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

PDL WITH INTERSECTION AND CONVERSE: SATISFIABILITY AND INFINITE-STATE MODEL CHECKING

STEFAN GÖLLER, MARKUS LOHREY, AND CARSTEN LUTZ

Abstract. We study satisfiability and infinite-state model checking in ICPDL, which extends Propositional Dynamic Logic (PDL) with intersection and converse operators on programs. The two main results of this paper are that (i) satisfiability is in 2EXPTIME, thus 2EXPTIME-complete by an existing lower bound, and (ii) infinite-state model checking of basic process algebras and pushdown systems is also 2EXPTIME-complete. Both upper bounds are obtained by polynomial time computable reductions to ω-regular tree satisfiability in ICPDL, a reasoning problem that we introduce specifically for this purpose. This problem is then reduced to the emptiness problem for alternating two-way automata on infinite trees. Our approach to (i) also provides a shorter and more elegant proof of Danecki's difficult result that satisfiability in IPDL is in 2EXPTIME. We prove the lower bound(s) for infinite-state model checking using an encoding of alternating Turing machines.

§1. Introduction. In 1979, Fischer and Ladner introduced Propositional Dynamic Logic (PDL) as a logical formalism for reasoning about programs [15]. Since then, PDL has become a classic of logic in computer science [23], and many extensions and variations have been proposed. Several of these extensions are 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 agent-based systems [33], regular path constraints for querying semi-structured data [2], and XML-querying [1, 38, 39]. In artifical intelligence, PDL received attention due to its close relationship to description logics [17] and epistemic logic [41, 42].

PDL comprises expressions of two sorts: formulas, built from boolean and modal operators and interpreted as sets of worlds of a Kripke structure; and programs, built from the operators test, union, composition, and Kleene star (reflexive transitive closure), and interpreted as binary relations in a Kripke structure. A prominent way to obtain extensions of PDL is to admit additional program operators. 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

1991 *Mathematics Subject Classification.* This is not required.
Key words and phrases. This is not required.
The first and second author are supported by the DFG project GELO; the third author is supported by the EU funded IST-2005-7603 FET Project Thinking Ontologies (TONES).

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

a dramatic way. For example, PDL extended with a converse program operator (CPDL) inherits the tree model property from PDL; moreover, the classical result of Fischer and Ladner stating that satisfiability in PDL is complete for EXPTIME (deterministic exponential time) [15, 36] extends to CPDL without difficulty [43]. Other extensions pose more challenges. A notorious such example is PDL extended with an intersection operator on programs (IPDL), which has an intricate model theory that involves 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. using automata on infinite trees [35] or embedding into the alternation-free fragment of Kozen's μ-calculus [24].

In this paper, we study PDL extended with both intersection and converse. The resulting logic ICPDL has a number of interesting applications in computer science. For example, the information logic DAL (for data analysis logic) [14] and a number of epistemic logics for reasoning about distributed knowledge (which corresponds to program intersection [13]) can be embedded into ICPDL. This enables the transfer of results such as decidability and upper complexity bounds. The presence of converse is important in this context because it admits to define programs that are interpreted as equivalence relations—see [30] for more details. ICPDL also provides an important background theory for description logics with role intersection [4]. The aim of this paper is to prove decidability of satisfiability and (some variants of) infinite-state model checking in ICPDL, and to pinpoint the exact computational complexity. While satisfiability is the most relevant problem in the applications mentioned above, infinite-state model checking is important for applications of ICPDL in reasoning about programs.

ICPDL is closely related to IPDL and, in particular, shares its complex model theory. The history of this family of extensions of PDL started in 1984, when Danecki proved an upper bound of 2EXPTIME (deterministic doubly exponential time) for satisfiability in IPDL using an intricate reduction to the emptiness of automata on infinite trees [9]. Alas, Danecki's proof is rather difficult and many details are omitted in the published version. More than 20 years later, a matching 2EXPTIME lower bound was shown by Lange et al. [28]. Only recently, satisfiability in ICPDL was proved to be decidable using a reduction to monadic second order logic over the infinite binary tree [30]. However, this only yields a non-elementary algorithm that does not match the 2EXPTIME lower bound inherited by ICPDL from IPDL.

We prove two main results, which are discussed in some detail in what follows. The first main result is that satisfiability in ICPDL is in 2EXPTIME, and thus 2EXPTIME-complete. The proof consists of three clearly separated parts. In part one, we establish a certain model property for ICPDL based on the notion of tree width. Intuitively, the tree width of a graph measures how close the graph is to a tree. We show that every satisfiable ICPDL formulas has a model of tree width at most two, i.e., the tree model property of PDL is replaced with an "almost tree model property" in (IPDL and) ICPDL.

In part two of our proof, we use the established model property to give a polynomial time computable reduction of satisfiability in ICPDL to a novel reasoning problem in ICPDL that we call *ω-regular tree satisfiability*. This problem is defined in terms of two-way alternating parity automata on infinite trees (TWAPTAs) [44]. The basic idea is that infinite node-labeled trees as accepted by TWAPTAs can be viewed in a natural way as Kripke structures, and thus we can interpret ICPDL formulas in such trees. Now, ω-regular tree satisfiability in ICPDL is the following problem: given an ICPDL formula φ and a TWAPTA $\mathcal{T}$, is there a tree accepted by $\mathcal{T}$ which is a model of φ? Our reduction of satisfiability to this problem is based on a suitable tree-encoding of Kripke structures of tree width at most two. The TWAPTA constructed in the reduction accepts precisely such encodings.

Finally, in part three we reduce ω-regular tree satisfiability to the non-emptiness problem for TWAPTAs. Since our reduction involves an exponential blow-up and the latter problem is EXPTIME-complete [44], we obtain a 2EXPTIME upper bound for ω-regular tree satisfiability in ICPDL. Via part two of our proof, we get the same result for (standard) satisfiability in ICPDL. A notable virtue of the described three-step approach is that it is shorter and (hopefully) more comprehensible than Danecki's original upper bound for IPDL. Also, large parts of the proof can be reused for deciding infinite-state model checking.

We also take a brief look at satisfiability in two relatives of ICPDL. First, we obtain as an easy corollary of the proof of our first main result that satisfiability in the extension of PDL with the *loop-construct* (loop-PDL) belongs to EXPTIME. The mentioned construct allows to express that we can execute a given program such that, at the end, we are at the same world where we started. Danecki used a dedicated proof to show that satisfiability in loop-PDL is in EXPTIME [10]. Second, we investigate the option of further extending ICPDL with program negation. Unfortunately, this addition turns out to destroy the nice computational behaviour of ICPDL. We prove that already IPDL extended with negation restricted to atomic programs is complete for Σ_1^1, the first level of the analytic hierarchy [37]. This is in contrast with the decidability result for PDL extended with atomic program negation, as given in [31].

Our second main result is concerned with infinite-state model checking. Model checking is the prime reasoning problem in program verification [8], and has traditionally been concerned with finite-state systems represented by finite Kripke structures. Lange studies this original problem in a PDL context in [27], proving PTIME-completeness for PDL and various extensions of it. In this paper, we consider model checking of infinite state systems, which are represented by infinite Kripke structures. PDL provides natural expressive power in this context; notably, it allows to express regular reachability properties, which were used in the context of infinite-state model checking e.g. in [29, 32, 47].

When model checking infinite-state systems, we need a formalism for finitely representing infinite Kripke structures. A number of different such formalisms are available which vary considerably in expressive power and give rise to different versions of the infinite-state model checking problem [40]. In the rest of this discussion, we follow Mayr's uniform classification in terms of parallel and sequential composition [32].

When parallel composition is admitted to describe infinite state systems, model checking is undecidable already in PDL. In particular, this holds for infinite-state systems described by BPPs (basic parallel processes), which correspond to Petri nets in which every transition needs exactly one token for firing. As described in more detail in [18], undecidability is a consequence of a result by Esparza stating that model checking Petri nets in EF is undecidable, where EF denotes the fragment of CTL that only contains the modalities "next" and "exists finally" [11]. Because of this negative result, we concentrate on sequential composition. Mayr's classification thus leads us to pushdown systems (PDS) and basic process algebras (BPA). Pushdown systems were used to model the state space of programs with nested procedure calls [12], and model checking PDSs was studied for various temporal logics such as LTL, CTL, and the modal μ-calculus [3, 12, 26, 45, 46]. BPAs are a special case of pushdown systems, namely those with a single internal state.

Infinite-state model checking in (several variations of) PDL was studied in [18, 19]. For example, it was shown that model-checking PDSs in test-free PDL is PSPACE-complete with respect to combined complexity, expression complexity, and data complexity, and that the same problems are EXPTIME-complete in PDL with the test-operator. With or without this operator, the combined complexity and expression complexity does not differ when moving from PDSs to BPAs, but the data complexity drops to PTIME-completeness.

The second main result of this paper is 2EXPTIME-completeness of model checking BPAs and PDSs in ICPDL. In contrast to the case of satisfiability, we have to establish both an upper and a lower bound. The upper bound concerns the wider class of PDSs, and is established by exhibiting a polynomial time reduction to ω-regular tree satisfiability in ICPDL. This allows to reuse our 2EXPTIME upper bound for ω-regular tree satisfiability in ICPDL, which we have obtained in the context of satisfiability. The same method can easily be lifted to prefix-recognizable systems [6, 5, 25], which generalize PDSs. As a corollary of our proof, we also obtain an EXPTIME upper bound for data complexity (i.e., for a fixed ICPDL formula). Matching lower bounds are proved by extending a technique from [45], which uses a reduction from the 2EXPTIME-complete word problem for exponentially space-bounded alternating Turing machines. It allows to prove a 2EXPTIME lower bound for model checking BPAs in test-free IPDL (i.e. the test operator and converse are not needed). Moreover, this lower bound already holds for expression complexity, which means that the BPA can assumed to be fixed. For the data complexity we prove an EXPTIME lower bound for the same problem, using a similar (but easier) reduction.

This paper is organized as follows. In Section 2, we introduce the syntax and semantics of ICPDL, and give a first example. Section 3 lays the foundation for our upper bounds by introducing ω-regular tree satisfiability, and showing that this problem is 2EXPTIME-complete in ICPDL. Then, Section 4 is concerned with the upper bound for satisfiability. We prove the almost tree model property, give the reduction to ω-regular tree satisfiability, point out containment in EXPTIME of loop-PDL, and prove Σ_1^1-completeness of IPDL extended with atomic negation of programs. We then turn to model checking in Section 5, where we introduce PDSs and BPAs, give an example of a useful property of programs expressible in ICPDL, and give the reduction to ω-regular tree satisfiability. This is followed by the lower bounds for model checking in Section 6. Finally, we conclude in Section 7.

The results presented in this paper are based on the conference papers [19, 20].

§2. ICPDL basics. Let $\mathbb{P}$ and $\mathbb{A}$ be countably infinite sets of *atomic propositions* and *atomic programs*, respectively. *Formulas* φ and *programs* π of ICPDL are defined by the following grammar, where p ranges over $\mathbb{P}$ and a over $\mathbb{A}$:

$$\begin{array}{rcl} \varphi & ::= & p \mid \neg\varphi \mid \langle\pi\rangle\, \varphi \\ \pi & ::= & a \mid \overline{a} \mid \pi_1 \cup \pi_2 \mid \pi_1 \cap \pi_2 \mid \pi_1 \circ \pi_2 \mid \pi^* \mid \varphi? \end{array}$$

We use the usual abbreviations $\texttt{false} = p \wedge \neg p$, $\texttt{true} = \neg\texttt{false}$, $\varphi_1 \wedge \varphi_2 = \langle\varphi_1?\rangle\varphi_2$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \to \varphi_2 = \neg\varphi_1 \vee \varphi_2$, and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$.

The semantics of ICPDL is defined in terms of Kripke structures. A *Kripke structure* is a tuple $K = (X, \{\to_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$, where

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

Given a Kripke structure $K = (X, \{\to_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$, we define for each ICPDL program π a binary relation $[\![\pi]\!]_K \subseteq X \times X$ and for each ICPDL formula φ a subset $[\![\varphi]\!]_K \subseteq X$, using mutual induction as follows: [1]

$$\begin{array}{rcl} [\![p]\!]_K & = & X_p \text{ for } p \in \mathbb{P} \\ [\![\neg\varphi]\!]_K & = & X \setminus [\![\varphi]\!]_K \\ [\![\langle\pi\rangle\varphi]\!]_K & = & \{x \mid \exists y : (x,y) \in [\![\pi]\!]_K \wedge y \in [\![\varphi]\!]_K\} \\ [\![a]\!]_K & = & \to_a \text{ for } a \in \mathbb{A} \\ [\![\overline{a}]\!]_K & = & \{(y,x) \mid x \to_a y\} \text{ for } a \in \mathbb{A} \\ [\![\varphi?]\!]_K & = & \{(x,x) \mid x \in [\![\varphi]\!]_K\} \\ [\![\pi^*]\!]_K & = & [\![\pi]\!]_K^* \\ [\![\pi_1 \text{ op } \pi_2]\!]_K & = & [\![\pi_1]\!]_K \text{ op } [\![\pi_2]\!]_K \text{ for op} \in \{\cup, \cap, \circ\} \end{array}$$

Note that the converse operator is only applied to atomic programs. This is not a restriction since applying converse to a test program does not make much sense and converse commutes with all other program operators.

If $x \in [\![\varphi]\!]_K$ for some $x \in X$, then the Kripke structure K is a *model* of φ. An ICPDL formula φ is *satisfiable* if there exists a model of φ. As mentioned in the introduction, ICPDL does neither have the tree model property nor the finite model property, i.e., there are satisfiable ICPDL formulas such that all models are not tree-shaped and infinite, respectively. The former is witnessed e.g. by the formulas

$$(1) \qquad \neg p \wedge \langle a \cap \overline{a}\rangle p \quad \text{and} \quad \neg p \wedge \langle (a \circ p? \circ a) \cap \texttt{true}?\rangle\texttt{true},$$

[1] Overloading notation, we use $\circ$ both as a program operator of ICPDL and to denote the composition operator for binary relations, i.e., $R \circ S = \{(a,b) \mid \exists c : (a,c) \in R, (c,b) \in S\}$. Likewise for $\cup$ and $\cap$.

which both enforce a cycle of length 2. It is easy to modify these formulas such that they enforce a cycle whose length is exponential in the length of the formula. Lack of the finite model property is witnessed by the formula

$$[a^*](\ \langle a\rangle\texttt{true} \wedge \neg\langle (a\circ a^*)\cap \texttt{true}?\rangle\texttt{true}\ ). \tag{2}$$

Note that the second conjunct of the latter formula prohibits the existence of a cycle labeled by a-transitions.

If we disallow the program operator $\cap$ and the programs $\overline{a}$ ($a \in \mathbb{A}$), we drop the letter I and C from ICPDL, respectively. Thus, we obtain the fragments PDL, CPDL, and IPDL, which we identify with the classes of admitted formulas. For a class $\mathcal{C}$ of ICPDL formulas, the *satisfiability problem in* $\mathcal{C}$ is the problem to decide, given a formula $\varphi \in \mathcal{C}$, whether φ is satisfiable. The second reasoning problem considered in this paper, infinite state model checking, requires some more preliminaries and is introduced in Section 5.

Another extension of PDL that will be considerd in this paper is loop-PDL, i.e., PDL extended with a formula operator $\mathrm{loop}(\pi)$, which has the following semantics:

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

Note that loop can be defined using intersection, as used implicitly already in Formulas (1) and (2):

$$[\![\mathrm{loop}(\pi)]\!]_K = [\![\langle \pi \cap \texttt{true}?\rangle\texttt{true}]\!]_K. \tag{3}$$

We now give an example of an ICPDL (even loop-PDL) formula that is useful for program verification. In the example and in the remainder of this paper, we identify Kripke structures

$$K = (X, \{\to_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\}) \text{ and } K' = (X, \{\to_a \mid a \in \mathsf{A}\}, \{X_p \mid p \in \mathsf{P}\})$$

if $\mathsf{A} \subseteq \mathbb{A}$ and $\mathsf{P} \subseteq \mathbb{P}$ are such that $\to_a = \emptyset = X_p$ for all $a \in \mathbb{A}\setminus \mathsf{A}$ and $p \in \mathbb{P}\setminus\mathsf{P}$. This is justified by the fact that for every ICPDL formula φ, which only uses atomic programs from A and atomic propositions from P, we have $[\![\varphi]\!]_K = [\![\varphi]\!]_{K'}$.

EXAMPLE 2.1. *Let* $\mathsf{A} \subseteq \mathbb{A}$ *be a finite set of atomic programs and let* $K = (X, \{\to_a \mid a \in \mathsf{A}\})$ *be a* deterministic *Kripke structure, i.e., for all* $x \in X$ *and* $a \in \mathsf{A}$, *there is at most one* $y \in X$ *with* $x \xrightarrow{a} y$.

A recovery world *in* K *is an* $x \in X$ *such that, wherever we go from* x, *we can always return. Formally, we require that* $(x,y) \in [\![\mathsf{A}^*]\!]_K$ *implies* $(y,x) \in [\![\mathsf{A}^*]\!]_K$, *for all* $y \in X$ *where* A^* *abbreviates* $(\bigcup_{a\in\mathsf{A}} a)^*$. *Observe that we travel atomic programs only in the forward direction. In program verification, the existence of a recovery world means that the program can always return to a safe state.*

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

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

Note that, due to the determinism of K, *the subformula* $\mathrm{loop}(a\circ A^*)$ *can be read as "for* all *worlds* y *reachable via* a, *we can return via* A^* *to the current world".*

As a final preliminary, we fix the *size* of ICPDL formulas and programs. It is denoted by $|\cdot|$ and defined by mutual induction: $|p| = |a| = |\overline{a}| = 1$ for all $p \in \mathbb{P}$ and $a \in \mathbb{A}$, $|\neg\psi| = |\psi?| = |\psi|+1$, $|\langle\pi\rangle\psi| = |\pi|+|\psi|+1$, $|\pi_1 \text{ op } \pi_2| = |\pi_1|+|\pi_2|+1$ for $\text{op} \in \{\cup, \cap, \circ\}$, and $|\pi^*| = |\pi| + 1$.

§3. Deciding ω-regular tree satisfiability. The aim of this section is to introduce an additional reasoning problem called ω-regular tree satisfiability, and to show that, in ICPDL, this problem is 2EXPTIME-complete. The main benefit of this result is that upper complexity bounds for satisfiability and infinite state model checking of prefix-recognizable systems can then be obtained by transparent reductions to ω-regular tree satisfiability.

3.1. Preliminaries. Informally, ω-regular tree satisfiability in ICPDL means to decide, given an ICPDL formula φ and a two-way alternating parity tree automaton (TWAPTA) $\mathcal{T}$ as defined in detail below, whether there is an infinite tree in $L(\mathcal{T})$ 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 $T : \Sigma_E^* \to \Sigma_N$ whose domain, denoted $\mathrm{dom}(T)$, is non-empty and prefix-closed. The elements of $\mathrm{dom}(T)$ are the *nodes* of T. If $\mathrm{dom}(T) = \Sigma_E^*$, then T is called *complete*. In the rest of the paper, we mostly work with complete trees. A node $va \in \mathrm{dom}(T)$ with $a \in \Sigma_E$, is called the *a-successor* of v, and v is the *a-predecessor* of va. We use $\mathsf{tree}(\Sigma_N, \Sigma_E)$ 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 are complete 2^{P}-labeled A-trees, where $\mathsf{A} \subseteq \mathbb{A}$ and $\mathsf{P} \subseteq \mathbb{P}$ are finite sets of atomic propositions and atomic programs, respectively. Such a tree T can be identified with the Kripke structure $(\mathsf{A}^*, \{\to_a \mid a \in \mathsf{A}\}, \{T_p \mid p \in \mathsf{P}\})$, where $\to_a = \{(u, ua) \mid u \in \mathsf{A}^*\}$ for all $a \in \mathsf{A}$ and $T_p = \{u \in \mathsf{A}^* \mid p \in T(u)\}$ for $p \in \mathsf{P}$. Observe that Kripke structures derived in this way are deterministic and total with respect to A, i.e., the transition relation $\to_a$ is a total function for all $a \in \mathsf{A}$.

To define TWAPTAs, we need a few preliminaries. For a finite set X, we denote by $\mathcal{B}^+(X)$ the set of all *positive boolean formulas* where the elements of X are used as variables. The constants `true` and `false` are admitted, i.e. we have $\mathtt{true}, \mathtt{false} \in \mathcal{B}^+(X)$ for any set X. A subset $Y \subseteq X$ can be seen as a valuation in the obvious way by assigning true to all elements of Y and false to all elements of $X \setminus Y$. 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 $d \in \Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\}$ define

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

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*,

- $\delta : S \times \Sigma_N \to \mathcal{B}^+(\text{mov}(\Sigma_E))$ is the *transition function*, where $\text{mov}(\Sigma_E) = S \times (\Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\})$ is the set of *moves*,
- $s_0 \in S$ is the *initial state*, and
- $\text{Acc} : S \to \mathbb{N}$ 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 automaton 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 send a copy to the a-predecessor (if existing), and $\langle s, \varepsilon\rangle$ means 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, $T \in \mathsf{tree}(\Sigma_N, \Sigma_E)$, $u \in \Sigma_E^*$ 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 $\alpha \in \mathsf{dom}(T_R)$, if $T_R(\alpha) = (p, v)$ and $\delta(p, T(v)) = \theta$, then there is a subset $Y \subseteq \text{mov}(\Sigma_E)$ that satisfies θ and such that for all $(p', d) \in Y$, $v \cdot d$ is defined and there exists a successor β of α in T_R with $T_R(\beta) = (p', v \cdot d)$.

We say that an (s, u)-run T_R is *successful* if for every infinite path $\alpha_1\alpha_2\cdots$ in T_R (which is assumed to start at the root), the number

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

is even. For $s \in S$ define

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

Now the language $L(\mathcal{T})$ accepted by $\mathcal{T}$ is defined as

$$L(\mathcal{T}) = \{T \in \mathsf{tree}(\Sigma_N, \Sigma_E) \mid (T, \varepsilon) \in [\![\mathcal{T}]\!]\}.$$

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 lengths of all positive boolean formulas that appear in the range of δ.

We remark that our model of TWAPTAs differs slightly from other definitions that can be found in the literature. First, we run TWAPTAs only on complete trees, which will facilitate some technical constructions later on. Second, standard TWAPTAs have moves of the form $(s, -1)$ for moving to the parent node. In our model, we use moves of the form $(s, \overline{a})$, 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 following result of Vardi also applies to our version of TWAPTAs. Let $\exp(n)$ be an abbreviation for $2^{n^{\mathcal{O}(1)}}$. Since $\exp(n)$ embraces all functions whose exponent to base 2 is an arbitrary polynomial, equalities such as $\exp(n + m) = \exp(n \cdot m)$ are true and will be used without further notice.

Theorem 3.1 (Vardi [44]). *For a given TWAPTA $\mathcal{T}$ with transition function δ, it can be checked in time $\exp(|\mathcal{T}| + i(\mathcal{T})) \cdot |\delta|^{\mathcal{O}(1)}$ whether $L(\mathcal{T}) \neq \emptyset$.*

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

The rest of this section is devoted to showing that ω-regular tree satisfiability in ICPDL is in 2EXPTIME. Moreover, 2EXPTIME-hardness of ω-regular tree satisfiability in ICPDL follows easily from a result in [28], where it is shown that satisfiability in IPDL over trees is already 2EXPTIME-hard. Therefore, we concentrate on the upper bound for ω-regular tree satisfiability in ICPDL.

We prove containment in 2EXPTIME by an exponential time reduction to the non-emptiness problem for TWAPTAs. The main ingredient of the reduction is a mutual inductive translation of (i) ICPDL formulas into TWAPTAs and (ii) of ICPDL programs into a certain kind of non-deterministic automata (NFAs). The latter resemble standard NFAs on words, but navigate in a complete Σ_E-tree reading symbols from $\{a, \overline{a} \mid a \in \Sigma_E\}$. They can also make conditional ε-transitions, which are executable only if the current tree node is accepted by a given TWAPTA.

Formally, a *non-deterministic finite automaton (NFA) $\mathcal{A}$ over a TWAPTA $\mathcal{T} = (S, \delta, s_0, \mathrm{Acc})$* is a tuple $(Q, p_0, q_0, \rightarrow_{\mathcal{A}})$, where Q is a finite set of *states*, p_0 and q_0 are two *selected states*, and $\rightarrow_{\mathcal{A}}$ is a set of labeled-transitions of the following form, where $q, q' \in Q$ and $a \in \Sigma_E$:

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

Transitions of the third kind are called *test transitions*. NFAs define binary relations on the set of nodes of a complete Σ_N-labeled Σ_E-tree. To make this explicit, let $T \in \mathsf{tree}(\Sigma_N, \Sigma_E)$ and define $\Rightarrow_{\mathcal{A},T} \subseteq (\Sigma_E^* \times Q) \times (\Sigma_E^* \times Q)$ as the smallest relation such that for all $u \in \Sigma_E^*$, $a \in \Sigma_E$, $p, q \in Q$, and $s \in S$, we have

$$(u, p) \Rightarrow_{\mathcal{A},T} (ua, q) \text{ if } p \xrightarrow{a}_{\mathcal{A}} q, \tag{4}$$

$$(ua, p) \Rightarrow_{\mathcal{A},T} (u, q) \text{ if } p \xrightarrow{\overline{a}}_{\mathcal{A}} q, \tag{5}$$

$$(u, p) \Rightarrow_{\mathcal{A},T} (u, q) \text{ if } p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q \text{ and } (T, u) \in [\![\mathcal{T}, s]\!]. \tag{6}$$

Define

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

When considering an NFA over a certain 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.

3.2. From ICPDL to automata. Fix a finite set of atomic propositions $\mathsf{P} \subseteq \mathbb{P}$ and atomic programs $\mathsf{A} \subseteq \mathbb{A}$ over which ICPDL formulas and programs are built. For ICPDL formulas ψ and programs π, let

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

The aim of this section is to show how to convert

- each formula ψ into a TWAPTA $\mathcal{T}(\psi)$ such that $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$ and
- each program π into a TWAPTA $\mathcal{T}(\pi)$ and an NFA $\mathcal{A}(\pi)$ over $\mathcal{T}(\pi)$ such that $[\![\mathcal{A}(\pi)]\!] = [\![\pi]\!]$.

All automata work over 2^{P}-labeled A-trees. The construction is by induction on the structure of ψ and π. We start with defining the TWAPTA $\mathcal{T}(\psi)$ for each formula ψ.

If $\psi = p \in \mathsf{P}$, we put $\mathcal{T}(\psi) = (\{s_0\}, \delta, s_0, s_0 \mapsto 1)$, where for all $\gamma \subseteq \mathsf{P}$ we have $\delta(s_0, \gamma) = \texttt{true}$ if $p \in \gamma$ and $\delta(s_0, \gamma) = \texttt{false}$ otherwise. Clearly, $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$.

If $\psi = \neg\theta$, then $\mathcal{T}(\psi)$ is obtained from $\mathcal{T}(\theta)$ 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. [34].

If $\psi = \langle\pi\rangle\theta$, then we have inductively constructed an NFA

$$\mathcal{A}(\pi) = (Q, p_0, q_0, \rightarrow_{\mathcal{A}}) \text{ over a TWAPTA } \mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$$

such that $[\![\pi]\!] = [\![\mathcal{A}(\pi)]\!]$. We have also constructed a TWAPTA

$$\mathcal{T}(\theta) = (S_2, \delta_2, s_2, \mathrm{Acc}_2)$$

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 $\mathcal{T}(\psi) = (S, \delta, p_0, \mathrm{Acc})$ with $S = Q \cup S_1 \cup S_2$. 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 first 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 s_2. Formally, for $q \in Q$ and $\gamma \subseteq \mathsf{P}$, we define

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

with an additional disjunct $\langle s_2, \varepsilon\rangle$ if $q = q_0$. The priority function Acc is defined by setting $\mathrm{Acc}(s) = 1$ if $s \in Q$ and $\mathrm{Acc}(s) = \mathrm{Acc}_i(s)$ whenever $s \in S_i$ with $i \in \{1, 2\}$. It is straightforward to check that $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$.

We now describe the inductive construction of $\mathcal{A}(\pi)$ and $\mathcal{T}(\pi)$ for an ICPDL program π.

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, $[\![\pi]\!] = [\![\mathcal{A}(\pi)]\!]$. 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 $\pi = \psi?$, we can assume that there exists a TWAPTA $\mathcal{T}(\psi) = (S, \delta, s_0, \mathrm{Acc})$ such that $[\![\psi]\!] = [\![\mathcal{T}(\psi)]\!]$. The TWAPTA $\mathcal{T}(\pi)$ is $\mathcal{T}(\psi)$ (without the initial state). The NFA $\mathcal{A}(\pi)$ has only the two selected states p_0 and q_0 with the transition $p_0 \xrightarrow{\mathcal{T}(\pi), s_0} q_0$. 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

$$\mathcal{A}(\pi_i) = (Q_i, p_i, q_i, \rightarrow_{\mathcal{A}(\pi_i)}) \text{ over the TWAPTA } \mathcal{T}(\pi_i) \tag{7}$$

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 tree node u to a tree node v 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 node 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 short-cut these detours. The modified NFA can always travel along the shortest path without any detours, and thus the product construction is applicable.

The 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. This is done in two steps: first, we extend $\mathcal{T}$ into a new TWAPTA $\widehat{\mathcal{T}}$ that can simulate detours made by $\mathcal{A}$. And second, we extend $\mathcal{A}$ into an NFA $\widehat{\mathcal{A}}$ over $\widehat{\mathcal{T}}$ by adding new test transitions that allow to short-cut detours, exploiting the additional capabilities of $\widehat{\mathcal{T}}$.

Let $\mathcal{A}$ and $\mathcal{T}$ be as above, and let $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$. We define a relation $\mathrm{loop}_{\mathcal{A},T} \subseteq \mathsf{A}^* \times Q \times Q$ that describes detours of $\mathcal{A}$. Intuitively, $(u, p, q) \in \mathrm{loop}_{\mathcal{A},T}$ means that, in the tree T, $\mathcal{A}$ 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) for all $u \in \mathsf{A}^*$ and $q \in Q$ we have $(u, q, q) \in \mathrm{loop}_{\mathcal{A},T}$,

(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 $(ua, p, q) \in \mathrm{loop}_{\mathcal{A},T}$,

(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.

LEMMA 3.2. *For all NFAs $\mathcal{A}$ and $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$, we have $(u, p, q) \in \mathrm{loop}_{\mathcal{A},T}$ if and only if $(u, p) \Rightarrow^*_{\mathcal{A},T} (u, q)$.*

PROOF. "only if": Let $n \geq 1$ be the smallest number such that $(u, p, q) \in \text{loop}_{\mathcal{A},T}$ 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 $(u, p, q) \in \text{loop}_{\mathcal{A},T}$ follows from application of rule (i), then $p = q$ and thus clearly $(u, p) \Rightarrow^*_{\mathcal{A},T} (u, q)$. If $(u, p, q) \in \text{loop}_{\mathcal{A},T}$ 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 $(u, p, q) \in \text{loop}_{\mathcal{A},T}$ 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 $(ua, p', q') \in \text{loop}_{\mathcal{A},T}$. Since $(ua, p', q') \in \text{loop}_{\mathcal{A},T}$ 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. The cases when $(u, p, q) \in \text{loop}_{\mathcal{A},T}$ is obtained by finally applying rules (iv) or (v) can be proven similarly as for (iii).

"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 Condition (i) from the definition of $\text{loop}_{\mathcal{A},T}$, 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 Condition (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, $(u, p, q_i), (u, q_i, q) \in \text{loop}_{\mathcal{A},T}$. Condition (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 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, $(ua, q_2, q_{n-1}) \in \text{loop}_{\mathcal{A},T}$. Thus, Condition (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, $(v, q_2, q_{n-1}) \in \text{loop}_{\mathcal{A},T}$. Thus, Condition (iv) yields $(u, p, q) \in \text{loop}_{\mathcal{A},T}$. ⊣

Since Conditions (i)–(v) can easily be translated into a TWAPTA, we obtain the following:

LEMMA 3.3. *For every NFA $\mathcal{A} = (Q, q_0, p_0, \rightarrow_{\mathcal{A}})$ over a TWAPTA $\mathcal{T} = (S, \delta, \text{Acc})$, 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 $p, q \in Q$:*

(i) $[\![\widehat{\mathcal{T}}, s]\!] = [\![\mathcal{T}, s]\!]$
(ii) $[\![\widehat{\mathcal{T}}, (p, q)]\!] = \{(T, u) \mid T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A}), u \in \mathsf{A}^*,$ *and* $(u, p) \Rightarrow^*_{\mathcal{A},T} (u, q)\}$
(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 $\delta'((q,q),\gamma) = \mathtt{true}$. If $p, q \in Q$ with $p \neq q$ and $\gamma \subseteq \mathsf{P}$, then we use the definition of $\mathrm{loop}_{\mathcal{A},T}$ 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 3.2, it is not hard to show that $\widehat{\mathcal{T}}$ is as required. $\dashv$

We now show how to modify an NFA $\mathcal{A} = (Q, q_0, p_0, \rightarrow_{\mathcal{A}})$ over a TWAPTA $\mathcal{T}$ into an NFA $\widehat{\mathcal{A}}$ over $\widehat{\mathcal{T}}$ such that $\widehat{\mathcal{A}}$ can short-cut detours via test transitions: $\widehat{\mathcal{A}} = (Q, p_0, q_0, \rightarrow_{\widehat{\mathcal{A}}})$ 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.

Let $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$ and $u, v \in \mathsf{A}^*$. Then $\inf(u,v)$ denotes 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 $\Uparrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ (for an arbitrary NFA $\mathcal{A}$ and tree T) in the same way as $\Rightarrow_{\mathcal{A},T}$, except that clause (4) in the definition of $\Rightarrow_{\mathcal{A},T}$ is dropped. The relation $\Downarrow_{\mathcal{A},T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q))$ is defined analogously, with clause (5) dropped. Thus, the relation $\Uparrow_{\mathcal{A},T}$ (resp. $\Downarrow_{\mathcal{A},T}$) allows the NFA $\mathcal{A}$ only to walk up (resp. down) in the tree and to stay in the same node 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 3.4. *Let $\mathcal{A} = (Q, p_0, q_0, \rightarrow_{\mathcal{A}})$ be an NFA over a TWAPTA $\mathcal{T}$, $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$, and $u, v \in \mathsf{A}^*$. Then the following 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 $(w,p) \Rightarrow^*_{\mathcal{A},T} (w,q)$. Finally, it remains to prove that (i) implies (iii). Assume $(T,u,v) \in [\![\mathcal{A}]\!]$. Then there exist nodes $w_0, \ldots, w_n \in \mathsf{A}^*$ and states $r_0, \ldots, r_n \in Q$ such that

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

Let $y_1, \dots, y_k \in \{w_0, \dots, w_n\}$ be the unique nodes such that, for some $j \in \{1, \dots, k\}$, the following conditions are satisfied:

(a) $y_1 = u$, $y_k = v$,
(b) $y_j = \inf(u, v)$,
(c) for $1 \le i < j$, we have $y_i = y_{i+1} a_i$ for some $a_i \in \mathsf{A}$, and
(d) for $j < i \le k$, we have $y_i = y_{i-1} a_i$ for some $a_i \in \mathsf{A}$.

Define the mapping $\phi : \{1, \dots, k\} \to \{1, \dots, n\}$ such that $\phi(i) = \max\{\ell \mid y_i = w_\ell\}$ for $1 \le i \le k$. There exist states $r'_1, \dots, r'_k \in Q$ such that the run (8) can be factorized as follows:

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

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 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 $(u, p_0) \Uparrow^*_{\widehat{\mathcal{A}},T} (\inf(u, v), r'_j) \Downarrow^*_{\widehat{\mathcal{A}},T} (v, q_0)$. ⊣

We now 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 (7). 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:

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

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

Lemma 3.5. $[\![\mathcal{A}(\pi_1 \cap \pi_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$.

Proof. Since $[\![\pi_1 \cap \pi_2]\!] = [\![\pi_1]\!] \cap [\![\pi_2]\!]$, it suffices to prove that $[\![\mathcal{A}_1]\!] \cap [\![\mathcal{A}_2]\!] = [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$. So let $(T, u, v) \in [\![\mathcal{A}_i]\!]$ for all $i \in \{1, 2\}$. Then Lemma 3.4 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 all $i \in \{1, 2\}$. 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 $(T, u, v) \in [\![\mathcal{A}(\pi_1 \cap \pi_2)]\!]$. On the other hand, 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 all $i \in \{1, 2\}$. By Lemma 3.4, we obtain $(T, u, v) \in [\![\mathcal{A}_i]\!]$ for all $i \in \{1, 2\}$. ⊣

This finishes the inductive translation of ICPDL formulas and programs into automata.

3.3. 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 ICPDL later on, it is useful to carry out a careful analysis that takes into account nestings of intersection. Since the number of nestings of intersection is too coarse for obtaining a tight complexity bounds for loop-CPDL, we will introduce a finer measure that 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 ICPDL program π is defined inductively:

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

Note that $\mathrm{iw}(\theta?) = 1$ although $\theta?$ may contain subprograms of intersection width stictly larger than 1.

For an NFA $\mathcal{A} = (Q, p_0, q_0, \rightarrow_{\mathcal{A}})$ over some TWPATA, we define the size $|\mathcal{A}|$ as $|Q|$. Note that the size of the TWPATA over which $\mathcal{A}$ is defined is not taken into account.

LEMMA 3.6. *For every ICPDL program π, $|\mathcal{A}(\pi)| \leq 2|\pi|^{\mathrm{iw}(\pi)}$.*

PROOF. The proof is by induction on the structure of π.

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

Inductive cases.

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

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

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

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

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

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

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. When speaking of a subprogram of a program or a formula, we also include programs that occur in the scope of a test operator. Moreover, each program is a subprogram of itself. If α is an ICPDL program or an ICPDL formula, then

$$\mathrm{IW}(\alpha) \;=\; \begin{cases} 1 & \text{if } \alpha \text{ has no subprograms} \\ \max\{\mathrm{iw}(\pi) \mid \pi \text{ is a subprogram of } \alpha\} & \text{otherwise.} \end{cases}$$

Note that we may have $\mathrm{IW}(\theta?) > 1$. For the subsequent lemma, we remind the reader that $i(\mathcal{T})$ denotes the index of the TWAPTA $\mathcal{T}$.

Lemma 3.7. *For every ICPDL program π and ICPDL formula ψ:*
- $|\mathcal{T}(\pi)| \leq |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)}$ *and* $i(\mathcal{T}(\pi)) \leq |\pi|$.
- $|\mathcal{T}(\psi)| \leq |\psi| + 8 \cdot |\psi| \cdot |\psi|^{2\cdot \mathrm{IW}(\psi)}$ *and* $i(\mathcal{T}(\psi)) \leq |\psi|$.

Proof. The proof is by mutual induction on the structure of ψ and π.

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 $i(\mathcal{T}(\pi)) = 1 = |\pi|$.

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

Inductive cases.

Let $\psi = \neg\theta$. The standard complementation construction yields $|\mathcal{T}(\psi)| = |\mathcal{T}(\theta)|$ and $i(\mathcal{T}(\psi)) = i(\mathcal{T}(\theta)) + 1 \leq |\theta| + 1 = |\psi|$.

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

$$\begin{aligned} |\mathcal{T}(\psi)| &= |\mathcal{A}(\pi)| + |\mathcal{T}(\pi)| + |\mathcal{T}(\theta)| \\ &\leq 2 \cdot |\pi|^{\mathrm{iw}(\pi)} + |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)} + |\theta| + 8 \cdot |\theta| \cdot |\theta|^{2\cdot \mathrm{IW}(\theta)} \\ &\leq |\psi| + 8 \cdot |\psi|^{\mathrm{IW}(\psi)} + 8 \cdot |\pi| \cdot |\psi|^{2\cdot \mathrm{IW}(\psi)} + 8 \cdot |\theta| \cdot |\psi|^{2\cdot \mathrm{IW}(\psi)} \\ &= |\psi| + 8 \cdot (1 + |\pi| + |\theta|) \cdot |\psi|^{2\cdot \mathrm{IW}(\psi)} \\ &= |\psi| + 8 \cdot |\psi| \cdot |\psi|^{2\cdot \mathrm{IW}(\psi)}. \end{aligned}$$

Moreover, $i(\mathcal{T}(\psi)) = \max\{i(\mathcal{T}(\pi)), i(\mathcal{T}(\theta))\} \leq \max\{|\pi|, |\theta|\} \leq |\psi|$.

Let $\pi = \theta?$. Then $\mathcal{T}(\pi) = \mathcal{T}(\theta)$ and, by induction, $|\mathcal{T}(\pi)|$ and $i(\mathcal{T}(\pi))$ are as required.

Let $\pi = \pi_1^*$. Then $\mathcal{T}(\pi) = \mathcal{T}(\pi_1)$ and, by induction, $|\mathcal{T}(\pi)|$ and $i(\mathcal{T}(\pi))$ are as required.

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

$$\begin{aligned} |\mathcal{T}(\pi)| &= |\mathcal{T}(\pi_1)| + |\mathcal{T}(\pi_2)| \\ &\leq |\pi_1| + 8 \cdot |\pi_1| \cdot |\pi_1|^{2\cdot \mathrm{IW}(\pi_1)} + |\pi_2| + 8 \cdot |\pi_2| \cdot |\pi_2|^{2\cdot \mathrm{IW}(\pi_2)} \\ &\leq |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)}. \end{aligned}$$

Moreover, $i(\mathcal{T}(\pi)) = \max\{i(\mathcal{T}(\pi_1)), i(\mathcal{T}(\pi_2))\} \leq \max\{|\pi_1|, |\pi_2|\} \leq |\pi|$.

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

$$\begin{aligned}
|\mathcal{T}(\pi)| &= |\mathcal{T}(\pi_1)| + |\mathcal{T}(\pi_2)| + |\mathcal{A}(\pi_1)|^2 + |\mathcal{A}(\pi_2)|^2 \\
&\leq |\pi_1| + 8 \cdot |\pi_1| \cdot |\pi_1|^{2\cdot \mathrm{IW}(\pi_1)} + |\pi_2| + 8 \cdot |\pi_2| \cdot |\pi_2|^{2\cdot \mathrm{IW}(\pi_2)} + \\
&\quad 4 \cdot |\pi_1|^{2\cdot \mathrm{IW}(\pi_1)} + 4 \cdot |\pi_2|^{2\cdot \mathrm{IW}(\pi_2)} \\
&\leq |\pi| + 8 \cdot |\pi_1| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)} + 8 \cdot |\pi_2| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)} + 8|\pi|^{2\cdot \mathrm{IW}(\pi)} \\
&= |\pi| + 8 \cdot (|\pi_1| + |\pi_2| + 1)|\pi|^{2\cdot \mathrm{IW}(\pi)} \\
&= |\pi| + 8 \cdot |\pi| \cdot |\pi|^{2\cdot \mathrm{IW}(\pi)}.
\end{aligned}$$

Finally, $i(\mathcal{T}(\pi))$ can be estimated in the same way as for $\pi = \pi_1 \cup \pi_2$. $\dashv$

In order to apply Theorem 3.1 we also need a bound for the size of the transition function δ of the TWAPTA $\mathcal{T}(\psi)$. A straightforward estimate together with Lemma 3.7 implies that $|\delta| \leq |\mathsf{A}| \cdot 2^{|\mathsf{P}|} \cdot |\psi|^{\mathcal{O}(\mathrm{IW}(\psi))}$. Since A and P may be restricted to those atomic programs and propositions that occur in ψ, we have $|\mathsf{A}|, |\mathsf{P}| \leq |\psi|$ and hence $|\delta| \leq 2^{|\psi|} \cdot |\psi|^{\mathcal{O}(\mathrm{IW}(\psi))}$

3.4. An upper bound for ω-regular tree satisfiability in ICPDL. The algorithm for deciding ω-regular tree satisfiability in ICPDL works as follows. Given an ICPDL formula φ and a TWAPTA $\mathcal{T}$ over 2^{P}-labeled A-trees, we convert φ into a TWAPTA $\mathcal{T}(\varphi)$ such that $[\![\mathcal{T}(\varphi)]\!] = [\![\varphi]\!]$, as described in Section 3.2. Then we construct a TWAPTA $\mathcal{T}'$ such that $L(\mathcal{T}') = L(\mathcal{T}) \cap L(\mathcal{T}(\varphi))$. For alternating automata, intersection is trivial and $\mathcal{T}'$ can be computed in linear time from $\mathcal{T}$ and $\mathcal{T}(\varphi)$. Clearly, $L(\mathcal{T}) \neq \emptyset$ if and only if there is a tree $T \in L(\mathcal{T})$ with $\varepsilon \in [\![\varphi]\!]_T$. By applying Lemma 3.7 and Theorem 3.1, we obtain the following upper bound on ω-regular tree satisfiability in ICPDL.

THEOREM 3.8. *For a TWAPTA $\mathcal{T}$ with transition function δ and an ICPDL formula φ, we can decide in time* $\exp(|\mathcal{T}| + i(\mathcal{T}) + |\varphi|^{\mathrm{IW}(\varphi)}) \cdot |\delta|^{\mathcal{O}(1)}$ *whether there exists some $T \in L(\mathcal{T})$ such that $\varepsilon \in [\![\varphi]\!]_T$. Hence, ω-regular tree satisfiability is in* 2EXPTIME.

Note that Theorem 3.8 yields a single exponential upper bound if the intersection width of the input formula is bounded by a constant.

§4. Satisfiability for ICPDL is in 2EXPTIME.

We prove that satisfiability in ICPDL can be decided in 2EXPTIME using a reduction to ω-regular tree satisfiability. This result matches the lower bound given for satisfiability in IPDL in [28]. The reduction involves two steps. In Section 4.1, we prove that to decide satisfiability in ICPDL, it suffices to consider only Kripke structures of tree width at most two. In Section 4.2, we then prove that satisfiability over such models can be reduced to ω-regular tree satisfiability.

4.1. Models of tree width two. We start with defining tree decompositions and the tree width of Kripke structures. Let

$$K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$$

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 X (called a *bag*) for all $v \in V$, and the following conditions are satisfied:

1. $\bigcup_{v \in V} B_v = X$,
2. For every transition $x \rightarrow_a y$ of K, there exists $v \in V$ with $x, y \in B_v$, and
3. For every $x \in X$, the set $\{v \in V \mid x \in B_v\}$ is a connected subset in T.

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.

For the reduction to ω-regular tree satisfiability, it will be more convenient to work with a special kind of tree decomposition called good. A tree decomposition $(T, (B_v)_{v \in V})$ with $T = (V, E)$ is *good* if

1. T is an undirected binary tree and
2. if $\{u, v\} \in E$, then $B_v \subseteq B_u$ or $B_u \subseteq B_v$.

Only countable Kripke structures can have a good tree decomposition of countable width. This is no problem since ICPDL inherits a Löwenheim-Skolem theorem from least fixed point logic [16, 21], of which it is a fragment. We call a tree decomposition $(T, (B_v)_{v \in V})$ countable if V is countable.

Lemma 4.1. *If K has a countable tree decomposition of width k, then K also has a good tree decomposition of width k.*

Proof. Let $(T, (B_v)_{v \in V})$ be a tree decomposition for K of width k with V countable. We show how to convert $(T, (B_v)_{v \in V})$ into a good tree decomposition, using two steps. First, every edge $\{u, v\}$ of T such that neither $B_u \subseteq B_v$ nor $B_v \subseteq B_u$ is replaced with the two edges $\{u, w\}$ and $\{w, v\}$, where w is a new node. The bag B_w is $B_u \cap B_v$. Second, 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 $\{u_1, u_2\}, \{u_2, u_3\}, \ldots$ of length ℓ, where $u_1 = u$ and $u_2, u_3, \ldots$ are new nodes. Each edge $\{u, v_i\}$ is replaced with $\{u_i, v_i\}$ and the nodes u_i all receive the same bag as u. ⊣

The goal of this section is to prove the following theorem.

Theorem 4.2. *Let K be a Kripke structure and let x_0 be a world of K. Then there exists a Kripke structure $K^\oplus$ of tree width at most two and a world $x_0^\oplus$ of $K^\oplus$ such that*

- *for every ICPDL formula φ we have $x_0 \in [\![\varphi]\!]_K$ iff $x_0^\oplus \in [\![\varphi]\!]_{K^\oplus}$;*
- *if K is countable, then $K^\oplus$ has a countable tree decomposition of width at most two.*

For the rest of this section, fix a Kripke structure

$$K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$$

and a world $x_0 \in X$. To prove Theorem 4.2, we start with inductively defining an undirected tree $T = (V, E)$ together with a node labeling $t_v \in X \cup X^2 \cup X^3$, for each $v \in V$. Based on this tree, we will define the Kripke structure $K^\oplus$ from Theorem 4.2. We will also use T to give a tree decomposition of $K^\oplus$ of width at most two.

The construction of T is by exhaustive application of the following rules:

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

2. If $v \in V$ and $t_v = x \in X$, then for every $y \in X$ add a child w of v and set $t_w = (x, y)$.
3. If $v \in V$ and $t_v = (x, y)$, then add
   - for every $z \in X$ a child w of v with $t_w = (x, z, y)$ and
   - a child w' with $t_{w'} = y$.
4. If $v \in V$ and $t_v = (x, z, y)$, then add children w_1 and w_2 with $t_{w_1} = (x, z)$ and $t_{w_2} = (z, y)$.

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 $(v, x) \in V \times X$ such that t_v is or contains x. We denote by P_T the set of places of T and define $\approx$ to be the smallest equivalence relation on P_T that contains all pairs of the form $((v, x), (u, x))$ such that $\{v, u\} \in E$. For all $(v, x) \in P_T$, we denote by $[v, x]$ the equivalence class of (v, x) with respect to $\approx$. The following is immediate by construction of T.

Lemma 4.3. *If $v \in V$ and t_v contains $x \in X$, then there exists some $v' \in V$ such that $(v, x) \approx (v', x)$ and $t_{v'} = x$.*

Define the Kripke structure $K^{\oplus} = (X', \{\rightarrow'_a \mid a \in \mathbb{A}\}, \{X'_p \mid p \in \mathbb{P}\})$, where

- $X' = \{[v, x] \mid (v, x) \in P_T\}$,
- $[v, x] \rightarrow'_a [v', y]$ whenever there exists a $u \in V$ such that $(v, x) \approx (u, x)$, $(v', y) \approx (u, y)$, and $x \rightarrow_a y$, and
- $X'_p = \{[v, x] \in X' \mid x \in X_p\}$.

We put $x_0^{\oplus} = [v_0, x_0]$. Intuitively, the construction of the Kripke structure $K^{\oplus}$ can be viewed as a kind of tree-width 2 analogue of the standard unravelling of a Kripke structure into a tree. In particular, our construction also untangles the original structure by duplicating worlds, i.e., a world x of K may be duplicated into two worlds $[v, x] \neq [u, x]$. In contrast to unravelling, the above construction does not produce a tree-shaped structure, but only a Kripke structure of tree width at most two. Indeed, it is not hard to see that $(T, (B_v)_{v \in V})$ with

$$B_v = \{[v, x] \in X' \mid x \text{ occurs in } t_v\}$$

is a tree decomposition of $K^{\oplus}$ of width at most two. Moreover, if X is countable, then the tree T is also countable. Finally, Point (3) of the following lemma shows that $K^{\oplus}$ satisfies the same ICPDL formulas as K.

Lemma 4.4. *For all $(v, x), (u, y) \in P_T$, all programs π, and all formulas φ:*

(1) if $t_v = (x, y)$ and $(x, y) \in [\![\pi]\!]_K$, then $([v, x], [v, y]) \in [\![\pi]\!]_{K^{\oplus}}$;
(2) if $([v, x], [u, y]) \in [\![\pi]\!]_{K^{\oplus}}$, then $(x, y) \in [\![\pi]\!]_K$;
(3) $x \in [\![\varphi]\!]_K$ if and only if $[v, x] \in [\![\varphi]\!]_{K^{\oplus}}$.

Proof. We prove the lemma by mutual induction on the structure of π and φ. For Point (1), we make a case distinction according to the form of π:

- $\pi = a$ for some $a \in \mathbb{A}$: That $t_v = (x, y)$ and $(x, y) \in [\![a]\!]_K$ implies $([v, x], [v, y]) \in [\![a]\!]_{K^{\oplus}}$ follows immediately from the definition of $K^{\oplus}$.
- $\pi = \overline{a}$ for some $a \in \mathbb{A}$: Follows immediately from the definition of $K^{\oplus}$ and the semantics of converse.

- $\pi = \psi?$: By the semantics, $(x, y) \in [\![\pi]\!]_K$ implies $x = y$ and $x \in [\![\psi]\!]_K$. Hence by Point (3) of the induction hypothesis, we obtain $[v, x] \in [\![\psi]\!]_{K^\oplus}$. By the semantics, it follows that $([v, x], [v, y]) \in [\![\pi]\!]_{K^\oplus}$ as required.
- $\pi = \pi_1 \cap \pi_2$: By the semantics, $(x, y) \in [\![\pi_1 \cap \pi_2]\!]_K$ implies $(x, y) \in [\![\pi_i]\!]_K$ for each $i \in \{1, 2\}$. It follows by Point (1) of the induction hypothesis that $([v, x], [v, y]) \in [\![\pi_i]\!]_{K^\oplus}$ for each $i \in \{1, 2\}$. Finally, by the semantics we obtain $([v, x], [v, y]) \in [\![\pi]\!]_{K^\oplus}$.
- $\pi = \pi_1 \cup \pi_2$: One can argue in the same way as in the previous case.
- $\pi = \pi_1 \circ \pi_2$: By the semantics, $(x, y) \in [\![\pi]\!]_K$ implies $(x, z) \in [\![\pi_1]\!]_K$ and $(z, y) \in [\![\pi_2]\!]_K$ for some $z \in X$. By construction of T the node v has a successor w with $t_w = (x, z, y)$ and w has successors w_1 and w_2 with $t_{w_1} = (x, z)$ and $t_{w_2} = (z, y)$. By Point (1) of induction hypothesis we have $([w_1, x], [w_1, z]) \in [\![\pi_1]\!]_{K^\oplus}$ and $([w_2, z], [w_2, y]) \in [\![\pi_2]\!]_{K^\oplus}$. By the semantics and since $(v, x) \approx (w_1, x)$, $(w_1, z) \approx (w, z) \approx (w_2, z)$ and $(w_2, y) \approx (v, y)$, we obtain $([v, x], [v, y]) \in [\![\pi]\!]_{K^\oplus}$.
- $\pi = \chi^*$: We prove that $(x, y) \in [\![\chi]\!]^i_K$ implies $([v, x], [v, y]) \in [\![\chi]\!]^i_{K^\oplus}$ by induction on i. If $i = 0$, then $(x, y) \in [\![\chi]\!]^i_K$ implies $x = y$ and therefore we have $([v, x], [v, y]) \in [\![\chi]\!]^i_{K^\oplus}$. Now assume $(x, y) \in [\![\chi]\!]^i_K$ for some $i > 0$. Hence, there exists some $z \in X$ such that $(x, z) \in [\![\chi]\!]^{i-1}_K$ and $(z, y) \in [\![\chi]\!]_K$. By construction of T node v has a successor w with $t_w = (x, z, y)$ and w has successors w_1 and w_2 with $t_{w_1} = (x, z)$ and $t_{w_2} = (z, y)$. By the inner induction hypothesis we have $([w_1, x], [w_1, z]) \in [\![\chi]\!]^{i-1}_{K^\oplus}$. By Point (1) of the outer induction hypothesis we have $([w_2, z], [w_2, y]) \in [\![\chi]\!]_{K^\oplus}$. By the semantics and since $(v, x) \approx (w_1, x)$, $(w_1, z) \approx (w, z) \approx (w_2, z)$ and $(w_2, y) \approx (v, y)$, we obtain $([v, x], [v, y]) \in [\![\chi]\!]^i_{K^\oplus}$.

To prove Point (2), we also make a case distinction according to the form of π:

- $\pi = a$ for some $a \in \mathbb{A}$: That $([v, x], [u, y]) \in [\![a]\!]_{K^\oplus}$ implies $(x, y) \in [\![a]\!]_K$ follows immediately from the definition of $K^\oplus$.
- $\pi = \overline{a}$ for some $a \in \mathbb{A}$: Follows immediately from the definition of $K^\oplus$ and the semantics of converse.
- $\pi = \psi?$: By the semantics $([v, x], [u, y]) \in [\![\pi]\!]_{K^\oplus}$ implies $[v, x] = [u, y]$ and $[v, x] \in [\![\psi]\!]_{K^\oplus}$. By definition of $\approx$ it follows $x = y$ and by Point (3) of induction hypothesis we have $x \in [\![\psi]\!]_K$. Thus, by the semantics $(x, y) \in [\![\pi]\!]_K$.
- The remaining cases are straightforward by Point (2) of the induction hypothesis and the semantics.

For Point (3), we make a case distinction according to the form of φ:

- $\varphi = p \in \mathbb{P}$: That $x \in [\![p]\!]_K$ if and only if $[v, x] \in [\![p]\!]_{K^\oplus}$ follows immediately from the definition of $K^\oplus$.
- $\varphi = \neg\psi$: Follows easily from the semantics and Point (3) of the induction hypothesis.
- $\varphi = \langle\pi\rangle\psi$: Let us first prove the "if"-direction. Assume $[v, x] \in [\![\varphi]\!]_{K^\oplus}$. Then there exists some $[u, y] \in X'$ such that $([v, x], [u, y]) \in [\![\pi]\!]_{K^\oplus}$ and $[u, y] \in [\![\psi]\!]_{K^\oplus}$. By Point (2) of induction hypothesis we have $(x, y) \in [\![\pi]\!]_K$ and by Point (3) of induction hypothesis we have $y \in [\![\psi]\!]_K$. Thus, we obtain $x \in [\![\varphi]\!]_K$.

Let us show the "only if"-direction. Assume $x \in [\![\varphi]\!]_K$. Then there exists some $y \in X$ such that $(x, y) \in [\![\pi]\!]_K$ and $y \in [\![\psi]\!]_K$. Since t_v contains x, by Lemma 4.3, there exists a node $v' \in V$ such that $(v, x) \approx (v', x)$ and $t_{v'} = x$. By construction of T node v' has a successor w with $t_w = (x, y)$. Applying Point (1) of induction hypothesis yields $([w, x], [w, y]) \in [\![\pi]\!]_{K^\oplus}$. Too, Point (3) of induction hypothesis implies $[w, y] \in [\![\psi]\!]_{K^\oplus}$. Since moreover $(v, x) \approx (v', x) \approx (w, x)$, we obtain $[v, x] \in [\![\varphi]\!]_{K^\oplus}$.

This finishes the proof of Lemma 4.4 and hence of Theorem 4.2. $\dashv$

4.2. Reduction to ω-regular tree satisfiability. To reduce satisfiability in ICPDL to ω-regular tree satisfiability, we show how to translate an ICPDL formula φ into an ICPDL formula φ' and a TWAPTA $\mathcal{T}$ such that φ is satisfiable if and only if φ' is satisfiable with respect to $\mathcal{T}$ in the sense of ω-regular tree satisfiability. The formula φ' uses atomic propositions and atomic programs

$$\mathsf{P} = \{t\} \uplus \mathsf{prop}(\varphi) \uplus (\{0,1,2\} \times \mathsf{prog}(\varphi) \times \{0,1,2\}) \quad \text{and} \quad \mathsf{A} = \{a, b, 0, 1, 2\},$$

where $\mathsf{prop}(\varphi) = \{p \in \mathbb{P} \mid p \text{ occurs in } \varphi\}$ and $\mathsf{prog}(\varphi) = \{a \in \mathbb{A} \mid a \text{ occurs in } \varphi\}$. The TWAPTA $\mathcal{T}$ works over 2^{P}-labeled A-trees. Intuitively, each tree T accepted by $\mathcal{T}$ encodes a Kripke structure K together with a good tree decomposition of K of width at most two, and T is a model of φ' if and only if K is a model of φ.

We first describe the mentioned encoding on an intuitive level. Let K be a Kripke structure and $(U, (B_v)_{v \in V})$ a good tree decomposition of K of width at most two, with $U = (V, E)$. Assume w.l.o.g. that $V \subseteq \{a, b\}^*$, with ε the root of the tree U and for each $v \in V$, va and vb the children of v. The tree T has roughly the structure of U. In particular, each node $v \in V$ is described by the same node v in T, together with additional children $v0$, $v1$, and $v2$ that v has in T. Intuitively, we think of each node v in U as providing three slots which can be empty or filled with a world of the Kripke structure K, i.e. the three slots correspond to an ordered representation of the bag B_v. The additional successors $v0, v1, v2$ of v in T describe these three slots. This explains our choice of A. When slot vi ($i \in \{0, 1, 2\}$) is occupied by a world of K, then vi receives the special label $t \in \mathsf{P}$ in T. Additionally, each node vi is labeled with the same atomic propositions as the world in K that it represents (if any). 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). We now formally define the described encoding. A complete 2^{P}-labeled A-tree T is called *valid* if the following holds for all $v \in \mathsf{A}^*$:

- if $v \in \{a, b\}^*$ and $i \in \{0, 1, 2\}$, then $T(vi) = \emptyset$ or $\{t\} \subseteq T(vi) \subseteq \{t\} \cup \mathbb{P}$; set $B_v = \{i \mid t \in T(vi)\}$;
- if $v \in \{a, b\}^*$, then $T(v) \subseteq B_v \times \mathsf{prog}(\varphi) \times B_v$;
- if $v \in \{a, b\}^*$ and $c \in \{a, b\}$, then $B_v \subseteq B_{vc}$ or $B_{vc} \subseteq B_v$;
- if $v \notin \{a, b\}^* \cup \{a, b\}^*\{0, 1, 2\}$, then $T(v) = \emptyset$.

Let T be a valid 2^{P}-labeled A-tree. We now make precise the Kripke structure $K(T)$ over $\mathsf{prop}(\varphi)$ and $\mathsf{prog}(\varphi)$ that is described by T. Define a set of *places*

$$P_T = \{u \in \{a, b\}^*\{0, 1, 2\} \mid t \in T(u)\}$$

and let $\sim$ be the smallest equivalence relation on P_T which contains all pairs $(vi, vci) \in P_T \times P_T$, where $v \in \{a,b\}^*$, $c \in \{a,b\}$, and $0 \le i \le 2$. For $u \in P_T$, we use $[u]$ to denote the equivalence class of u with respect to $\sim$. Now set

$$K(T) = (X, \{\rightarrow_\gamma \mid \gamma \in \mathsf{prog}(\varphi)\}, \{X_p \mid p \in \mathsf{prop}(\varphi)\}),$$

where

$$\begin{aligned} X &= \{[u] \mid u \in P_T\}, \\ \rightarrow_\gamma &= \{([vi],[vj]) \mid v \in \{a,b\}^*, (i,\gamma,j) \in T(v)\}, \text{ and} \\ X_p &= \{[u] \in X \mid p \in T(u)\}. \end{aligned}$$

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 Section 3: 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 following lemma, which is easy to prove, establishes the correctness of our encoding.

LEMMA 4.5. *If K is a countable Kripke structure that has a good tree decomposition of at most width two, then there exists a valid $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$ such that K is isomorphic to $K(T)$. Conversely, $K(T)$ has tree width at most two for every valid $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$.*

Recall that our aim is to convert the ICPDL formula φ whose satisfiability is to be decided into an ICPDL formula φ' over P and A and a TWAPTA $\mathcal{T}$. The TWAPTA $\mathcal{T}$ is defined such that it accepts the set of all valid 2^{P}-labeled A-trees. Such a TWAPTA is easily designed, and details are left to the reader. To define the formula φ', we first introduce the auxiliary program

$$\pi^1_\sim = \bigcup_{i \in \{0,1,2\}} t? \circ \overline{i} \circ (a \cup b \cup \overline{a} \cup \overline{b}) \circ i \circ t?$$

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

- every atomic program $\gamma \in \mathsf{prog}(\varphi)$ by
$$\widehat{\gamma} \quad = \bigcup_{i,j \in \{0,1,2\}} \pi_\sim \circ \overline{i} \circ (i,\gamma,j)? \circ j \circ \pi_\sim \qquad \text{and}$$
- every atomic proposition $p \in \mathsf{prop}(\varphi)$ by $\widehat{p} = \langle \pi_\sim \rangle p$.

Observe that the definitions of $\widehat{\gamma}$ and $\widehat{p}$ directly reflect the definition of $K(T)$. The proof of the following lemma is straightforward by induction on the structure of ψ and π. The base case is easy by definition of $\sim$ and $\pi_\sim$ and $K(T)$, and the inductive step is simple. Details are left to the reader.

LEMMA 4.6. *For all subformulas ψ of φ, subprograms π of φ, valid trees $T \in \mathsf{tree}(2^{\mathsf{P}}, \mathsf{A})$, and places $u, v \in P_T$, we have:*

1. *$u \in [\![\widehat{\psi}]\!]_T$ if and only if $[u] \in [\![\psi]\!]_{K(T)}$;*
2. *$(u,v) \in [\![\widehat{\pi}]\!]_T$ if and only if $([u],[v]) \in [\![\pi]\!]_{K(T)}$.*

Now define $\varphi' = \langle (a \cup b)^* \circ (0 \cup 1 \cup 2) \circ t? \rangle \widehat{\varphi}$.

Lemma 4.7. *φ is satisfiable if and only if φ' is satisfiable with respect to $\mathcal{T}$. Moreover,* $\mathrm{IW}(\varphi') = \mathrm{IW}(\varphi)$.

Proof. The statement on the intersection width is obvious. Now, assume that φ is satisfiable. From the results in [16, 21] and the obvious embedding of ICPDL into least fixed point logic, it follows that φ has a countable model. Hence, by Theorem 4.2, φ has a model K with a countable tree decomposition of width at most two. By Lemma 4.1 this tree decomposition can be assumed to be good. By Lemma 4.5, K is of the form $K(T)$ for some valid tree T. Point (1) of Lemma 4.6 and the definition of φ' imply that T is a model of φ', i.e., φ' is satisfiable with respect to $\mathcal{T}$.

For the other direction assume that φ' is satisfiable with respect to $\mathcal{T}$ and let T be a valid tree such that $\varepsilon \in [\![\varphi']\!]_T$. Then, by point (1) of Lemma 4.6, $K(T)$ is a model of φ, i.e., φ is satisfiable. $\dashv$

From Theorem 3.8 and Lemma 4.7, we get the main result of this section.

Theorem 4.8. *Satisfiability in ICPDL is* 2EXPTIME*-complete. For every constant c, satisfiability in* $\{\varphi \in \mathit{ICPDL} \mid \mathrm{IW}(\varphi) \leq c\}$ *is* EXPTIME*-complete.*

We have already noted that a loop-CPDL formula φ can be translated into an ICPDL formula φ' by replacing every subformula $\mathrm{loop}(\pi)$ with $\langle \pi \cap \mathtt{true}? \rangle \mathtt{true}$. We claim that $\mathrm{IW}(\varphi') \leq 2$. To see this, let π be an arbitrary program occuring in φ'. We have to show that $\mathrm{iw}(\pi) \leq 2$. By the definition of $\mathrm{iw}(\cdot)$, it suffices to consider the case that the topmost operator in π is an intersection. Thus, $\pi = \rho \cap \mathtt{true}?$ and $\mathrm{iw}(\pi) = \mathrm{iw}(\rho)+1$. But by construction of φ', every intersection operator in ρ occurs within a subprogram $\theta?$ of ρ. Hence, $\mathrm{iw}(\rho) = 1$. We obtain:

Corollary 4.9. *Satisfiability in loop-CPDL is* EXPTIME*-complete.*

4.3. Σ^1_1-completeness of IPDL with negation of atomic programs. It is well-known that adding full program negation to PDL results in satisfiability to become undecidable [23], whereas PDL with program negation restricted to atomic programs remains decidable and EXPTIME-complete [31]. Therefore, it is a natural question whether satisfiability remains decidable when we extend IPDL and ICPDL 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, see e.g. [37].

For a Σ^1_1 upper bound, observe again that from [16, 21] it follows that every satisfiable formula of least fixed point logic has a countable model. Since $\mathrm{IPDL}^{(\neg)}$ is clearly a fragment of least fixed point logic, it remains, for a given $\mathrm{IPDL}^{(\neg)}$ formula φ, to give an existential second-order formula of arithmetic over the naturals that expresses the existence of a countable model of φ. However, the latter is straightforward, see for example Section 3.4 in [23].

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 [22], 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.

We use IPDL$^{(\neg)}$ to denote the extension of IPDL with negation of atomic programs, which we write as $\neg a$ ($a \in \mathbb{A}$). The semantics of the new constructor is defined in the obvious way, i.e., $[\![\neg a]\!]_K = (X \times X) \setminus [\![a]\!]_K$ for each Kripke structure K with set of worlds X. 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 of T as atomic propositions. The formula $\varphi_{\mathcal{S}}$ is the conjunction consisting of the following 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:

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

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

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

(c) the programs a_x and a_y are confluent:

$$[(a_x \cup a_y)^*]\;[(a_y \circ a_x) \cap (a_x \circ \neg a_y)]\texttt{false}$$

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

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

(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 4.10. *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 $K = (X, \{\rightarrow_{a_x}, \rightarrow_{a_y}\}, \{X_t \mid t \in T\})$ 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 $\pi : \mathbb{N} \times \mathbb{N} \rightarrow X$, 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 worlds x_i ($i \geq 0$) such that $x_0 = \pi(0, 0)$, $(x_i, x_{i+1}) \in [\![a_y]\!]_K$ for all $i \geq 0$ and such that $x_j \in X_{t_r}$ for infinitely many $j \geq 0$ (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 $x \in X$ such that $(\pi(i, j), x) \in [\![a_x]\!]_K$ and set $\pi(i+1, j) = x$. 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 $\tau : \mathbb{N} \times \mathbb{N} \to T$ 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 $\mathcal{S}$. ⊣

We have thus established the following result.

THEOREM 4.11. *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 [23], using a (trivial) reduction of the equivalence problem for the algebra of binary relations with complementation.

§5. Infinite state model checking. We introduce pushdown systems (PDSs) and basic process algebras (BPAs) as formalisms for representing infinite Kripke structures and, based on this, the infinite-state model checking problem for ICPDL. We then prove upper complexity bounds by reduction to ω-regular tree satisfiability. Matching lower bounds are established in Section 6.

In the context of model checking, we generally use Kripke structures without atomic propositions. This simplifies notation and can be done w.l.o.g. since an atomic proposition p is easily simulated using a reflexive loop of a new atomic program a_p, which can then be tested by the formula $\mathrm{loop}(a_p)$. In the following, $\mathsf{A} \subseteq \mathbb{A}$ always denotes a finite set of atomic programs.

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 $\xrightarrow{a}_{\mathcal{Y}} \subseteq Q \times \Gamma \times Q \times \Gamma^*$ is a finite set of *rewriting rules*, for each $a \in \mathsf{A}$. For better readability, we write $(q, b, q', v) \in \xrightarrow{a}_{\mathcal{Y}}$ in infix notation as $qb \xrightarrow{a}_{\mathcal{Y}} q'v$. Intuitively, a PDS describes a (unique) 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 $K(\mathcal{Y}) = (Q\Gamma^*, \{\to_a \mid a \in \mathsf{A}\})$, where

$$\to_a = \{(qbw, q'vw) \mid w \in \Gamma^*, qb \xrightarrow{a}_{\mathcal{Y}} q'v\}.$$

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

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 $a \in \mathsf{A}$. Intuitively, a BPA is simply a PDS with a single state. Similary to the case of PDSs, we write $b \xrightarrow{a}_{\mathcal{X}} v$ rather than $(b, v) \in \xrightarrow{a}_{\mathcal{X}}$. The Kripke structure defined by a BPA $\mathcal{X}$ is the obvious one, i.e., $K(\mathcal{X}) = (\Gamma^*, \{\to_a \mid a \in \mathsf{A}\})$, where

$$\to_a = \{(bw, vw) \mid w \in \Gamma^*, b \xrightarrow{a}_{\mathcal{X}} v\}.$$

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

The *model checking problem* for pushdown systems is defined as follows:

		BPA/PDS/PRS
loop-(C)PDL	**data**	EXPTIME-complete
	expression	EXPTIME-complete
	combined	EXPTIME-complete
I(C)PDL	**data**	EXPTIME-complete
	expression	2EXPTIME-complete
	combined	2EXPTIME-complete

TABLE 1. Infinite state model checking in I(C)PDL and loop-I(C)PDL.

INPUT: A PDS $\mathcal{Y}$, a world $w_0 \in K(\mathcal{Y})$, and an ICPDL formula φ.
QUESTION: $w_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$?

The model checking problem for basic process algebras is defined analogously.

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

EXAMPLE 5.1. *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 $i \in \{1, 2\}$. The systems $\mathcal{Y}_1$ and $\mathcal{Y}_2$ can synchronize by entering a common configuration, and thus we assume that $Q_1 \cap Q_2 \neq \emptyset$. To distinguish steps 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*

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

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. This property is expressed by the following 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.

When analyzing the complexity of model checking, we distinguish combined complexity (where both the formula and the system is part of the input), data complexity (where the formula is assumed to be fixed), and expression complexity (where the system is assumed to be fixed). Our results are summarized in Table 1, where we also mention *prefix recognizable systems* (PRSs) [5, 25]. These systems generalize pushdown systems. It is straightforward to extend our upper bounds from PDAs to PRSs. Note also that the converse operator has no impact on the complexity of model-checking.

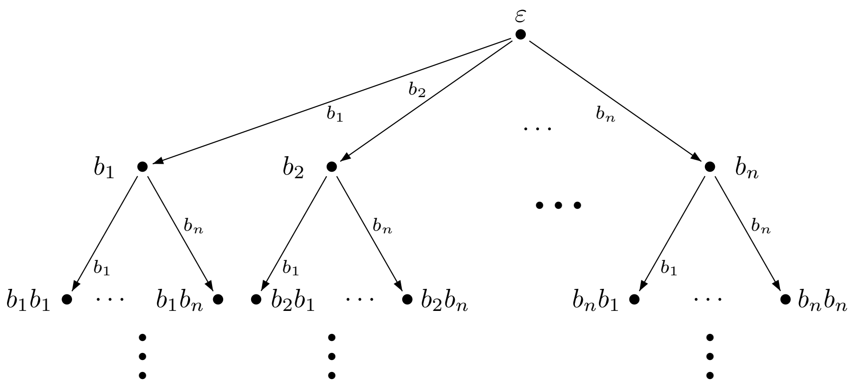

FIGURE 1. The complete B-tree

5.1. Upper bounds for infinite state model checking. In this section, we reduce model checking of PDSs in ICPDL to ω-regular tree satisfiability in ICPDL.

Let φ be an ICPDL formula, $\mathcal{Y} = (Q, \Gamma, \{\xrightarrow{a}_{\mathcal{Y}} \mid a \in \mathsf{A}\})$ a PDS, and $w_0 \in Q\Gamma^*$ a world of $K(\mathcal{Y})$. We want to check whether $w_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$. The basic idea is to use a reduction to ω-regular tree satisfiability in ICPDL over a single tree T_0. More precisely, we set $\mathsf{B} = Q \cup \Gamma$, and then T_0 is the complete B-tree in which $T_0(v) = \emptyset$, for each $v \in \mathsf{B}^*$. This tree corresponds to the Kripke structure $(\mathsf{B}^*, \{\rightarrow_b \mid b \in \mathsf{B}\})$, where $\rightarrow_b = \{(u, ub) \mid u \in \mathsf{B}^*\}$ for each $b \in \mathsf{B}$. It is displayed in Figure 1.

For a word $w = b_1 \cdots b_n \in \mathsf{B}^*$, let $\overleftarrow{w} = b_n \cdots b_1$ be the result of *reversing* it. Each configuration qv of $\mathcal{Y}$ is represented as $\overleftarrow{qv}$ in T_0, as this allows us to travel between configurations in an easier way. Let φ' be the ICPDL formula that is obtained from φ by replacing every occurence of $a \in \mathsf{A}$ with

$$\bigcup_{qc \xrightarrow{a}_{\mathcal{Y}} pv} \overline{q}\,\overline{c}\,\overleftarrow{v}\; p.$$

Clearly, φ' can be interpreted in T_0. It is not hard to verify that $w_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$ if and only if $\varepsilon \in [\![\langle \overleftarrow{w_0} \rangle \varphi']\!]_{T_0}$. It remains to construct a TWAPTA $\mathcal{T}$ such that $L(\mathcal{T}) = \{T_0\}$, which is a triviality. Thus, $w_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$ if and only if there is a $T \in L(\mathcal{T})$ with $\varepsilon \in [\![\langle \overleftarrow{w_0} \rangle \varphi']\!]_T$. ¿From Theorem 3.8 we obtain:

THEOREM 5.2. *For a PDS $\mathcal{Y}$, a world $w_0 \in K(\mathcal{Y})$ and an ICPDL formula φ, we can check in time* $\exp((|w_0| + |\varphi| \cdot |\mathcal{Y}|)^{\mathrm{IW}(\varphi)})$ *whether $w_0 \in [\![\varphi]\!]_{K(\mathcal{Y})}$. Hence, model checking PDSs in ICPDL is in* $\mathsf{2EXPTIME}$ *with respect to combined (and thus also expression) complexity, and in* $\mathsf{EXPTIME}$ *with respect to data complexity.*

As a corollary of Theorem 5.2 and our translation of loop-CPDL into ICPDL in (3) we obtain the following.

Corollary 5.3. *Model checking of PDSs in loop-CPDL is in* EXPTIME *with respect to combined complexity.*

§6. Lower bounds. We prove two lower bounds for infinite-state model checking in fragments of ICPDL. The first result establishes EXPTIME-hardness of the data complexity of model checking BPAs in loop-PDL. This is somewhat surprising since, in many modal logics such as PDL and the modal μ-calculus, the data complexity of model checking is in PTIME [46]. The second result states that the expression complexity of model checking BPAs in IPDL is 2EXPTIME-hard. Thus, we obtain tight complexity bounds as summarized in Table 1. Both lower bounds are proved by a reduction of the word problem of space-bounded alternating Turing machines.

6.1. Alternating Turing Machines. An *alternating Turing machine (ATM)* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ where (i) $Q = Q_{\mathrm{acc}} \uplus Q_{\mathrm{rej}} \uplus Q_{\exists} \uplus Q_{\forall}$ is a finite set of *states* Q which is partitioned into *accepting* (Q_{acc}), *rejecting* (Q_{rej}), *existential* ($Q_{\exists}$), and *universal* ($Q_{\forall}$) states, (ii) Γ is a *finite alphabet*, (iii) $\Sigma \subseteq \Gamma$ is the *input alphabet*, (iv) $q_0 \in Q$ is the *initial state*, (v) $\Box \in \Gamma \setminus \Sigma$ is the *blank symbol*, and (vi) the map $\delta : (Q_{\exists} \cup Q_{\forall}) \times \Gamma \to \mathrm{Moves} \times \mathrm{Moves}$, with $\mathrm{Moves} = Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, assigns to every pair $(q, \gamma) \in (Q_{\exists} \cup Q_{\forall}) \times \Gamma$ a pair of *moves*. A *configuration* of $\mathcal{M}$ is a word from $\Gamma^* Q \Gamma^*$, with the usual meaning. Assume a configuration c of $\mathcal{M}$ is in current state $q \in Q_{\exists} \cup Q_{\forall}$ and scans a symbol $\gamma \in \Gamma$. If $\delta(q, \gamma) = (\mu_1, \mu_2)$ and c_i is the successor configuration reached from c by the move μ_i ($i \in \{1, 2\}$), we write $c \vdash_{\mathcal{M}}^{\mu_i} c_i$. We call c_1 the *left successor configuration* and c_2 the *right successor configuration* of c. A configuration c of $\mathcal{M}$ in current state $q \in Q$ is *existential* if $q \in Q_{\exists}$, *universal* if $q \in Q_{\forall}$, and *accepting* if one of the following three conditions holds:

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

An input $w \in \Sigma^*$ is *accepted* by an ATM $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ if and only if $q_0 w$ is an accepting configuration. It follows from this definition that acceptance of w by $\mathcal{M}$ is witnessed by the existence of an *accepting computation tree* of $\mathcal{M}$ on w, i.e., a finite tree whose nodes are accepting configurations and such that the root is the initial configuration of $\mathcal{M}$ on w, each node that is an existential configuration has one successor configuration as a child node, and each node that is a universal configuration has both of its successor configurations as child nodes. By $L(\mathcal{M}) \subseteq \Sigma^*$, we denote the *language* accepted by $\mathcal{M}$. It is well known that alternating polynomial space equals EXPTIME and alternating exponential space equals 2EXPTIME [7]. Whenever we talk about hardness, we mean hardness with respect to logarithmic space reductions.

6.2. Data Complexity in loop-PDL. We start with a summary of the main ideas behind the proof, which follows [45]. Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ be a $p(n)$-space bounded ATM ($p(n)$ a polynomial) whose word problem is EXPTIME-hard, and $w \in \Sigma^*$ 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 worlds of $K(\mathcal{X})$

represent sequences of configurations of $\mathcal{M}$ of length at most $p(n)$, separated by certain markers. Then, φ verifies the existence of an accepting computation tree of $\mathcal{M}$ on w by traversing the tree in a depth-first and left-to-right fashion, while walking through $K(\mathcal{X})$.

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. Such a sequence represents the current state of the traversal of the computation tree. In particular, a marker $L(\mu_1,\mu_2)$ on top of a universal configuration means that we are currently exploring the left subtree of that configuration, and $R(\mu_1,\mu_2)$ refers to the right subtree.

The main ingredient to the formula φ is a program "traverse", which implements the individual steps of the traversal. 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 carries out the following basic steps:

- If the top of the stack consists of a universal configuration c in current state $q \in Q_\forall$ and scanning a symbol γ, then the direction marker $L(\mu_1,\mu_2)$ is pushed on the stack where $\delta(q,\gamma) = (\mu_1,\mu_2)$, and on top of it the left successor configuration c' of c.
- If the top of the stack consists of an existential configuration c in current state $q \in Q_\forall$ and scanning a symbol γ, then we non-deterministically push a direction marker $E(\mu)$ and on top of it a successor configuration c' of c, where μ is the move from $\delta(q,\gamma)$ executed to reach c' from c.
- If the top of the stack consists of a configuration with current state from Q_{acc}, then the configuration can be popped.
- 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 the right successor configuration of c on top of it.
- 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.

The traversal should end at the empty stack.

Theorem 6.1. *There exists a loop-PDL formula φ such that, given a basic process algebra $\mathcal{X}$, it is* EXPTIME*-hard to decide whether $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$.*

Proof. Let $\mathcal{M} = (Q,\Sigma,\Gamma,q_0,\delta,\Box)$ be a $p(n)$-space bounded ATM ($p(n)$ a polynomial) with an EXPTIME-hard word problem. Let $w = w_1w_2\cdots w_n \in \Sigma^*$ 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 $\Omega_N = \{a(i) \mid a \in \Omega, 0 \leq i \leq N\}$. A configuration $c = a_0 a_1 \cdots a_{i-1} q a_{i+1} \cdots a_N \in \bigcup_{0 \leq i \leq N-1} \Gamma^i Q \Gamma^{N-i}$ 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 rules:

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

Observe that not all worlds in $\mathrm{K}(\mathcal{X})$ 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. These are defined next. In the following, for $X \subseteq \mathrm{Dir} \cup \Omega$, we use pop_X to denote $\bigcup_{x \in X} \overline{x}$.

- The program $\overline{\text{conf}} = \text{pop}_\Omega^* \circ \text{pop}_N$ pops the top configuration from the stack.
- The program conf pushes a potential configuration onto the stack, i.e., a string of the form $a_0(0) a_1(1) \cdots a_N(N)$, $a_i \in \Omega$:
$$\text{conf} \quad = \quad \text{check}_{\mathrm{Dir}} \circ \text{push}^* \circ \text{check}_0$$
- 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 has the following form, where the top of the stack is to the left:
$$a_k(k) a_{k+1}(k+1) \cdots a_N(N) d a'_0(0) \cdots a'_N(N)$$
with $a_i, a'_i \in \Omega$. For $a, b \in \Omega$, we give a program $\text{test}_{a,b}$ that is executable only if $a_k = a$ and $a'_k = b$, and that pops $a_k(k)$:
$$\text{test}_{a,b} = (a/b) \circ [\text{loop}\,(\text{pop}_\Omega^* \circ \text{pop}_{\mathrm{Dir}} \circ \text{pop}_\Omega^* \circ \text{copy} \circ \text{shift}^*)]? \circ \overline{b}$$

- For $\mu \in \text{Moves}$, the program check_μ is executable only if the top of the stack has the form $c'dc$, where c and c' are configurations, $d \in \text{Dir}$, and $c \vdash^{\mu}_{\mathcal{M}} c'$. Execution of check_μ leaves the stack unchanged. We only treat the case $\mu = (p, b, \leftarrow)$ explicitly:

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

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

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

- The program final tests if the state of the top configuration is from Q_{acc}:

$$\text{final} \quad = \quad \bigcup_{\substack{q\in Q_{\text{acc}}\\ a\in\Gamma}} \text{scan}_{q,a}$$

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

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

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 $\varepsilon \in [\![\varphi]\!]_{\mathcal{K}(\mathcal{X})}$. ⊣

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

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

and replacing programs $\text{loop}(\alpha)?$ with $\alpha \cap \text{loopp}$, one can modify the above construction such that φ is a test-free IPDL formula. Hence, model checking BPAs in test-free IPDL is also EXPTIME-hard with respect to data complexity.

6.3. Expression Complexity in IPDL. The proof of the 2EXPTIME lower bound is similar to the one given in the previous section. In particular, in the relevant states of the BPA, the stack consists of a sequence of configurations separated by direction markers. Also, we again ensure the existence of an accepting configuration 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 2EXPTIME-hardness, and thus reduce the word problem of *exponentially* space bounded alternating Turing machines. This makes it necessary to encode a configuration $c = a_0 a_1 \cdots a_{i-1} q a_{i+1} \cdots a_{2^N-1}$ in a different way, namely as the word

$$a_0[0]a_1[1] \cdots a_{i-1}[i-1]q[i]a_{i+1}[i+1] \cdots a_{2^N-1}[2^N-1], \tag{9}$$

where $[k]$ denotes the binary representation of $k \le 2^N - 1$ 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 [28] and [45]. 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 6.2. *There exists a basic process algebra $\mathcal{X}$ such that, given a test-free IPDL formula φ, it is* 2EXPTIME*-hard to decide whether $\varepsilon \in [\![\varphi]\!]_{K(\mathcal{X})}$.*

Proof. Let $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta, \Box)$ be a fixed $(2^{p(n)} - 1)$-space bounded ATM ($p(n)$ a polynomial) with a 2EXPTIME-hard word 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 $\varphi = \varphi(w, \mathcal{M})$ 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 $\overline{\Gamma}_{\mathcal{X}}$ 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), and where each world has a reflexive transition for the program loopp. The rewrite rules of $\mathcal{X}$ are thus as follows:

$$\begin{array}{rll} \varepsilon \xrightarrow{\quad a \quad}_{\mathcal{X}} a & \text{for } a \in \Gamma_{\mathcal{X}} \\ a \xrightarrow{\quad \overline{a} \quad}_{\mathcal{X}} \varepsilon & \text{for } a \in \Gamma_{\mathcal{X}} \\ \varepsilon \xrightarrow{\ \mathrm{loopp}\ }_{\mathcal{X}} \varepsilon & \end{array}$$

We now 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 $i < N$, $\overline{\mathrm{pop}_i} = \overline{\{0,1\}}^i$ pops i bits;
  - $\overline{\mathrm{cell}} = \overline{\Omega} \circ \overline{\mathrm{pop}_N}$ 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 $X \subseteq \Gamma_{\mathcal{X}}$, $\overline{X} = \bigcup_{x \in X} \overline{x}$ pushes a single symbol from X onto the stack.
- The program $\overline{\text{inc}}$ 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 $\overline{\text{inc}}$, 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:

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

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

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

  Note that the final composition with $\Gamma_{\mathcal{X}}^*$ and the final intersection with $\overline{\text{cell}}$ are needed to ensure that $\overline{\text{inc}}$ 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 $\overline{\text{conf}}$ pops a configuration from the stack, ensuring that is of the shape (9):

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

- For $a, a' \in \Omega$, we give a program $\text{test}_{a,a'}$ that will be executed only when the top of the stack consists of a suffix of a configuration followed by a direction marker $d \in \text{Dir}$ 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 $\text{test}_{a,a'}$ 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:

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

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

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

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

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

- For $\mu \in \text{Moves}$, the program check_μ is executable if the top of the stack is of the form $c'dc$, where c and c' are configurations, $d \in \text{Dir}$, and $c \vdash^\mu c'$. Execution of check_μ leaves the stack unchanged. We show only the case where $\mu = (p, b, \leftarrow)$:

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

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

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

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

We now define the program traverse:

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

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

$$\text{check}_w = \Bigg(\Big(\overline{q_0} \circ \text{pop}_N \circ (\overline{w_1} \circ \text{pop}_N) \circ \cdots \circ (\overline{w_n} \circ \text{pop}_N) \circ (\overline{\Box} \circ \text{pop}_N)^*$$
$$\cap \; \overline{\text{conf}}\Big) \;\circ\; \Gamma^*_{\mathcal{X}}\Bigg) \cap \text{loopp}$$

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

§7. Conclusion. We have determined the computational complexity of satisfiability and infinite-state model checking in ICPDL and several of its variations. An interesting open problem is the decidability and exact complexity of *finite* satisfiability in I(C)PDL, which differs from unrestricted satisfiability due to IPDLs lack of the finite model property. Also, it may be interesting to further extend the expressive power of ICPDL by adding fixedpoint operators. Decidability and complexity of the resulting logic are open.

REFERENCES

[1] Loredana Afanasiev, Patrick Blackburn, Ioanna Dimitriou, Evan Goris Bertrand Gaiffe, Maarten Marx, and Maarten de Rijke, *PDL for ordered trees*, ***Journal of Applied Non-Classical Logics***, vol. 15 (2005), no. 2, pp. 115–135.

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

[3] Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas W. Reps, and Mihalis Yannakakis, *Analysis of recursive state machines*, ***ACM Transactions on Programming Languages and Systems (TOPLAS)***, vol. 27 (2005), no. 4, pp. 786–818.

[4] Franz Baader, Deborah L. McGuiness, Daniele Nardi, and Peter Patel-Schneider, ***The description logic handbook: Theory, implementation and applications***, Cambridge University Press, 2003.

[5] Thierry Cachat, *Uniform Solution of Parity Games on Prefix-Recognizable Graphs*, ***Electronic Notes in Theoretical Computer Science***, vol. 68 (2002), no. 6.

[6] Didier Caucal, *On infinite transition graphs having a decidable monadic theory*, ***Theoretical Computer Science***, vol. 290 (2002), no. 1, pp. 79–115.

[7] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer, *Alternation*, ***Journal of the Association for Computing Machinery***, vol. 28 (1981), no. 1, pp. 114–133.

[8] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, ***Model checking***, MIT Press, 2000.

[9] Ryszard Danecki, *Nondeterministic propositional dynamic logic with intersection is decidable*, ***Proceedings of the 5th Symposium on Computation Theory (Zaborw, Poland)***, Lecture Notes in Computer Science, no. 208, 1984, pp. 34–53.

[10] ———, *Propositional Dynamic Logic with Strong Loop Predicate*, ***Proceedings of Mathematical Foundations of Computer Science 1984 (MFCS 1984)***, Lecture Notes in Computer Science, no. 176, 1984, pp. 573–581.

[11] J. Esparza, *On the Decidabilty of Model Checking for Several mu-calculi and Petri Nets*, ***Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP '94), Edinburgh (U.K.)*** (S. Tison, editor), Lecture Notes in Computer Science, no. 787, Springer, 1994, pp. 115–129.

[12] Javier Esparza, Antonín Kucera, and Stefan Schwoon, *Model checking LTL with regular valuations for pushdown systems*, ***Information and Computation***, vol. 186 (2003), no. 2, pp. 355–376.

[13] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, ***Reasoning About Knowledge***, MIT Press, 1995.

[14] L. Farinas Del Cerro and E. Orlowska, *DAL-a logic for data analysis*, ***Theoretical Computer Science***, vol. 36 (1985), no. 2-3, pp. 251–264.

[15] Michael J. Fischer and Richard E. Ladner, *Propositional Dynamic Logic of Regular Programs*, ***Journal of Computer and System Sciences***, vol. 18 (1979), no. 2, pp. 194–211.

[16] J. Flum, *On the (infinite) model theory of fixed point logics*, ***Models, Algebras, and Proofs: selected papers of the X Latin American symposium on mathematical logic held in Bogota*** (X. Caicedo and C. H. Montenegro, editors), Lecture Notes in Pure and Applied Mathematics, vol. 203, Marcel Dekker, Inc., 1999, pp. 67–75.

[17] Giuseppe De Giacomo and Maurizio Lenzerini, *Boosting the Correspondence between Description Logics and Propositional Dynamic logics*, ***Proceedings of the 12th National Conference on Artifical Intelligence (AAAI'94)***, 1994, pp. 205–212.

[18] Stefan Göller and Markus Lohrey, *Infinite State Model-Checking of Propositional Dynamic Logics*, ***Technical Report 2006/04***, Universität Stuttgart, FMI, February 2006.

[19] ———, *Infinite State Model-Checking of Propositional Dynamic Logics*, ***Proceedings of the 20th International Conference on Computer Science Logic (CSL 2006), Szeged (Hungary)***, Lecture Notes in Computer Science, no. 4207, Springer, 2006, pp. 349–364.

[20] Stefan Göller, Markus Lohrey, and Casten Lutz, *PDL with Intersection and Converse Is 2 EXP-Complete*, ***Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2007)***, Lecture Notes in Computer Science, no. 4423, Springer, 2007, pp. 198–212.

[21] E. Grädel, *Guarded fixed point logics and the monadic theory of countable trees*, ***Theoretical Computer Science***, vol. 288 (2002), no. 1, pp. 129–152.

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

[23] David Harel, Dexter Kozen, and Jerzy Tiuryn, ***Dynamic Logic***, Foundations of computing, The MIT Press, 2000.

[24] D. Kozen, *Results on the propositional μ-calculus*, ***Automata, Languages and Programming, 9th Colloquium*** (Mogens Nielsen and Erik Meineche Schmidt, editors), Lecture Notes in Computer Science, vol. 140, Springer-Verlag, 1982, pp. 348–359.

[25] O. Kupferman, N. Piterman, and M. Vardi, *Model Checking Linear Properties of Prefix-Recognizable Systems*, ***Proceedings of the 14. International Conference on Computer Aided Verification CAV 2002, Copenhagen (Denmark)*** (Ed Brinksma and Kim Guldstrand Larsen, editors), Lecture Notes in Computer Science, vol. 2404, 2002, pp. 371–385.

[26] Orna Kupferman and Moshe Y. Vardi, *An automata-theoretic approach to reasoning about infinite-state systems*, ***Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000), Chiacago (USA)*** (E. Allen Emerson and A. Prasad Sistla, editors), Lecture Notes in Computer Science, no. 1855, Springer, 2000, pp. 36–52.

[27] M. Lange, *Model Checking Propositional Dynamic Logic with All Extras*, ***Journal of Applied Logic***, vol. 4 (2005), no. 1, pp. 39–49.

[28] M. Lange and C. Lutz, *2-ExpTime Lower Bounds for Propositional Dynamic Logics with Intersection*, this Journal, vol. 70 (2005), no. 4, pp. 1072–1086.

[29] Denis Lugiez and Ph. Schnoebelen, *The regular viewpoint on PA-processes*, ***Theoretical Computer Science***, vol. 274 (2002), no. 1–2, pp. 89–115.

[30] Carsten Lutz, *PDL with Intersection and Converse Is Decidable*, ***Proceedings of the 19th International Workshop on Computer Science Logic (CSL 2005), Oxford (UK)*** (C.-H. Luke Ong, editor), Lecture Notes in Computer Science, no. 3634, Springer, 2005, pp. 413–427.

[31] Carsten Lutz and Dirk Walther, *PDL with Negation of Atomic Programs*, ***Journal of Applied Non-Classical Logics***, vol. 15 (2005), no. 2, pp. 189–213.

[32] Richard Mayr, *Decidability of model checking with the temporal logic EF*, ***Theoretical Computer Science***, vol. 256 (2001), no. 1-2, pp. 31–62.

[33] John-Jules Ch. Meyer, *Dynamic logic for reasoning about actions and agents*, ***Logic-based artificial intelligence*** (Jack Minker, editor), Kluwer Academic Publishers, 2000, pp. 281–311.

[34] David Muller and Paul Schupp, *Alternating automata on infinite trees*, ***Theoretical Computer Science***, vol. 54 (1987), no. 2-3, pp. 267–276.

[35] M.Y. Vardi and P. Wolper, *Automata theoretic techniques for modal logics of programs*, ***Journal of Computer and System Sciences***, vol. 32 (1986), no. 2, pp. 183–221.

[36] V.R. Pratt, *A near-optimal method for reasoning about action*, ***Journal of Computer and System Sciences***, vol. 20 (1980), pp. 231–254.

[37] H. Rogers, ***Theory of recursive functions and effective computability***, McGraw-Hill, 1968.

[38] B. ten Cate and C. Lutz, *The Complexity of Query Containment in Expressive Fragments of XPath 2.0*, ***Proceedings of the 26th ACM Symposium on Principles of Database Systems (PODS 2007)***, ACM Press, 2007.

[39] Balder ten Cate, *The expressivity of XPath with transitive closure*, ***Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)***, ACM Press, 2006, pp. 328–337.

[40] Wolfgang Thomas, *Some Perspectives of Infinite-State Verification*, ***Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA 2005), Taipei (Taiwan)*** (Doron Peled and Yih-Kuen Tsay, editors), Lecture Notes in Computer Science, no. 3707, Springer, 2005, pp. 3–10.

[41] W. van der Hoek and J. Meyer, *A complete epistemic logic for multiple agents - combining distributed and common knowledge*, 1997.

[42] Hans P. van Ditmarsch, Wiebe van der Hoek, and Barteld P. Kooi, *Concurrent dynamic epistemic logic for MAS*, ***Proceedings of the 2nd international joint conference on autonomous agents and multiagent systems (aamas 2003)***, ACM Press, 2003, pp. 201–208.

[43] Moshe Y. Vardi, *The taming of converse: Reasoning about two-way computations*, ***Proceedings of logics of programs***, Lecture Notes in Computer Science, no. 193, Springer, 1985, pp. 413–423.

[44] ———, *Reasoning about The Past with Two-Way Automata*, ***Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP '98), Aalborg (Denmark)*** (Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors), Lecture Notes in Computer Science, no. 1443, Springer, 1998, pp. 628–641.

[45] Igor Walukiewicz, *Model Checking CTL Properties of Pushdown Systems*, ***Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000), New Delhi (India)*** (Sanjiv Kapoor and Sanjiva Prasad, editors), Lecture Notes in Computer Science, no. 1974, Springer, 2000, pp. 127–138.

[46] ———, *Pushdown Processes: Games and Model-Checking*, ***Information and Computation***, vol. 164 (2001), no. 2, pp. 234–263.

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

UNIVERSITÄT LEIPZIG
INSTITUT FÜR INFORMATIK
ABTEILUNG ALGEBRAISCHE UND LOGISCHE GRUNDLAGEN DER INFORMATIK
JOHANNISGASSE 26
04103 LEIPZIG
GERMANY
E-mail: goeller@informatik.uni-leipzig.de

UNIVERSITÄT LEIPZIG
INSTITUT FÜR INFORMATIK
ABTEILUNG ALGEBRAISCHE UND LOGISCHE GRUNDLAGEN DER INFORMATIK
JOHANNISGASSE 26
04103 LEIPZIG
GERMANY
E-mail: lohrey@informatik.uni-stuttgart.de

DRESDEN UNIVERSITY OF TECHNOLOGY
INSTITUTE FOR THEORETICAL COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
NÖTHNITZER STR. 46
01062 DRESDEN
GERMANY
E-mail: lutz@tcs.inf.tu-dresden.de