.

[68] O. Kupferman and M. Y. Vardi. An Automata-Theoretic Approach to Reasoning about Infinite-State Systems. In *Proceedings of the 12th International Conference on Computer Aided Verification*, number 1855 in Lecture Notes in Computer Science, pages 36–52. Springer, 2000.

[69] D. Kuske and M. Lohrey. Hamiltonicity of automatic graphs. In *Fifth Ifip International Conference On Theoretical Computer Science - Tcs 2008*, volume 273 of *IFIP International Federation for Information Processing*, pages 445–459, 2008.

[70] M. Lange. A Lower Complexity Bound for Propositional Dynamic Logic with Intersection. In *Advances in Modal Logic*, volume 5, pages 133–147. King's College Publications, 2005.

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

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

[73] M. Leucker. Prefix-Recognizable Graphs and Monadic Logic. In *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, pages 263–284. Springer, 2001.

[74] C. Löding. Reachability Problems on Regular Ground Tree Rewriting Graphs. *Theory of Computing Systems*, 39(2):347–383, 2006.

[75] C. Löding, C. Lutz, and O. Serre. Propositional Dynamic Logic with Recursive Programs. *Journal of Logic and Algebraic Programming*, 73(1–2):51–69, 2007.

[76] C. Löding and P. Rohde. Model checking and satisfiability for sabotage modal logic. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 302–313, 2003.

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

[78] C. Lutz. PDL with Intersection and Converse is Decidable. In *Proceedings of the Annual Conference for Computer Science Logic*, volume 3624 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[79] C. Lutz. Personal communication. 2007.

[80] C. Lutz and D. Walther. PDL with Negation of Atomic Programs. *Journal of Applied Non-Classical Logic*, 15(2):189–214, 2005.

[81] A. J. Mayer and L. J. Stockmeyer. The complexity of word problems - This time with interleaving. *Information and Computation*, 115(2):293–311, 1994.

[82] A. J. Mayer and L. J. Stockmeyer. The complexity of PDL with interleaving. *Theoretical Computer Science*, 161(1-2):109–122, 1996.

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

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

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

[86] J. Meyer. Dynamic Logic for Reasoning about Actions and Agents. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 281–311. Kluwer Academic Publishers, 2000.

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

[88] D. Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189(1-2):1–69, 1997.

[89] C. H. Papadimitriou. On the complexity of unique solutions. *Journal of the ACM*, 31(2):392–400, 1984.

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

[91] V. R. Pratt. Semantical Considerations on Floyd-Hoare Logic. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pages 109–121. IEEE, 1976.

[92] V. R. Pratt. A practical decision method for propositional dynamic logic (preliminary report). In *Proceedings of the 10th annual ACM Symposium on Theory of computing*, pages 326–337, New York, NY, USA, 1978. ACM Press.

[93] V. R. Pratt. Using Graphs to Understand PDL. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 387–396. Springer, 1981.

[94] V.R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20:231–254, 1980.

[95] R. Pucella and V. Weissman. Reasoning about Dynamic Policies. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 453–467. Springer, 2004.

[96] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1968.

[97] P. Rohde. *On Games and Logics over Dynamically Changing Structures*. PhD thesis, RWTH Aachen, 2005.

[98] U. Sattler and M. Y. Vardi. The Hybrid μ-Calculus. In *Proceedings of the First International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2001.

[99] W. J. Savitch. Deterministic Simulation of Non-Deterministic Turing Machines. In *Conference Record of First ACM Symposium on Theory of Computing*, pages 247–248. ACM Press, 1969.

[100] O. Serre. Parity Games Played on Transition Graphs of One-Counter Processes. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures*, volume 3921 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006.

[101] H. Spakowski and J. Vogel. θ_2^p-Completeness: A Classical Approach for New Results. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2000.

[102] J. Srba. Strong Bisimilarity and Regularity of Basic Process Algebra Is PSPACE-Hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 716–727. Springer, 2002.

[103] J. Srba. Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation. In *Proceedings of the 15th Annual Conference for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2006.

[104] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[105] Colin Stirling. Decidability of bisimulation equivalence for pushdown processes. unpublished, 2000.

[106] L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[107] D. Suwimonteerabuth, S. Schwoon, and J. Esparza. Efficient Algorithms for Alternating Pushdown Systems with an Application to the Computation of Certificate Chains. In *Proceedings of th 4th International Symposium on Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2006.

[108] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[109] B. ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems*, pages 328–337. ACM Press, 2006.

[110] B. ten Cate and C. Lutz. The Complexity of Query Containment in Expressive Fragments of XPath 2.0. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems*, pages 73–82. ACM Press, 2007.

[111] W. Thomas. Some Perspectives of Infinite-State Verification. In *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis*, volume 3707 of *Lecture Notes in Computer Science*, pages 3–10. Springer, 2005.

[112] S. La Torre, P. Madhusudan, and G. Parlato. A Robust Class of Context-Sensitive Languages. In *Proceedings of the 22nd Symposium on Logic in Computer Science*, pages 161–170. IEEE, 2007.

[113] D. Vakarelov. Filtration theorem for dynamic algebras with tests and inverse operator. In *Logic of Programs*, volume 148 of *Lecture Notes in Computer Science*, pages 314–324. Springer, 1980.

[114] J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Napoli, 1983.

[115] J. van Benthem. An essay on sabotage and obstruction. In *Festschrift in Honour of Prof. Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer, 2002.

[116] W. van der Hoek and J. Meyer. A complete epistemic logic for multiple agents - combining distributed and common knowledge. In *Epistemic Logic and the Theory of Games and Decisions*, pages 35–68. Kluwer, 1997.

[117] R. van der Meyden. The Dynamic Logic of Permission. *Journal of Logic and Computation*, 6(3):465–479, 1996.

[118] H. P. van Ditmarsch, W. van der Hoek, and B. P. Kooi. Concurrent dynamic epistemic logic for MAS. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 201–208. ACM Press, 2003.

[119] M. Y. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 137–146. ACM Press, 1982.

[120] M. Y. Vardi. The Taming of Converse: Reasoning about Two-way Computations. In *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 1985.

[121] M. Y. Vardi. Reasoning about The Past with Two-Way Automata. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443, pages 628–641, London, UK, 1998. Springer.

[122] M. Y. Vardi and P. Wolper. Automata-Theoretic techniques for modal logics of programs. *Journal of Computer and Systen Sciences*, 32(2):183–221, 1986.

[123] K. W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[124] I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.

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

[126] R. J. Wieringa and J. C. Meyer. *Applications of Deontic Logic in Computer Science: A Concise Overview*, pages 17–40. John Wiley & Sons, 1993.

[127] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Belgian Mathematical Society*, 8(2):359–391, 2001. Journées Montoises d'Informatique Théorique (Marne-la-Vallée, 2000).

[128] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. *Logical Methods in Computer Science*, 3(4), 2007.

[129] Stefan Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products*. PhD thesis, RWTH Aachen, 2005.

[130] J. Zappe. Modal μ-Calculus and Alternating Tree Automata. In *Automata, Logics, and Infinite Games*, number 2500 in Lecture Notes in Computer Science, pages 95–106. Springer, 2002.

Index

List of Figures

List of Tables

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Die Promotionsordnung der Fakultät für Mathematik und Informatik der Universität Leipzig vom 25. Februar 2002 erkenne ich an.

Die vorliegende Dissertation wurde an der Universität Stuttgart und der Universität Leipzig unter der wissenschaftlichen Betreuung von Prof. Dr. Markus Lohrey angefertigt.

Leipzig, den 7. Juli 2008 Stefan Göller

Wissenschaftlicher Werdegang

seit 09/07	Wissenschaftlicher Mitarbeiter *Universität Leipzig, Institut für Informatik*
04/05 – 08/07	Wissenschaftlicher Mitarbeiter *Universität Stuttgart, Institut für Formale Methoden der Informatik*
02/05	Diplom in Informatik Titel der Diplomarbeit: *Paritätsspiele auf hierarchischen Graphen*
10/00 – 02/05	Studium der Softwaretechnik *Universität Stuttgart*
06/00	Abitur *Otto-Hahn-Gymnasium Ludwigsburg*