PDL with Intersection and Converse is 2EXP-complete

Stefan Göller[1]*, Markus Lohrey[1], and Carsten Lutz[2]

[1] Universität Stuttgart, FMI, Germany
[2] Institute for Theoretical Computer Science, TU Dresden, Germany
goeller,lohrey@informatik.uni-stuttgart.de,
lutz@tcs.inf.tu-dresden.de

Abstract. We study the complexity of satisfiability in the expressive extension ICPDL of PDL (Propositional Dynamic Logic), which admits intersection and converse as program operations. Our main result is containment in 2EXP, which improves the previously known non-elementary upper bound and implies 2EXP-completeness due to an existing lower bound for PDL with intersection. The proof proceeds by showing that every satisfiable ICPDL formula has a model of tree-width at most two and then giving a reduction to the (non)-emptiness problem for alternating two-way automata on infinite trees. In this way, we also reprove in an elegant way Danecki's difficult result that satisfiability for PDL with intersection is in 2EXP.

1 Introduction

Propositional Dynamic Logic (PDL) was introduced by Fischer and Ladner in 1979 as a modal logic for reasoning about the input/output behaviour of programs [6]. In PDL, there are two syntactic entities: formulas, built from Boolean and modal operators and interpreted as sets of nodes 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. Since its invention, many different extensions of PDL have been proposed, mainly by allowing additional operators on programs. Three prominent such extensions are PDL with the converse operator (CPDL), PDL with the intersection operator (IPDL), and PDL with the negation operator on programs (NPDL), see the monograph [9] and references therein. While some of these extensions such as CPDL are well-suited for reasoning about programs, many of them aim at the numerous other applications that PDL has found since its invention. Notable examples of such applications include agent-based systems [14], regular path constraints [2], and XML-querying [1, 17]. In AI, PDL received attention due to its close relationship to description logics [7] and epistemic logic [18, 10].

The most important decision problem for PDL is satisfiability: is there a Kripke structure which satisfies a given formula at some node? A classical result of Fischer and Ladner states that satisfiability for PDL is EXP-complete [6, 16]. The EXP upper bound extends without difficulty to CPDL and can even be established for several extensions

* 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).

thereof [19]. In contrast, the precise complexity of satisfiability for IPDL was a long standing open problem. In [4], Danecki proved a 2EXP upper bound. Alas, Danecki's proof is rather difficult and many details are omitted in the published version. One of the reasons for the difficulty of IPDL is that, unlike PDL, it lacks the tree model property, i.e., a satisfiable IPDL formula does not necessarily have a tree model. Danecki proved that every satisfiable IPDL formula has a special model which can be encoded by a tree. This observation paves the way to using automata theoretic techniques in decision procedures for IPDL. Only recently, a matching 2EXP lower bound for IPDL was shown by Lange and the third author [11]. Regarding NPDL, it is long known that satisfiability is undecidable [9]. As recently shown in [9], the fragment of NPDL in which program negation is restricted to atomic programs is decidable and EXP-complete.

In this paper, we consider extensions of PDL with (at least two of) converse, intersection, and negation. Our main result concerns the complexity of satisfiability in ICPDL, the extension of PDL with both converse and intersection. Decidability was shown by the third author in [12] using a reduction to monadic second order logic over the infinite binary tree. However, this only yields a nonelementary algorithm which does not match the 2EXP lower bound that ICPDL inherits from IPDL. We prove that satisfiability in ICPDL can be decided in 2EXP, and thus settle the complexity of ICPDL as 2EXP-complete. There are some additional virtues of our result. First, we provide a shorter and (hopefully) more comprehensible proof of the 2EXP upper bound for IPDL. Second, the information logic DAL (data analysis logic) [5] is a fragment of ICPDL (but not of IPDL) and thus inherits the 2EXP upper bound. And third, our result has applications in description logic and epistemic logic, see [12] for more details.

Our main result is proved in three clearly separated parts. In part one, we establish a model property for ICPDL based on the notion of tree width. Tree width measures how close a graph is to a tree, and is one of the most important concepts in modern graph theory with many applications in computer science. As mentioned earlier, IPDL (and hence also ICPDL) does not have the tree model property. We prove that ICPDL enjoys an "almost tree model property": every satisfiable ICPDL formula has a model of tree width at most two This part of our proof is comparable to Danecki's observation that every satisfiable IPDL formula has a special model which can be encoded by a tree.

In part two of our proof, we use the established model property to give a polytime reduction of satisfiability in ICPDL to what we call *ω-regular tree satisfiability* in ICPDL. The latter problem is defined in terms of two-way alternating parity tree automata (TWAPTAs). A TWAPTA is an alternating automaton with a parity acceptance condition that runs on infinite node-labeled trees and can move upwards and downwards in the tree. Infinite node-labeled trees 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 in ICPDL to this problem is based on a suitable encoding of width two tree decompositions of Kripke structures. The TWAPTA constructed in the reduction accepts precisely such encodings.

Finally, in part three we reduce ω-regular tree satisfiability in ICPDL to the nonemptiness problem for TWAPTAs. The latter problem was shown to be EXP-complete in [20]. Since our reduction of ω-regular tree satisfiability in ICPDL to TWAPTA-non-emptiness involves an exponential blow-up in automata size, we obtain an 2EXP upper bound for ω-regular tree satisfiability in ICPDL and also for standard satisfiability in ICPDL. The reduction employs a technique from [8], where the first and second author proved that the model-checking problem for IPDL over transition graphs of pushdown automata is 2EXP-complete. In fact, this model-checking problem can be easily reduced to ω-regular tree satisfiability in ICPDL. This illustrates that ω-regular tree satisfiability in ICPDL is of interest beyond its application in the current paper.

To obtain a more complete picture, we finally investigate the option of extending ICPDL with program negation. It turns out that in the presence of intersection, program negation is problematic from a computational perspective. In particular, we prove that already IPDL extended with negation restricted to atomic programs is undecidable. This should be contrasted with the decidability result for PDL extended with atomic negation mentioned above [13].

2 ICPDL

Let $\mathbb{P}$ be a set of *atomic propositions* and $\mathbb{A}$ a set of *atomic programs*. *Formulas* φ and *programs* π of the logic ICPDL are defined by the following grammar, where p ranges over $\mathbb{P}$ and a over $\mathbb{A}$:

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

We introduce the usual abbreviations $\varphi_1 \wedge \varphi_2 = \langle\varphi_1?\rangle\varphi_2$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$. The fragment IPDL of ICPDL is obtained by dropping the $\overline{\pi}$ clause from the above grammar.

The *semantics* of ICPDL is defined in terms of Kripke structures. A *Kripke structure* is a tuple $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$, where (i) X is a set of *states*, (ii) $\rightarrow_a \subseteq X \times X$ is a *transition relation* for each $a \in \mathbb{A}$, and (iii) $\rho : X \rightarrow 2^{\mathbb{P}}$ assigns to each state a set of atomic propositions. Given a Kripke structure $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$, we define by mutual induction for each ICPDL program π a binary relation $[\![\pi]\!]_K \subseteq X \times X$ and for each ICPDL formula φ a subset $[\![\varphi]\!]_K \subseteq X$ as follows: [3]

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

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

For $x \in X$ we write $(K, x) \models \varphi$ if $x \in [\![\varphi]\!]_K$. If $(K, x) \models \varphi$ for some $x \in X$, then K is a *model* of φ. The formula φ is *satisfiable* if there exists a model of φ.

Since the converse operator can be pushed down to atomic programs, we assume for the rest of this paper that converse is only applied to atomic programs. Let us set $\overline{\mathbb{A}} = \{\overline{a} \mid a \in \mathbb{A}\}$. The size $|\varphi|$ of an ICPDL formula φ and the size $|\pi|$ of an ICPDL program π is defined as follows: $|p| = |a| = 1$ for all $p \in \mathbb{P}$ and $a \in \mathbb{A} \cup \overline{\mathbb{A}}$, $|\neg\varphi| = |\varphi?| = |\varphi| + 1$, $|\langle\pi\rangle\varphi| = |\pi| + |\varphi| + 1$, $|\pi_1 \text{ op } \pi_2| = |\pi_1| + |\pi_2| + 1$ for op $\in \{\cup, \cap, \circ\}$, and $|\pi^*| = |\pi| + 1$.

The main result of this paper is the following.

Theorem 1. *Satisfiability in ICPDL is* 2EXP*-complete.*

As discussed in the introduction, it suffices to give a 2EXP algorithm for satisfiability in ICPDL because of the known 2EXP lower bound for IPDL [11]. The rest of the paper is organized as follows. In Section 3, we show that every satisfiable ICPDL formula has a model of tree width at most two. In Section 4, satisfiability of ICPDL formulas in a model of tree width at most two is reduced to ω-regular tree satisfiability in ICPDL. In Section 5, the latter problem is shown to be in 2EXP. Finally, Section 6 contains the undecidability proof for IPDL extended with negation of atomic programs.

3 Models of Tree-Width Two Suffice

We start with defining tree decompositions and the tree-width of Kripke structures. Although we do not assume countability of Kripke structures in general, it suffices to consider tree decompositions and the tree width only of countable Kripke structures. Let $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$ be a countable Kripke structure. A *tree decomposition* of K is a tuple $(T, (X_v)_{v \in V})$, where $T = (V, E)$ is a countable undirected tree, X_v is a subset of X (also called a *bag*) for all $v \in V$, and the following conditions are satisfied:

– $\bigcup_{v \in V} X_v = X$
– For every transition $x \rightarrow_a y$ of K there exists $v \in V$ with $x, y \in X_v$.
– For every $x \in X$, the set $\{v \in V \mid x \in X_v\}$ is a connected subset of the tree T.

The width of this tree decomposition is the supremum of $\{|X_v| - 1 \mid v \in V\}$. The *tree width* of a Kripke structure K is the minimal k such that K has a tree decomposition of width k. The purpose of this section is to prove the following theorem.

Theorem 2. *Every satisfiable ICPDL formula has a countable model of tree width at most two.*

As a preliminary to proving Theorem 2, we mutually define the set of *subprograms* $\mathsf{subp}(\alpha)$ and the set of *subformulas* $\mathsf{subf}(\alpha)$, where α is either an ICPDL formula or an ICPDL program:

– $\mathsf{subp}(a) = \{a\}$, $\mathsf{subp}(\overline{a}) = \{a, \overline{a}\}$, $\mathsf{subf}(a) = \mathsf{subf}(\overline{a}) = \emptyset$ for $a \in \mathbb{A}$;
– $\mathsf{subp}(\pi) = \{\pi\} \cup \mathsf{subp}(\pi_1) \cup \mathsf{subp}(\pi_2)$ and $\mathsf{subf}(\pi) = \mathsf{subf}(\pi_1) \cup \mathsf{subf}(\pi_2)$ if $\pi = \pi_1 \text{ op } \pi_2$ for op $\in \{\cup, \cap, \circ\}$;
– $\mathsf{subp}(\pi^*) = \{\pi^*\} \cup \mathsf{subp}(\pi)$ and $\mathsf{subf}(\pi^*) = \mathsf{subf}(\pi)$;

- $\mathsf{subp}(\varphi?) = \{\varphi?\} \cup \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\varphi?) = \mathsf{subf}(\varphi)$
- $\mathsf{subp}(p) = \emptyset$ and $\mathsf{subf}(p) = \{p\}$ for $p \in \mathbb{P}$;
- $\mathsf{subp}(\neg\varphi) = \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\neg\varphi) = \{\neg\varphi\} \cup \mathsf{subf}(\varphi)$;
- $\mathsf{subp}(\langle\pi\rangle\varphi) = \mathsf{subp}(\pi) \cup \mathsf{subp}(\varphi)$ and $\mathsf{subf}(\langle\pi\rangle\varphi) = \{\langle\pi\rangle\varphi\} \cup \mathsf{subf}(\pi) \cup \mathsf{subf}(\varphi)$.

To prove Theorem 2, fix a satisfiable formula φ_0, a (not necessarily countable) model $K = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$ of φ_0, and a state $x_0 \in [\![\varphi_0]\!]_K$. Also fix choice functions W (for witness), U (for union), C (for composition), and S (for star) such that

- if $\varphi = \langle\pi\rangle\psi \in \mathsf{subf}(\varphi_0)$ and $x \in [\![\varphi]\!]_K$, then $W(x, \varphi) = y \in X$ such that $y \in [\![\psi]\!]_K$ and $(x, y) \in [\![\pi]\!]_K$;
- if $\pi = \chi \cup \sigma \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$, then $U(x, \pi, y) = \tau \in \{\chi, \sigma\}$ such that $(x, y) \in [\![\tau]\!]_K$.
- if $\pi = \chi \circ \sigma \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$, then $C(x, \pi, y) = z \in X$ such that $(x, z) \in [\![\chi]\!]_K$ and $(z, y) \in [\![\sigma]\!]_K$;
- if $\pi = \chi^* \in \mathsf{subp}(\varphi_0)$ and $(x, y) \in [\![\pi]\!]_K$ with $x \neq y$, then $S(x, \pi, y) = z \in X$ such that there exists a sequence $x_0, \ldots, x_n \in X$ with
  1. $x_0 = x$ and $x_n = y$;
  2. $(x_i, x_{i+1}) \in [\![\chi]\!]_K$ for all $i < n$;
  3. $x_0, \ldots, x_n$ is a shortest sequence with Properties 1 and 2;
  4. $x_1 = z$.

Now we inductively define a node-labeled tree $(T, (t_v)_{v \in V})$ with $T = (V, E)$ and $t_v \in X \cup X^2 \cup X^3$ for all $v \in V$. During the construction, each node in the tree is assigned a type, which may either be "singleton" or π for $\pi \in \mathsf{subp}(\varphi_0)$. Figure 1 illustrates the different cases, which are as follows:

1. Start the construction with a root node v of type singleton and set $t_v = x_0$;
2. if $v \in V$ is of type singleton and $t_v = x$, then for every $\varphi = \langle\pi\rangle\psi \in \mathsf{subf}(\varphi_0)$ such that $x \in [\![\varphi]\!]_K$, add a successor w of type π and set $t_w = (x, W(x, \varphi))$;
3. if $v \in V$ is of type a or $\overline{a}$, where $a \in \mathbb{A}$ and $t_v = (x, y)$, then add a successor w of type singleton and set $t_w = y$;
4. if $v \in V$ is of type $\pi = \chi \cup \sigma$ and $t_v = (x, y)$, then
   - add a successor w of type singleton and set $t_w = y$;
   - add a successor w' of type $U(x, \pi, y)$ and set $t_{w'} = (x, y)$;
5. if $v \in V$ is of type $\pi = \chi \cap \sigma$ and $t_v = (x, y)$, then
   - add a successor w of type singleton and set $t_w = y$;
   - add successors u, u' of type χ and σ, respectively, and set $t_u = t_{u'} = (x, y)$;
6. if $v \in V$ is of type $\pi = \chi \circ \sigma$ and $t_v = (x, y)$, then
   - add a successor w of type singleton and set $t_w = y$;
   - add a successor w' of type π and set $t_w = (x, C(x, \pi, y), y)$;
7. if $v \in V$ is of type $\pi = \chi \circ \sigma$ and $t_v = (x, z, y)$, then add successors u, u' of type χ and σ and set $t_u = (x, z)$ and $t_{u'} = (z, y)$;
8. if $v \in V$ is of type $\pi = \chi^*$ and $t_v = (x, y)$ with $x \neq y$, then
   - add a successor w of type singleton and set $t_w = y$;
   - add a successor w' of type π and set $t_w = (x, S(x, \pi, y), y)$;
9. if $v \in V$ is of type $\pi = \chi^*$ and $t_v = (x, z, y)$, then add successors u, u' of type χ and π, respectively, and set $t_u = (x, z)$ and $t_{u'} = (z, y)$.

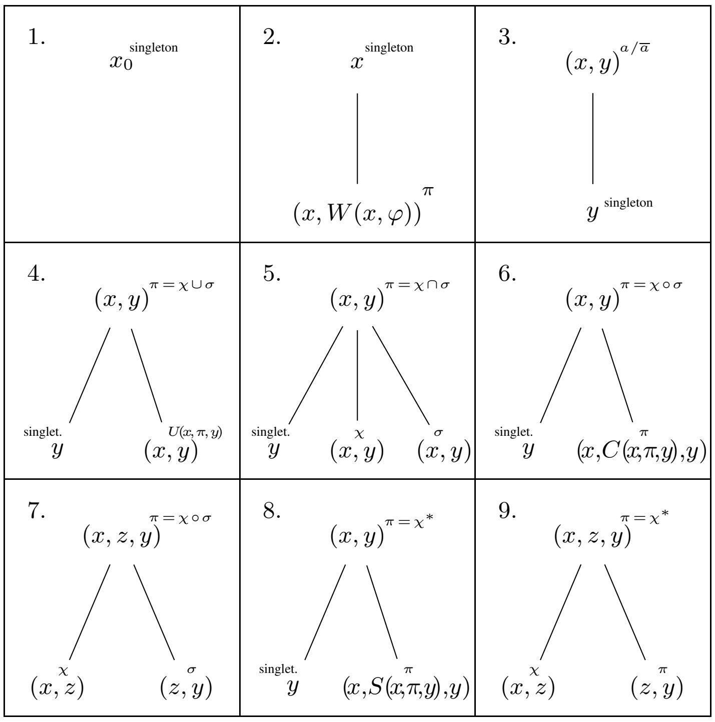

Fig. 1. Inductive definition of $(T, (t_v)_{v\in V})$.

We assume that successors are added at most once to each node in the induction step and that the construction proceeds in a breadth first manner. Note that nodes of type $\psi?$ are always leafs, and so are nodes v of type χ^* with $t_v = (x,x)$ for some $x \in X$. Another important property, which illustrates the connection between K and the constructed tree, is the following:

$$\forall v \in V : \text{if } v \text{ is of type } \pi \text{ and } t_v = (x,y), \text{ then } (x,y) \in [\![\pi]\!]_K. \qquad (\dagger)$$

A *place* is a pair (v,x) such that x is a member of t_v. We denote the set of all places with P and let $\sim$ be the smallest equivalence relation on P which contains all pairs of the form $((u,x),(v,x))$, where $(u,v) \in E$ is an edge of the tree T. We use $[v,x]$ to denote the equivalence class of $(v,x) \in P$ w.r.t. the relation $\sim$. Define a Kripke structure $K' = (X', \{\to'_a \mid a \in \mathbb{A}\}, \rho')$ as follows:

– $X' = \{[v,x] \mid (v,x) \in P\}$;
– $[v,x] \to'_a [v',y]$ if and only if at least one of the following holds:
  • there is $u \in V$ of type a s.t. $t_u = (x,y)$, $(u,x) \sim (v,x)$, and $(u,y) \sim (v',y)$;
  • there is $u \in V$ of type $\overline{a}$ s.t. $t_u = (y,x)$, $(u,x) \sim (v,x)$, and $(u,y) \sim (v',y)$.
– $\rho'([v,x]) = \rho(x)$.

Since K' is clearly countable, to finish the proof it suffices to show the following:

1. setting $X_v = \{[v,x] \mid x \text{ occurs in } t_v\}$ for all $v \in V$, we obtain a tree decomposition $(T, (X_v)_{v \in V})$ of K' of width two;
2. K' satisfies φ_0.

Using the definitions of K' and $\sim$, it is readily checked that $(T, (X_v)_{v\in V})$ is a tree decomposition of K'. Tree width two is then immediate by construction of $(T, (t_v)_{v\in V})$. Finally, we can prove the following, whose Point 3 yields that K' is a model of φ_0.

Lemma 1. *For all $v, u \in V$, $x, y \in X$, $\pi \in \mathsf{subp}(\varphi_0)$, and $\varphi \in \mathsf{subf}(\varphi_0)$,*

1. *if $t_v = (x,y)$ and v is of type π, then $([v,x],[v,y]) \in [\![\pi]\!]_{K'}$;*
2. *if $(v,x),(u,y) \in P$ and $([v,x],[u,y]) \in [\![\pi]\!]_{K'}$, then $(x,y) \in [\![\pi]\!]_K$;*
3. *if $(v,x) \in P$, then $(K,x) \models \varphi$ if and only if $(K',[v,x]) \models \varphi$.*

4 Reduction to ω-Regular Tree Satisfiability

We exploit the model property established in the previous section to reduce satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL. Since the latter is defined in terms of alternating automata on infinite trees, we start with introducing these automata and the trees on which they work.

Let Γ and Υ be finite sets. A Γ-labeled (directed) Υ-tree is a partial function $T : \Upsilon^* \to \Gamma$ such that $\mathrm{dom}(T)$ (the set of nodes) is prefix-closed. If $\mathrm{dom}(T) = \Upsilon^*$, then T is called *complete*. If Υ is understood or not important, we simply talk of Γ-labeled trees. We deliberately work with two kinds of trees here: undirected trees as a basis for tree decompositions in Section 3, and directed trees introduced here as the objects on which alternating tree automata work.

Let P be a finite set of atomic propositions and A a finite set of atomic programs, not necessarily identical to the sets $\mathbb{P}$ and $\mathbb{A}$ fixed in Section 2. A 2^{P}-labeled A-tree T can be viewed as a Kripke structure $K_T = (\mathsf{A}^*, \{\to_a \mid a \in \mathsf{A}\}, T)$ over the set of atomic propositions P and atomic programs A, where $\to_a = \{(u, ua) \mid ua \in T\}$ for all $a \in \mathsf{A}$. In the following, we identify T and the associated Kripke structure K_T.

We now define alternating automata on complete Γ-labeled Υ-trees. For a finite set X we denote by $\mathcal{B}^+(X)$ the set of all *positive boolean formulas* with elements of X used as variables. The constants `true` and `false` are admitted. A subset $Y \subseteq X$ can be seen as a valuation in the obvious way: it *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if and only if by assigning `true` to all elements in Y the formula θ is evaluated to `true`. Define the set of Υ-*moves* as $\mathrm{mov}(\Upsilon) = \Upsilon \uplus \overline{\Upsilon} \uplus \{\varepsilon\}$, where $\overline{\Upsilon} = \{\overline{a} \mid a \in \Upsilon\}$. For $u \in \Upsilon^*$ and $a \in \Upsilon$, define $u\overline{a} = v$ if $u = va$ for some $v \in \Upsilon^*$ and $u\overline{a} =$ undefined if $u \notin \Upsilon^* a$. A *two-way alternating parity tree automaton* (TWAPTA for short) over Γ-labeled Υ-trees is a tuple $\mathcal{T} = (S, \delta, s_0, \mathrm{Acc})$, where (i) S is a finite non-empty set of states, (ii) $\delta : S \times \Gamma \to \mathcal{B}^+(S \times \mathrm{mov}(\Upsilon))$ is the *transition function*, (iii) $s_0 \in S$ is the *initial state*, and (iv) $\mathrm{Acc} : S \to \{0, \ldots, m\}$ is the *priority function* (where $m \in \mathbb{N}$) which assigns to each state an integer between 0 and m. Define $|\mathrm{Acc}| = \max\{\mathrm{Acc}(s) \mid s \in S\}$. Let T be a complete Γ-labeled Υ-tree, $u \in \Upsilon^*$ a node, and $s \in S$ a state. An $(s,u)_T$-*run* of $\mathcal{T}$ is a (not necessarily complete) $(S \times \Upsilon^*)$-labeled Ω-tree T_R for some finite set Ω such that the following two conditions are satisfied: (i) $T_R(\varepsilon) = (s,u)$, and (ii) if $\alpha \in \mathrm{dom}(T_R)$

with $T_R(\alpha) = (q, v)$ and $\delta(q, T(v)) = \theta$, then there exists a subset $Y \subseteq S \times \mathrm{mov}(\Upsilon)$ that satisfies the formula θ and for all $(s', e) \in Y$, ve is defined and there exists a $\sigma \in \Omega$ with $\alpha\sigma \in \mathrm{dom}(T_R)$ and $T_R(\alpha\sigma) = (s', ve)$. We say that an $(s, u)_T$-run is *successful*, if for every infinite path $\alpha_1\alpha_2 \cdots \in \mathrm{dom}(T_R)^\omega$ of T_R ($\alpha_1 = \varepsilon$, $\alpha_{i+1} = \alpha_i\sigma$ for some $\sigma \in \Omega$), the number $\min\{\mathrm{Acc}(q) \mid q \in S, T_R(\alpha_i) \in \{q\} \times \Upsilon^* \text{ for infinitely many } i\}$ is even. Define

$$\begin{aligned} [\![\mathcal{T}, s]\!]_T &= \{u \in \Upsilon^* \mid \text{ there exists a successful } (s, u)_T\text{-run of } \mathcal{T}\} \\ L(\mathcal{T}) &= \{T \mid \varepsilon \in [\![\mathcal{T}, s_0]\!]_T\} \end{aligned}$$

The subscript T is omitted if clear from the context. An *ω-regular tree language L* is a set of complete Γ-labeled Υ-trees such that $L(\mathcal{T}) = L$ for some TWAPTA $\mathcal{T}$.

Our TWAPTA model differs slightly from other definitions in the literature: First, we run TWAPTA only on complete trees; this will be convenient in Section 5. Second, usually a TWAPTA has an operation $\uparrow$ for moving to the parent node of the current node. In our model, $\uparrow$ is replaced by the operations $\overline{a} \in \overline{\Upsilon}$ for all $a \in \Upsilon$. The operation $\overline{a}$ can only be executed if the current node is an a-successor of its parent node. It is easy to see that these two models are equivalent.

In Section 5, we will make use of the following result of Vardi:

Theorem 3 ([20]). *For a given TWAPTA $\mathcal{T} = (Q, \delta, s_0\mathrm{Acc})$ it can be checked in time exponential in $|Q| \cdot |\mathrm{Acc}|$ whether $L(\mathcal{T}) = \emptyset$.*

We are now in the position to formally define *ω-regular tree satisfiability in ICPDL*: given a TWAPTA $\mathcal{T}$ over 2^{P}-labeled A-trees and an ICPDL formula φ using only atomic propositions from P and atomic programs from A (in the following we simply say that φ is *over* P *and* A), decide whether there is a $T \in L(\mathcal{T})$ such that $(T, \varepsilon) \models \varphi$.

To reduce satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL, we translate an ICPDL formula φ over $\mathbb{P}$ and $\mathbb{A}$ into a TWAPTA $\mathcal{T}$ and an ICPDL formula $\widehat{\varphi}$ over

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

where $\mathsf{prop}(\varphi) = \mathsf{subf}(\varphi) \cap \mathbb{P}$ and $\mathsf{prog}(\varphi) = \mathsf{subp}(\varphi) \cap \mathbb{A}$. Intuitively, each 2^{P}-labeled A-tree T accepted by $\mathcal{T}$ encodes a tree decomposition of a Kripke structure K over $\mathbb{P}$ and $\mathbb{A}$ of tree width at most two (in a sense yet to be made precise), and T is a model of $\widehat{\varphi}$ if and only if K is a model of φ. To achieve an elegant encoding of tree decompositions, we work with *good* tree decompositions. A tree decomposition $(T, (X_v)_{v \in V})$ with $T = (V, E)$ is called good if

- $V = \{a, b\}^*$, i.e., T is a complete binary tree, and
- $X_v \subseteq X_{vc}$ or $X_{vc} \subseteq X_v$ for all $v \in V$ and $c \in \{a, b\}$.

It is easily seen how to convert a tree decomposition of a Kripke structure K of width k into a good tree decomposition of K of width k by introducing additional nodes.

Lemma 2. *Every countable Kripke structure of tree width k has a good tree decomposition of width k.*

In the following, we only need the case where $k = 2$. To encode a good tree decomposition $(T, (X_v)_{v \in V})$ of width two of a Kripke structure as a 2^{P}-labeled A-tree, we think of every tree node $v \in \{a, b\}^*$ as being divided into three slots which can be empty or filled with a state of the Kripke structure. When moving to a child, by the second condition of good tree decompositions we either add nodes to empty slots or remove nodes from slots, but not both. The three slots of the node v are described by new leafs $v0, v1, v2$. This explains our choice of A above. When slot vi is occupied by a state of the Kripke structure, then vi receives the special label $t \in \mathsf{P}$ (and probably propositional letters as additional labels). Information about the edges of the Kripke structure are stored in tree nodes from $\{a, b\}^*$. We now formally define these encodings. We work with complete trees because TWAPTAs work on such trees. Nodes that are present only to ensure completeness of the tree are labelled with the empty set. 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 either $T(vi) = \emptyset$ or $\{t\} \subseteq T(vi) \subseteq \{t\} \cup \mathbb{P}$; set $X_v := \{i \mid t \in T(vi)\}$;
– if $v \in \{a, b\}^*$, then $T(v) \subseteq X_v \times \mathbb{A} \times X_v$;
– if $v \in \{a, b\}^*$ and $c \in \{a, b\}$, then $X_v \subseteq X_{vc}$ or $X_{vc} \subseteq X_v$;
– 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 $\mathbb{P}$ and $\mathbb{A}$ whose good tree decomposition is described by T. Define a set of *places* $P = \{u \in \mathsf{A}^* \mid t \in T(u)\}$ and let $\sim$ be the smallest equivalence relation on P which contains all pairs $(vi, vci) \in P \times P$, where $v \in \{a, b\}^*$, $c \in \{a, b\}$, and $0 \leq i \leq 2$. For $u \in P$, we use $[u]$ to denote the equivalence class of u w.r.t. $\sim$. Now set $K(T) = (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \rho)$, where:

$$
\begin{aligned}
X &= \{[u] \mid u \in P\} \\
\rightarrow_a &= \{([vi], [vj]) \mid v \in \{a, b\}^*, (i, a, j) \in T(v)\} \\
\rho([u]) &= \bigcup_{v \in [u]} T(v) \cap \mathbb{P}
\end{aligned}
$$

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 this section: the original formula φ whose satisfiability is to be decided is interpreted in $K(T)$ whereas the reduction formula $\widehat{\varphi}$ is interpreted in T viewed as a Kripke structure. The following two lemmas are easily proved.

Lemma 3. *If T is a valid 2^{P}-labeled A-tree, then the Kripke structure $K(T)$ has tree width at most two. Conversely, if K is of tree width at most two, then there exists a valid 2^{P}-labeled A-tree T such that K is isomorphic to $K(T)$.*

Lemma 4. *The set of all valid 2^{P}-labeled A-trees is an ω-regular tree language.*

Now we show how to convert formulas ψ and programs π over $\mathsf{prop}(\varphi)$ and $\mathsf{prog}(\varphi)$ into formulas $\widehat{\psi}$ and programs $\widehat{\pi}$ over P and A such that for every valid 2^{P}-labeled A-tree T, we have (i) $[\![\widehat{\pi}]\!]_T \subseteq P \times P$ and (ii) for all $u, v \in P$,

$$
\begin{aligned}
u \in [\![\widehat{\psi}]\!]_T &\Leftrightarrow [u] \in [\![\psi]\!]_{K(T)} \\
(u, v) \in [\![\widehat{\pi}]\!]_T &\Leftrightarrow ([u], [v]) \in [\![\pi]\!]_{K(T)}
\end{aligned}
$$

First, we define the auxiliary program

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

and let $\pi_\sim = (\pi^1_\sim)^*$. Note that $[\![\pi_\sim]\!]_T$ equals $\sim$. Now, for all $a \in \mathsf{prog}(\varphi)$ and $p \in \mathsf{prop}(\varphi)$ we define

$$\widehat{a} = \bigcup_{i,j \in \{0,1,2\}} \pi_\sim \circ \bar{i} \circ (i,a,j)? \circ j \circ \pi_\sim \quad \text{and} \quad \widehat{p} = \langle \pi_\sim \rangle p.$$

To extend this translation to complex ICPDL formulas and programs, we can simply replace all atomic programs a and formulas p with $\widehat{a}$ and $\widehat{p}$, respectively. From the construction of $\widehat{\varphi}$ and Lemmas 2 and 3, we obtain the following.

Proposition 1. *The formula φ has a model of tree width at most two if and only if there is a valid 2^{P}-labeled A-tree T such that $(T,\varepsilon) \models \langle (0 \cup 1 \cup 2) \circ t? \rangle \widehat{\varphi}$.*

From Theorem 2, Lemma 4, and Proposition 1, we obtain:

Theorem 4. *There is a polynomial time reduction from satisfiability in ICPDL to ω-regular tree satisfiability in ICPDL.*

5 ω-Regular Tree Satisfiability in ICPDL is in 2EXP

Our remaining goal is to show that ω-regular tree satisfiability in ICPDL can be solved in doubly exponential time. This is achieved by a reduction to the EXP-complete (non)-emptiness problem for TWAPTAs. The main ingredient of the reduction is an inductive translation of ICPDL formulas into TWAPTAs and ICPDL programs into a certain kind of non-deterministic automata which we call NFAs. NFAs resemble word automata, but navigate in a complete A-tree reading symbols from $\mathsf{A} \cup \overline{\mathsf{A}}$. They can make conditional ε-transitions, which are executable only if the current tree node is accepted by some fixed TWAPTA. We start with presenting NFAs and the inductive translation.

Fix a finite set of atomic propositions P and a finite set of atomic programs A. For the rest of this section, it is more convenient to assume that a TWAPTA does not have an initial state. Hence, it is just a tuple of the form $(S, \delta, \mathrm{Acc})$. A *non-deterministic finite automaton (NFA) A* over a TWAPTA $\mathcal{T} = (S, \delta, \mathrm{Acc})$ is a pair $(Q, \rightarrow_A)$, where Q is a finite set of *states* and $\rightarrow_A$ is a set of transitions of the following form, where $q, q' \in Q$:

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

Transitions of the latter kind are called *test transitions*. Let T be a complete 2^{P}-labeled A-tree. Define the relation $\Rightarrow_{A,T} \subseteq (\mathsf{A}^* \times Q) \times (\mathsf{A}^* \times Q)$ as the smallest relation such that

- $(u,p) \Rightarrow_{A,T} (ua,q)$ if $p \xrightarrow{a}_A q$ $(a \in \mathsf{A}, u \in \mathsf{A}^*)$;
- $(ua,p) \Rightarrow_{A,T} (u,q)$ if $p \xrightarrow{\bar{a}}_{A,T} q$ $(\bar{a} \in \overline{\mathsf{A}}, u \in \mathsf{A}^*)$;
- $(u,p) \Rightarrow_{A,T} (u,q)$ if $p \xrightarrow{\mathcal{T},s}_A q$ and $u \in [\![\mathcal{T},s]\!]_T$ $(u \in \mathsf{A}^*)$.

For a pair $(p,q) \in Q \times Q$, define $[\![A,p,q]\!]_T = \{(u,v) \in \mathsf{A}^* \times \mathsf{A}^* \mid (u,p) \Rightarrow^*_{A,T} (v,q)\}$.

5.1 From ICPDL to Automata

For each ICPDL formula φ, we construct a TWAPTA $\mathcal{T}(\varphi)$ such that for all 2^{P}-labeled A-trees T, $[\![\mathcal{T}(\varphi), s]\!]_T = [\![\varphi]\!]_T$, where s is some selected state of $\mathcal{T}(\varphi)$. For each ICPDL program π, we construct a TWAPTA $T(\pi)$ and an NFA $A(\pi)$ over $\mathcal{T}(\pi)$ such that for all 2^{P}-labeled A-trees T, $[\![A(\pi), p, q]\!]_T = [\![\pi]\!]_T$, where p, q are two selected states of $A(\pi)$. In the following, the index T will be omitted for brevity. The construction is by induction on the structure of φ and π. We start with the construction of the TWAPTAs $T(\varphi)$ for ICPDL formulas φ.

If $\psi = p \in \mathsf{P}$, we put $\mathcal{T}(\psi) = (\{s\}, \delta, s \mapsto 1)$, where for all $Y \subseteq \mathsf{P}$ we have $\delta(s, Y) = \mathtt{true}$ if $p \in Y$ and $\delta(s, Y) = \mathtt{false}$ otherwise.

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. [15].

If $\psi = \langle\pi\rangle\theta$, then we have inductively constructed $A = A(\pi)$ with state set Q over a TWAPTA $\mathcal{T}(\pi) = (S_1, \delta_1, \mathrm{Acc}_1)$ such that $[\![\pi]\!] = [\![A, p_0, q_0]\!]$ for some $p_0, q_0 \in Q$. We have also constructed a TWAPTA $\mathcal{T}(\theta) = (S_2, \delta_2, \mathrm{Acc}_2)$ such that $[\![\theta]\!] = [\![\mathcal{T}(\theta), s_0]\!]$ for some $s_0 \in S_2$. We construct the TWAPTA $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$ with $S = Q \uplus S_1 \uplus S_2$. For states in S_1 or in S_2, the transitions of $\mathcal{T}(\psi)$ are as in $\mathcal{T}(\pi)$ and $\mathcal{T}(\theta)$, respectively. It remains to simulate A. Handling transitions of A of the form $q \xrightarrow{a}_A q'$ is easy: $\mathcal{T}(\psi)$ simply navigates up or down in the tree as required. When $\mathcal{T}(\psi)$ is in state $q \in Q$ and there is a transition $q \xrightarrow{\mathcal{T}(\pi), s}_A r$, we branch universally to simulate both $\mathcal{T}(\pi)$ in state s *and* the state change of A to state r. Finally, we admit an ε-transition from state q_0 to s_0, thus simulating $\mathcal{T}(\theta)$ after finishing the simulation of A. Formally, for $q \in Q$ and $Y \subseteq \mathsf{P}$, we define

$$\begin{aligned}\delta(q, Y) = &\bigvee\{\langle r, a\rangle \mid r \in Q, a \in \mathsf{A} \cup \overline{\mathsf{A}}, q \xrightarrow{a}_A r\} \ \vee \\ &\bigvee\{\langle s, \varepsilon\rangle \wedge \langle r, \varepsilon\rangle \mid r \in Q, s \in S_1, q \xrightarrow{\mathcal{T}(\pi), s}_A r\} \ \vee \\ &((q = q_0) \wedge \langle s_0, \varepsilon\rangle)\end{aligned}$$

The priority function Acc is defined by setting $\mathrm{Acc}(s) = 1$ if $s \in Q$ and $\mathrm{Acc}(s) = (\mathrm{Acc}_1 \uplus \mathrm{Acc}_2)(s)$ for $s \in S_1 \uplus S_2$. We set $\mathrm{Acc}(s) = 1$ for all $s \in Q$ since we want to assure that the NFA A is simulated for finitely many steps only, as $\psi = \langle\pi\rangle\theta$ is a diamond formula. We obtain $[\![\psi]\!] = [\![\mathcal{T}(\psi), p_0]\!]$.

We now describe the inductive construction of $A(\pi)$ and $\mathcal{T}(\pi)$ for an ICPDL program π. If $\mathcal{T}_i = (S_i, \delta_i, \mathrm{Acc}_i)$, $i \in \{1, 2\}$, are two TWAPTAs with disjoint sets of states, in what follows we use $\mathcal{T}_1 \uplus \mathcal{T}_2 = (S_1 \uplus S_2, \delta_1 \uplus \delta_2, \mathrm{Acc}_1 \uplus \mathrm{Acc}_2)$ denote their disjoint union; it is defined in the obvious way.

If $\pi = a \in \mathsf{A} \cup \overline{\mathsf{A}}$, the NFA $A(\pi)$ has two states p and q with the only transition $p \xrightarrow{a} q$. Hence, $[\![\pi]\!] = [\![A(\pi), p, q]\!]$.

If $\pi = \psi?$, we can assume that there exists a TWAPTA $\mathcal{T}(\psi)$ with a state s such that $[\![\psi]\!] = [\![\mathcal{T}(\psi), s]\!]$. The TWAPTA $\mathcal{T}(\pi)$ is $\mathcal{T}(\psi)$. The NFA $A(\pi)$ has two states p and

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

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

It remains to construct $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$, which is the most difficult step of the construction. Assume that the NFAs $A(\pi_i) = (Q_i, \rightarrow_{A(\pi_i)})$ over the TWAPTAs $\mathcal{T}(\pi_i)$ have already been constructed, for $i \in \{1,2\}$. Thus, $[\![A(\pi_i), p_i, q_i]\!] = [\![\pi_i]\!]$ for some states $p_i, q_i \in Q_i$. A natural idea for defining an NFA for $\pi_1 \cap \pi_2$ is to apply a product construction to A_1 and A_2. A naive attempt to do this is bound to fail because a run of A_1 in T and a run of A_2 in T, both starting in a tree node u and ending in a tree node v, may proceed along different paths. More precisely, the two runs both travel along the unique shortest from u to v, but they may make different "detours" from this shortest path. In order to eliminate this problem and make the product construction available, we modify $A(\pi_1)$ and $A(\pi_2)$ by admitting additional test transitions that allow to short-cut the mentioned detours. These modified NFAs can always travel along the shortest path without any detours, and thus the product construction can be used.

Before we can construct $A(\pi_1 \cap \pi_2)$, we make a digression to introduce the mentioned modification of NFAs. Let $\mathcal{T} = (S, \delta, \mathrm{Acc})$ be a TWAPTA and $A = (Q, \rightarrow_A)$ an NFA over $\mathcal{T}$. Define the relation $\mathrm{loop}_A \subseteq \mathsf{A}^* \times Q \times Q$ as the smallest set such that:

(i) for all $u \in \mathsf{A}^*$ and $q \in Q$ we have $(u, q, q) \in \mathrm{loop}_A$,
(ii) if $(ua, p', q') \in \mathrm{loop}_A$, $p \xrightarrow{a}_A p'$ and $q' \xrightarrow{\overline{a}}_A q$, then $(u, p, q) \in \mathrm{loop}_A$,
(iii) if $(u, p', q') \in \mathrm{loop}_A$, $p \xrightarrow{\overline{a}}_A p'$, and $q' \xrightarrow{a}_A q$, then $(ua, p, q) \in \mathrm{loop}_A$,
(iv) if $(u, p, r) \in \mathrm{loop}_A$ and $(u, r, q) \in \mathrm{loop}_A$, then $(u, p, q) \in \mathrm{loop}_A$, and
(v) if $u \in [\![\mathcal{T}, s]\!]$ and $p \xrightarrow{\mathcal{T},s}_A q$ for $s \in S$, then $(u, p, q) \in \mathrm{loop}_A$.

Intuitively, loop_A describes detours, i.e., (parts of) a run of A that start at some node in the tree and eventually return to the very same node. It is not too difficult to prove the following.

Lemma 5. *We have* $(u, p, q) \in \mathrm{loop}_A$ *if and only if* $(u, p) \Rightarrow_A^* (u, q)$.

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

Lemma 6. *There is a TWAPTA* $\mathcal{U} = (S', \delta', \mathrm{Acc}')$ *with* $S' = S \uplus (Q \times Q)$ *s.t.*

(i) $[\![\mathcal{U}, s]\!] = [\![\mathcal{T}, s]\!]$ *for all* $s \in S$,
(ii) $[\![\mathcal{U}, (p,q)]\!] = \{u \in \mathsf{A}^* \mid (u, p, q) \in \mathrm{loop}_A\}$ *for all* $(p, q) \in Q \times Q$, *and*
(iii) $|\mathrm{Acc}'| = |\mathrm{Acc}|$.

Now define a new NFA $B = (Q, \rightarrow_B)$ over the TWAPTA $\mathcal{U}$, that results from A by adding for every pair $(p, q) \in Q \times Q$, the test transition $p \xrightarrow{\mathcal{U},(p,q)}_B q$. The following lemma shows that our modification did not damage the NFA.

Lemma 7. *Let* $u, v \in \mathsf{A}^*$ *and let* $p, q \in Q$. *Then* $(u, v) \in [\![A, p, q]\!]$ *iff* $(u, v) \in [\![B, p, q]\!]$.

We now return to the construction of $A(\pi_1 \cap \pi_2)$ and $\mathcal{T}(\pi_1 \cap \pi_2)$ from $A(\pi_1)$, $A(\pi_2)$, $\mathcal{T}(\pi_1)$, and $\mathcal{T}(\pi_2)$. For $i \in \{1,2\}$, we first construct the NFA $B(\pi_i)$ over the TWAPTA $\mathcal{U}(\pi_i) = (S'_i, \delta'_i, \mathrm{Acc}'_i)$ as described above. Note that $|S'_i| = |S_i| + |Q_i|^2$. We take $\mathcal{T}(\pi_1 \cap \pi_2) = \mathcal{U}(\pi_1) \uplus \mathcal{U}(\pi_2)$. The NFA $A(\pi_1 \cap \pi_2)$ is the product automaton of $B(\pi_1) = (Q_1, \rightarrow_{B(\pi_1)})$ and $B(\pi_2) = (Q_2, \rightarrow_{B(\pi_2)})$, where test transitions can be carried out asynchronously:

- The state set of $A(\pi_1 \cap \pi_2)$ is $Q_1 \times Q_2$.
- For $a \in \mathsf{A} \cup \overline{\mathsf{A}}$ we have $(r_1, r_2) \xrightarrow{a}_{A(\pi_1 \cap \pi_2)} (r'_1, r'_2)$ if and only if $r_1 \xrightarrow{a}_{B(\pi_1)} r'_1$ and $r_2 \xrightarrow{a}_{B(\pi_2)} r'_2$.
- For $s \in S'_1 \uplus S'_2$ we have the test transition $(r_1, r_2) \xrightarrow{\mathcal{T}(\pi_1 \cap \pi_2), s}_{A(\pi_1 \cap \pi_2)} (r'_1, r'_2)$ if and only if (i) $s \in S'_1$, $r_2 = r'_2$, and $r_1 \xrightarrow{\mathcal{U}(\pi_1), s}_{B(\pi_1)} r'_1$ or (ii) $s \in S'_2$, $r_1 = r'_1$, and $r_2 \xrightarrow{\mathcal{U}(\pi_2), s}_{B(\pi_1)} r'_2$.

It is possible to show that $[\![A(\pi_1 \cap \pi_2), (p_1, p_2), (q_1, q_2)]\!] = [\![\pi_1 \cap \pi_2]\!]$.This finishes the inductive translation of ICPDL formulas and programs into automata. A careful analysis of the constructions outlined above, allows us to inductively establish the following bounds.

Lemma 8. *For every ICPDL formula ψ and every ICPDL program π we have:*

1. *If $\mathcal{T}(\psi) = (S, \delta, \mathrm{Acc})$, then $|S| \le 2^{|\psi|^2}$ and $|\mathrm{Acc}| \le |\psi|$.*
2. *If $A(\pi) = (Q, \rightarrow_{A(\pi)})$ and $\mathcal{T}(\pi) = (S, \delta, \mathrm{Acc})$ then $|Q| \le 2^{|\pi|}$, $|S| \le 2^{|\pi|^2}$, and $|\mathrm{Acc}| \le |\pi|$.*

The *double* exponential bound in Point 1 of Lemma 8 is due to the fact that the construction for dealing with program intersection blows up the size of NFAs quadratically. In contrast, all other constructions involve only a linear blowup.

5.2 Wrapping Up

It is now easy to decide ω-regular tree satisfiability in ICPDL. Let $\mathcal{T}_0$ be a TWAPTA over 2^{P}-labeled A-trees and let φ be an ICPDL formula with $\mathsf{prop}(\varphi) \subseteq \mathsf{P}$ and $\mathsf{prog}(\varphi) \subseteq \mathsf{A}$. There is a state s of $\mathcal{T}(\varphi)$ such that $[\![\mathcal{T}(\varphi), s]\!]_T = [\![\varphi]\!]_T$ for all 2^{P}-labeled A-trees T. Let the TWAPTA $\mathcal{T}$ be the intersection of $\mathcal{T}_0$ and $\mathcal{T}(\varphi)$ (taking the intersection of TWAPTAs is trivial an can be done in linear time), where s becomes the initial state of $\mathcal{T}(\varphi)$. Clearly, $L(\mathcal{T}) \neq \emptyset$ if and only if there exists some tree $T \in L(\mathcal{T}_0)$ with $(T, \varepsilon) \models \varphi$. By Lemma 8 and Theorem 3, we thus obtain a $\mathsf{2EXP}$ upper bound for ω-regular tree satisfiability in ICPDL. A matching lower bound is obtained by a straightforward reduction of satisfiability in ICPDL in tree-shaped Kripke structures. It was shown in [11] that this problem is $\mathsf{2EXP}$-hard.

Theorem 5. *ω-regular tree satisfiability in ICPDL is $\mathsf{2EXP}$-complete.*

Together with Theorem 4, this finally proves our main result Theorem 1. It is interesting to note that the bound given in Point 1 of Lemma 8 improves to single exponential if the intersection height (which can be defined in the obvious way) of ICPDL programs is bounded by a constant. Thus, we actually obtain EXP-completeness for this case.

6 Negation of Atomic Programs

We consider extensions of IPDL and ICPDL with negation of programs. It is well known that adding full program negation renders PDL undecidable [9], whereas PDL with program negation restricted to atomic programs remains decidable and EXP-complete [13]. In this section, we show that IPDL and hence also ICPDL become undecidable already when extended with atomic program negation. Since intersection of programs can be defined in terms of program union and (full) program negation, this also yields an alternative proof of the undecidability of PDL with full program negation.

Our proof proceeds by reduction from the undecidable tiling problem of the first quadrant of the plane [3]. A *tiling system* $\mathcal{T} = (T, H, V)$ consists of a finite set of *tile types* T and horizontal and vertical matching relations $H, V \subseteq T \times T$. A *solution* to $\mathcal{T}$ is a mapping $\tau : \mathbb{N} \times \mathbb{N} \to T$ such that for all $(x, y) \in \mathbb{N} \times \mathbb{N}$, we have

- if $\tau(x, y) = t$ and $\tau(x + 1, y) = t'$, then $(t, t') \in H$, and
- if $\tau(x, y) = t$ and $\tau(x, y + 1) = t'$, then $(t, t') \in V$.

The *tiling problem* is to decide, given a tiling system $\mathcal{T}$, whether $\mathcal{T}$ 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., $[\![\neg a]\!]_K = (X \times X) \setminus [\![a]\!]_K$. To reduce the tiling problem to satisfiability in IPDL$^{(\neg)}$, we give a translation of tiling systems $\mathcal{T} = (T, H, V)$ into formulas $\varphi_\mathcal{T}$ of IPDL$^{(\neg)}$ such that $\mathcal{T}$ has a solution if and only if $\varphi_\mathcal{T}$ is satisfiable. In the formula $\varphi_\mathcal{T}$, we use two atomic programs a_x and a_y for representing the grid $\mathbb{N} \times \mathbb{N}$ and we use the elements of T as atomic propositions for representing tile types. More precisely, $\varphi_\mathcal{T}$ is a conjunction consisting of the following conjuncts:

(a) every element of a (connected) model of $\varphi_\mathcal{T}$ 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:

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

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

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

Lemma 9. *$\mathcal{T}$ has a solution if and only if $\varphi_\mathcal{T}$ is satisfiable.*

We have thus established the following result.

Theorem 6. *Satisfiability in IPDL$^{(\neg)}$ is undecidable.*

References

1. L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. J. Marx, and M. de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115-135, 2005.
2. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.
3. R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66, 1966.
4. R. Danecki. Nondeterministic Propositional Dynamic Logic with intersection is decidable. In *Proc. 5th Symp. Computation Theory*, LNCS 208, pages 34–53, 1984.
5. L. Farinas Del Cerro and E. Orlowska. DAL-a logic for data analysis. *Theoretical Computer Science*, 36(2-3):251–264, 1985.
6. M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
7. G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. AAAI94*, pages 205–212, 1994.
8. S. Göller and M. Lohrey. Infinite state model-checking of propositional dynamic logics. In *Proc. CSL 2006*, LNCS 4207, pages 349–364. Springer, 2006.
9. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of computing. The MIT Press, 2000.
10. W. van der Hoek and J.J. Meyer. A complete epistemic logic for multiple agents – Combining distributed and common knowledge. *Epistemic Logic and the Theory of Games and Decisions*, pages 35–68. Kluwer, 1997.
11. M. Lange and C. Lutz. 2-ExpTime Lower Bounds for Propositional Dynamic Logics with Intersection. *Journal of Symbolic Logic*, 70(4):1072–1086, 2005.
12. C. Lutz. PDL with intersection and converse is decidable. In *Proc. CSL 2005*, LNCS 3634, pages 413–427. Springer, 2005.
13. C. Lutz and D. Walther. PDL with negation of atomic programs. *Journal of Applied Non-Classical Logics*, 15(2):189–213, 2005.
14. J. Meyer. Dynamic logic for reasoning about actions and agents. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 281–311. Kluwer Academic Publishers, 2000.
15. D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987.
16. V. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20:231–254, 1980.
17. B. ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 328–337. ACM Press, 2006.
18. H. P. van Ditmarsch, W. van der Hoek, and B. P. Kooi. Concurrent dynamic epistemic logic for MAS. In *Proc. AAMAS 2003*, pages 201–208. ACM Press, 2003.
19. M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In *Proc. Logics of Programs*, LNCS 193, pages 413–423. Springer, 1985.
20. M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. ICALP '98*, LNCS 1443, pages 628–641. Springer, 1998.