PDL with negation of atomic programs

Carsten Lutz[†] — **Dirk Walther**[‡]

[†] *Institute for Theoretical Computer Science,*
TU Dresden, Germany

lutz@tcs.inf.tu-dresden.de

[‡] *Department of Computer Science,*
University of Liverpool, UK

dirk@csc.liv.ac.uk

ABSTRACT. *Propositional dynamic logic (PDL) is one of the most successful variants of modal logic. To make it even more useful for applications, many extensions of PDL have been considered in the literature. A very natural and useful such extension is with negation of programs. Unfortunately, as long-known, reasoning with the resulting logic is undecidable. In this paper, we consider the extension of PDL with negation of atomic programs, only. We argue that this logic is still useful, e.g. in the context of description logics, and prove that satisfiability is decidable and* EXPTIME*-complete using an approach based on Büchi tree automata.*

KEYWORDS: *propositional dynamic logic, program negation, complexity.*

1. Introduction

Propositional dynamic logic (PDL) is a variant of propositional modal logic that has been developed in the late seventies as a tool for reasoning about programs [PRA 76, FIS 77, FIS 79, HAR 84, HAR 00]. Since then, PDL was used rather successfully in a large number of application areas such as reasoning about knowledge [FAG 95], reasoning about actions [De 95, PRE 96], description logics [GIA 94], and others. Starting almost with its invention around 1979 [FIS 79], many extensions of PDL have been proposed with the goal to enhance the expressive power and make PDL even more applicable; see e.g. [PAS 91, HAR 84, HAR 00]. Some of these extensions are tailored toward specific application areas, such as the *halt* predicate that allows to state termination in the context of reasoning about programs [HAR 78]. The majority of proposed extensions, however, is of a general nature and has been employed in many different application areas—for instance, the extension of PDL with the widely applied converse operator [VAR 85].

Among the general purpose extensions of PDL, two of the most obvious ones are the addition of program intersection "$\cap$" and of program negation "$\neg$" [DAN 84, HAR 84, HAR 00]. Since PDL already provides for program union "$\cup$", the latter is more general than the former: $\alpha \cap \beta$ can simply be expressed as $\neg(\neg\alpha \cup \neg\beta)$. The main obstacle for using these two extensions in practical applications is that they are problematic w.r.t. their computational properties: first, adding intersection destroys many of the nice model-theoretic properties of PDL. The only known algorithm for reasoning in the resulting logic PDL$^{\cap}$ is the quite intricate one given in [DAN 84]. Up to now, it is unknown whether the provided 2-EXPTIME upper bound is tight—in contrast to EXPTIME-complete reasoning in PDL. Second, the situation with PDL extended with negation (PDL$^{\neg}$) is even worse: it was observed quite early in 1984 that reasoning in PDL$^{\neg}$ is undecidable [HAR 84].

This undecidability was often regretted [HAR 84, PAS 91, BRO 03], in particular since reasoning in PDL$^{\neg}$ would be quite interesting for a number of application areas. To illustrate the usefulness of this logic, let us give three examples of its expressive power: first, it was already noted that negation can be employed to express intersection. Intersection, in turn, is very useful for reasoning about programs since it allows to capture the parallel execution of programs. Second, program negation allows to express the universal modality $\Box_U \varphi$ by writing $[a]\varphi \wedge [\neg a]\varphi$, with a an arbitrary atomic program. The universal modality is a very useful extension of modal logics that comes handy in many applications; see e.g. [GOR 92]. Third, program negation can be used to express the window operator $\boxbar_a$ [HUM 83, GAR 87, GOR 90], whose semantics is as follows: $\boxbar_a \varphi$ holds at a world w iff φ holding at a world w' implies that w' is a-accessible from w. In PDL$^{\neg}$, we can thus just write $[\neg a]\neg\varphi$ instead of $\boxbar_a \varphi$. The window operator can be viewed as expressing sufficiency in contrast to the standard box operator of modal logic, which expresses necessity. Moreover, the window operator has important applications, e.g. in description logics [LUT 00].

Due to the usefulness of program negation, it is natural to attempt the identification of fragments of PDL$^{\neg}$ that still capture some of the useful properties of program negation, but are well-behaved in a computational sense. One candidate for such a fragment is PDL$^{\cap}$. As has already been noted, this fragment is indeed decidable, but has a quite intricate model theory. The purpose of this paper is to explore another natural option: PDL$^{(\neg)}$, the fragment of PDL$^{\neg}$ that allows the application of program negation to *atomic* programs, only. Note that, in PDL$^{(\neg)}$, negated atomic programs may be freely used inside other program operators. We show that reasoning in PDL$^{(\neg)}$ is decidable, and EXPTIME-complete—thus not harder than reasoning in PDL itself. Moreover, PDL$^{(\neg)}$ has a simpler model theory than PDL$^{\cap}$: we are able to use a decision procedure that is an extension of the standard automata-based decision procedure for PDL [VAR 86], and of the standard automata-based decision procedure for Boolean modal logic [LUT 01]. Finally, we claim that PDL$^{(\neg)}$ is still useful for applications: while intersection cannot be expressed any more, the universal modality and the window operator are still available.

To give some more concrete examples of the usefulness of PDL$^{(\neg)}$, we take a description logic perspective. Description logics are a family of logics that originated in artificial intelligence as a tool for the representation of conceptual knowledge [BAA 03]. It is well-known that many description logics (DLs) are notational variants of modal logics [SCH 91, GIA 94]. In particular, the description logic $\mathcal{ALC}_{\rm reg}$, which extends the basic DL $\mathcal{ALC}$ with regular expressions on roles, corresponds to PDL [GIA 94]. More precisely, DL concepts can be understood as PDL formulas, and DL roles as PDL programs. Thus, the extension $\mathcal{ALC}^{(\neg)}_{\rm reg}$ of $\mathcal{ALC}_{\rm reg}$ with negation of atomic roles is a notational variant of PDL$^{(\neg)}$. We give two examples of knowledge representation with $\mathcal{ALC}^{(\neg)}_{\rm reg}$. These examples, which use DL syntax rather than PDL syntax, illustrate that the combination of regular expressions on roles and of atomic negation of roles is a very useful one.

1. Assume that we want to use $\mathcal{ALC}^{(\neg)}_{\rm reg}$ to capture the relevant notions of universities and their students. In particular, we want to describe the preferences that universities have when selecting students. Since some private universities prefer to admit students whose ancestors donated money to the university, we could write

$$\mathsf{UniversityX} \rightarrow \forall \mathsf{prefer}.(\mathsf{Top\text{-}SAT} \sqcup \exists \mathsf{parent}^{+}.\mathsf{Donator})$$

to state that all students preferred by University X have either passed the (American) SAT test with a top score or have an ancestor that donated money. But perhaps the policy of University X is even stronger in that it prefers *all* students whose ancestors have donated money. This can be expressed using the window operator (that can, in turn, be expressed via atomic negation):

$$\mathsf{UniversityX} \rightarrow \forall \neg \mathsf{prefer}.\neg(\exists \mathsf{parent}^{+}.\mathsf{Donator})$$

states that all students with donating ancestors are preferred.

2. Suppose that we want to use $\mathcal{ALC}^{(\neg)}_{\rm reg}$ to talk about trust and mistrust among negotiating parties. Also assume that we have a very strong notion of trust, namely that it is transitive: if I trust x, and x trusts y, then I trust y as well. An analogous assumption for mistrust should clearly not be made. Then, we can model mistrust by using an atomic role $\mathsf{mistrust}$, and trust by using $(\neg\mathsf{mistrust})^*$. We can now say, e.g., that I trust some politicians and never mistrust a family member :

$$\exists(\neg\mathsf{mistrust})^*.\mathsf{Politician} \sqcap \forall \mathsf{mistrust}.\neg\mathsf{Familymember}.$$

Note that reversing the roles of trust and mistrust does not work: first, to achieve transitivity of trust, we'd have to introduce an atomic $\mathsf{direct\text{-}trust}$ relation. And second, we could then only speak about the negation of $\mathsf{direct\text{-}trust}$, but not about the negation of $\mathsf{direct\text{-}trust}^*$, which corresponds to mistrust.

This paper is organized as follows.[1] In Section 2, we introduce PDL$^{\neg}$ and its fragment PDL$^{(\neg)}$. For illustrative purposes, we give a proof of the undecidability of

1. Note that the current paper is an extended version of [LUT 04].

full PDL$^\neg$. In preparation for the automata-based decision procedure, we introduce a variant APDL$^{(\neg)}$ of PDL$^{(\neg)}$ in Section 3. This variant uses finite automata rather than regular expressions as modal parameters and is better suited for automata-based decision procedures. As one of the two core parts of our decision procedure, in Section 4 we then abstract models of PDL$^{(\neg)}$-formulas φ to Hintikka-trees for φ . The other core part is presented in Section 5, where we define Büchi-tree automata that can be used to check for existence of Hintikka-trees (and thus models) of PDL$^{(\neg)}$ formulas. We conclude in Section 6 giving some possible directions for future research.

2. PDL with Negation

In this section, we introduce propositional dynamic logic (PDL) with negation of programs. We start with defining full PDL$^\neg$, i.e. PDL extended with negation of (possibly complex) programs. Then, the logics PDL and PDL$^{(\neg)}$, are defined as fragments of PDL$^\neg$.

DEFINITION 1 (PDL$^\neg$ SYNTAX). — *Let* Φ_0 *and* Π_0 *be countably infinite and disjoint sets of* propositional letters *and* atomic programs, *respectively. Then the set* $\Pi^\neg$ *of* PDL$^\neg$-programs *and the set* $\Phi^\neg$ *of* PDL$^\neg$-formulas *are defined by simultaneous induction, i.e., they are the smallest sets such that:*

– $\Phi_0 \subseteq \Phi^\neg$ *and* $\Pi_0 \subseteq \Pi^\neg$*;*

– *if* $\varphi, \psi \in \Phi^\neg$*, then* $\{\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi\} \subseteq \Phi^\neg$*;*

– *if* $\pi_1, \pi_2 \in \Pi^\neg$*, then* $\{\neg\pi_1, \pi_1 \cup \pi_2, \pi_1;\pi_2, \pi_1^*\} \subseteq \Pi^\neg$*;*

– *if* $\pi \in \Pi^\neg$*, and* $\varphi \in \Phi^\neg$*, then* $\{\langle\pi\rangle\varphi, [\pi]\varphi\} \subseteq \Phi^\neg$*;*

– *if* $\varphi \in \Phi^\neg$*, then* $\varphi? \in \Pi^\neg$

We use $\top$ *as abbreviation for an arbitrary propositional tautology, and* $\bot$ *as abbreviation for* $\neg\top$*. Moreover, for* $\pi, \pi' \in \Pi^\neg$ *we use* $\pi \cap \pi'$ *as abbreviation for* $\neg(\neg\pi \cup \neg\pi')$*.*

A formula $\varphi \in \Phi^\neg$ *is called a* PDL$^{(\neg)}$-formula *(*PDL-formula*) if, in* φ*, negation occurs only in front of atomic programs and formulas (only in front of formulas).*

Throughout this paper, the operator $\langle\pi\rangle$ is called the diamond operator, $[\pi]$ is called the box operator, and programs of the form $\psi?$ are called *tests*. Let us note how formulas of PDL$^\neg$ can be converted into concepts of the description logic $\mathcal{ALC}^{(\neg)}_{\text{reg}}$ mentioned in the introduction: simply replace $\wedge$, $\vee$, $\langle\pi\rangle\psi$, and $[\pi]\psi$ with $\sqcap$, $\sqcup$, $\exists\pi.\psi$, and $\forall\pi.\psi$, respectively.

DEFINITION 2 (PDL$^\neg$ SEMANTICS). — *Let* $\mathcal{M} = (W, \mathcal{R}, V)$ *be a* Kripke structure *where* W *is the* set of worlds, $\mathcal{R}$ *is a family of* accessibility relations *for atomic programs* $\{R_\pi \subseteq W^2 \mid \pi \in \Pi_0\}$*, and* $V : \Phi_0 \to 2^W$ *is a* valuation function. *In the following, we define accessibility relations for compound programs and the satisfaction relation* $\models$ *by simultaneous induction, where* $\cdot^*$ *denotes the reflexive-transitive closure:*

$$\begin{array}{lcl} R_{\varphi?} & := & \{(u,u) \in W^2 \mid \mathcal{M}, u \models \varphi\} \\ R_{\neg\pi} & := & W^2 \setminus R_\pi \end{array}$$

$$\begin{array}{rcl}
R_{\pi_1\cup\pi_2} & := & R_{\pi_1} \cup R_{\pi_2} \\
R_{\pi_1;\pi_2} & := & R_{\pi_1} \circ R_{\pi_2} \\
R_{\pi^*} & := & (R_\pi)^* \\
\mathcal{M}, u \models p & \textit{iff} & u \in V(p) \textit{ for any } p \in \Phi \\
\mathcal{M}, u \models \neg\varphi & \textit{iff} & \mathcal{M}, u \not\models \varphi \\
\mathcal{M}, u \models \varphi_1 \vee \varphi_2 & \textit{iff} & \mathcal{M}, u \models \varphi_1 \textit{ or } \mathcal{M}, u \models \varphi_2 \\
\mathcal{M}, u \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \mathcal{M}, u \models \varphi_1 \textit{ and } \mathcal{M}, u \models \varphi_2 \\
\mathcal{M}, u \models \langle\pi\rangle\varphi & \textit{iff} & \textit{there is a } v \in W \textit{ with } (u,v) \in R_\pi \textit{ and } \mathcal{M}, v \models \varphi \\
\mathcal{M}, u \models [\pi]\varphi & \textit{iff} & \textit{for all } v \in W,\ (u,v) \in R_\pi \textit{ implies } \mathcal{M}, v \models \varphi
\end{array}$$

If $\mathcal{M}, u \models \varphi$ for some formula $\varphi \in \Phi^\neg$ and world $u \in W$, then φ is true *at u in $\mathcal{M}$, and $\mathcal{M}$ is called* model *of φ. A formula is* satisfiable *if it has a model. It is called* valid *if it is true at all worlds in all Kripke structures.*

It is well-known that satisfiability and validity of PDL$^\neg$-formulas are undecidable [HAR 84]. Since this can be established in a very simple way, we give a proof for illustrative purposes.

The most common proof technique is simply to state that undecidability of PDL$^\neg$ is an immediate consequence of the undecidability of the relation algebra that has the operators $\neg$, $\cap$, $\cup$, and $\circ$ (composition). Though this is of course a valid method, we believe that it is more instructive to directly reduce the undecidable word-problem for finitely presented semi-groups [MAT 67], which is the standard approach for proving undecidability of the afore mentioned relation algebras; see e.g. [AND 01]. This word problem can be formulated as follows: given a set of word equations $E = \{u_1 \approx v_1, \ldots, u_k \approx v_k\}$ and another equation $u \approx v$, the task is to decide whether every semigroup satisfying E also satisfies $u \approx v$. To reduce this problem to PDL$^\neg$-satisfiability, we need to introduce the universal modality $\Box_U\varphi$, which has the following semantics:

$$\mathcal{M}, u \models \Box_U\varphi \quad \text{iff} \quad \mathcal{M}, v \models \varphi \text{ for all } v \in W.$$

Clearly, in PDL$^\neg$ we can replace $\Box_U\varphi$ with the equivalent $[a]\varphi \wedge [\neg a]\varphi$, where $a \in \Pi_0$ is an arbitrary atomic program. Using the universal modality, the reduction is now easy: we assume that, for every generator of the semi-group, there is an atomic program of the same name, and then note that $\{u_1 \approx v_1, \ldots, u_k \approx v_k\}$ implies $u \approx v$ if and only if the following formula is unsatisfiable:

$$\Big(\langle u \cap \neg v\rangle\top \vee \langle \neg u \cap v\rangle\top\Big) \wedge \Box_U\Big(\bigwedge_{i=1..k} [u_i \cap \neg v_i]\bot \wedge [v_i \cap \neg u_i]\bot\Big).$$

Here, we assume that the symbols of the words u_i and v_i (and of u and v) are separated by program composition ";".

Proof sketch: For both directions, the contrapositive is proved. ("only if"). If the set of equations $E = \{u_1 \approx v_1, \ldots, u_k \approx v_k\}$ does not imply $u \approx v$, then there is a semigroup $\mathfrak{A} = (A, ;, a_1, \ldots, a_n)$ such that $\mathfrak{A}$ satisfies E but not $u \approx v$. From this semigroup, we can construct a model $\mathcal{M}$ for the reduction formula as follows: $\mathcal{M} := (W, \{R_{a_1}, \ldots, R_{a_n}\}, V)$ where

– $W = A \cup \{\varepsilon\}$ where $\varepsilon \notin A$,

– $R_{a_i} = \{(\varepsilon, a_i)\} \cup \{(w, w') \in A^2 \mid \mathcal{A}$ satisfies $w' \approx wa_i\}$ for any $1 \leq i \leq n$,

– V is arbitrary.

It can be checked that the reduction formulas is true at $\varepsilon \in W$ in $\mathcal{M}$.
("if"). Suppose that the reduction formula is satisfiable in a model $\mathcal{M} = (W, \{R_{a_1}, \ldots, R_{a_n}\}, V)$ at a world $u \in W$. Then we construct a semi-group as follows: $\mathfrak{A} = (A, ;, a_1, \ldots, a_n)$, where

– $A = 2^W \times 2^W$,

– $a_i = R_{a_i}$ for $1 \leq i \leq n$,

– $w; w' = R_{ww'}$ for any two words w, w' (i.e. compositions $a_{i_1}; a_{i_2}; \cdots; a_{i_k}$ with $i_1, \ldots, i_k \in \{1, \ldots, n\}$).

Now the first conjunct of the reduction formula implies that $\mathfrak{A}$ does not satisfy $u \approx v$, and the second conjunct implies that $\mathfrak{A}$ satisfies E. *End of proof sketch.*

Since PDL$^{\neg}$ is a very useful logic for a large number of purposes, this undecidability result is rather disappointing. As has been argued in the introduction, it is thus a natural idea to search for decidable fragments of PDL$^{\neg}$ that still extend PDL in a useful way. In the remainder of this paper, we will prove that PDL$^{(\neg)}$ is such a fragment. Note that, in PDL$^{(\neg)}$, we can still define the universal modality as described above. Also note that we can use negated atomic programs nested inside other program operators. Anyway, it is no longer possible to encode the word problem in PDL$^{(\neg)}$ since one can express equivalence of atomic programs, but not equivalence of compositions of atomic programs.

3. An Automata-based Variant of PDL$^{(\neg)}$

Similar to some related results in [VAR 86], our decidability proof is based on Büchi-automata on infinite trees. It has turned out that, for such proofs, it is rather convenient to use variants of PDL in which complex programs are described by means of automata on finite words, rather than by regular expressions. Therefore, in this section we define a corresponding variant APDL$^{(\neg)}$ of PDL$^{(\neg)}$.

DEFINITION 3 (FINITE AUTOMATA). — *A* (nondeterministic) finite automaton (NFA) $\mathcal{A}$ *is a quintuple* $(Q, \Sigma, q_0, \Delta, F)$ *where*

– Q *is a finite set of* states,

– Σ *is a finite* alphabet,

– q_0 *is an* initial state,

– $\Delta : Q \times \Sigma \to 2^Q$ *is a (partial)* transition function, *and*

– $F \subseteq Q$ *is the set of* accepting states.

The function Δ *can be inductively extended to a function from* $Q \times \Sigma^*$ *to* 2^Q *in a* *natural way:*

– $\Delta(q, \varepsilon) := \{q\}$, *where* ε *is the* empty word*;*

– $\Delta(q, wa) := \{q'' \in Q \mid q'' \in \Delta(q', a)$ *for some* $q' \in \Delta(q, w)\}$.

A sequence $p_0, \ldots, p_n \in Q$, $n \geq 0$, *is a* run *of* $\mathcal{A}$ *on the word* $a_1 \cdots a_n \in \Sigma^*$ *if* $p_0 = q_0$, $p_i \in \Delta(p_{i-1}, a_i)$ *for* $0 < i \leq n$, *and* $p_n \in F$. *A word* $w \in \Sigma^*$ *is* accepted *by* $\mathcal{A}$ *if there exists a run of* $\mathcal{A}$ *on* w. *The* language *accepted by* $\mathcal{A}$ *is the set* $\mathcal{L}(\mathcal{A}) := \{w \in \Sigma^* \mid w$ *is accepted by* $\mathcal{A}\}$.

To obtain APDL$^{(\neg)}$ from PDL$^{(\neg)}$, we replace complex programs (i.e. regular expressions) inside boxes and diamonds with automata. For the sake of exactness, we give the complete definition.

DEFINITION 4 (APDL$^{(\neg)}$ SYNTAX). — *The set* $\Pi_0^{(\neg)}$ *of* program literals *is defined as* $\{a, \neg a \mid a \in \Pi_0\}$. *The sets* $A\Pi^{(\neg)}$ *of* program automata *and* $A\Phi^{(\neg)}$ *of* APDL$^{(\neg)}$-formulas *are defined by simultaneous induction, i.e.,* $A\Pi^{(\neg)}$ *and* $A\Phi^{(\neg)}$ *are the smallest sets such that:*

– $\Phi_0 \subseteq A\Phi^{(\neg)}$*;*

– *if* $\varphi, \psi \in A\Phi^{(\neg)}$, *then* $\{\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi\} \subseteq A\Phi^{(\neg)}$*;*

– *if* $\alpha \in A\Pi^{(\neg)}$ *and* $\varphi \in A\Phi^{(\neg)}$, *then* $\{\langle\alpha\rangle\varphi, [\alpha]\varphi\} \subseteq A\Phi^{(\neg)}$*;*

– *if* α *is a finite automaton with alphabet* $\Sigma \subseteq \Pi_0^{(\neg)} \cup \{\psi? \mid \psi \in A\Phi^{(\neg)}\}$, *then* $\alpha \in A\Pi^{(\neg)}$

Note that the alphabet of program automata is composed of atomic programs, of negated atomic programs, and of tests.

DEFINITION 5 (APDL$^{(\neg)}$ SEMANTICS). — *Let* $\mathcal{M} = (W, \mathcal{R}, V)$ *be a Kripke structure as in Definition 2. We inductively define a relation* R *mapping each program literal, each test, and each program automaton to a binary relation over* W. *This is done simultaneously with the definition of the satisfaction relation* $\models$*:*

$$\begin{array}{rcl}
R(a) & := & R_a \text{ for each } a \in \Pi_0 \\
R(\neg a) & := & W^2 \setminus R_a \text{ for each } a \in \Pi_0 \\
R(\psi?) & := & \{(u,u) \in W^2 \mid \mathcal{M}, u \models \psi\} \\
R(\alpha) & := & \{(u,v) \in W^2 \mid \text{there is a word } w = w_1 \cdots w_m \in \mathcal{L}(\alpha), \\
 & & m \geq 0, \text{ and worlds } u_0, \ldots, u_m \in W \text{ such that} \\
 & & u = u_0 R(w_1) u_1 R(w_2) \cdots u_{m-1} R(w_m) u_m = v\} \\
\mathcal{M}, u \models p & \text{iff} & u \in V(p) \text{ for any } p \in \Phi, \\
\mathcal{M}, u \models \neg\varphi & \text{iff} & \mathcal{M}, u \not\models \varphi, \\
\mathcal{M}, u \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathcal{M}, u \models \varphi_1 \text{ or } \mathcal{M}, u \models \varphi_2, \\
\mathcal{M}, u \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{M}, u \models \varphi_1 \text{ and } \mathcal{M}, u \models \varphi_2, \\
\mathcal{M}, u \models \langle\alpha\rangle\varphi & \text{iff} & \text{there is a } u' \in W \text{ with } (u,u') \in R(\alpha) \text{ and } \mathcal{M}, u' \models \varphi, \\
\mathcal{M}, u \models [\alpha]\varphi & \text{iff} & \text{for all } u' \in W, (u,u') \in R(\alpha) \text{ implies } \mathcal{M}, u' \models \varphi.
\end{array}$$

Truth, satisfiability, *and* validity *are defined as in the PDL*$^{\neg}$ *case.*

Since every language defined by a regular expression can also be accepted by a finite automaton and vice versa [KLE 56], it is straightforward to verify that PDL$^{(\neg)}$ and APDL$^{(\neg)}$ have the same expressive power. Moreover, upper complexity bounds carry over from APDL$^{(\neg)}$ to PDL$^{(\neg)}$ since conversion of regular expressions to finite automata can be done with at most a polynomial blow-up in size. Note that the converse does not hold true: since there exist automata $\mathcal{A}$ accepting languages that can only be defined by regular expressions whose size is exponential in the size of $\mathcal{A}$ [EHR 74], there are APDL$^{(\neg)}$-formulas φ such that the length of all equivalent PDL$^{(\neg)}$-formulas is exponential in the length of φ.

It is interesting to note that, in many automata-based decision procedures for variants of PDL, a *deterministic* version of APDL is used, i.e. a variant of APDL in which there may be at most one successor for each world and each atomic program [VAR 86]. In a second step, satisfiability in the non-deterministic APDL-variant is then reduced to satisfiability in the deterministic one. We cannot take this approach here since we cannot w.l.o.g. assume that both atomic programs *and their negations* are deterministic. Indeed, this would correspond to limiting the size of Kripke structures to only two worlds.

4. Hintikka-trees

This section provides a core step toward using Büchi-tree automata for deciding the satisfiability of APDL$^{(\neg)}$-formulas. The intuition behind this approach is as follows: to decide the satisfiability of an APDL$^{(\neg)}$-formula φ, we translate it into a Büchi-tree automaton $\mathcal{B}_\varphi$ such that the trees accepted by the automaton correspond in some way to models of the formula φ. To decide satisfiability of φ, it then remains to perform a simple emptiness-test on the automaton $\mathcal{B}_\varphi$: the accepted language will be non-empty if and only if φ has a model.

In the case of APDL$^{(\neg)}$, one obstacle to this approach is that APDL$^{(\neg)}$ does not enjoy the *tree model property (TMP)*, i.e., there are APDL$^{(\neg)}$-formulas that are satisfiable only in non-tree models. For example, for each $n \in \mathbb{N}$ the following PDL$^{(\neg)}$-formula enforces a cycle of length n:

$$\psi_1^n \wedge \langle a\rangle(\psi_2^n \wedge \langle a\rangle(\cdots(\psi_n^n \wedge [\neg a]\neg\psi_1^n)\cdots)),$$

where, for $1 \leq i \leq n$, $\psi_i^n = p_1 \wedge \cdots \wedge \neg p_i \wedge \cdots \wedge p_n$ with $p_1, \ldots, p_n$ propositional variables. Note that the formula $[\neg a]\neg\psi_1^n$ inside the diamonds simulates the window operator and in this way closes the cycle. Thus, we have to invest some work to obtain tree-shaped representations of (possibly non-tree) models that can then be accepted by Büchi-automata.

As a preliminary, we assume that all APDL$^{(\neg)}$-formulas are in *negation normal form (NNF)*, i.e. that negation occurs only in front of propositional letters. This assumption can be made w.l.o.g. since each formula can be converted into an equivalent one in NNF by exhaustively eliminating double negation, applying DeMorgan's rules, and exploiting the duality between diamonds and boxes.

For the sake of brevity, we introduce the following notational conventions:

– for each $APDL^{(\neg)}$-formula φ, $\dot{\neg}\varphi$ denotes the NNF of $\neg\varphi$;

– for each program literal π, $\overline{\pi}$ denotes $\neg\pi$ if π is an atomic program, and a if $\pi = \neg a$ for some atomic program a;

– for each program automaton α, we use Q_α, Σ_α, q_α, Δ_α, and F_α to denote the components of $\alpha = (Q, \Sigma, q_0, \Delta, F)$;

– for each program automaton α and state $q \in Q_\alpha$, we use α_q to denote the automaton $(Q_\alpha, \Sigma_\alpha, q, \Delta_\alpha, F_\alpha)$, i.e. the automaton obtained from α by using q as the new initial state.

Before we can develop the tree-shaped abstraction of models, we need to fix a *closure*, i.e. a set of formulas $\mathsf{cl}(\varphi)$ relevant for deciding the satisfiability of an input formula φ. This is done analogous to [FIS 79, VAR 86]. In the following, when we talk of a subformula ψ of a formula φ, we mean that ψ can be obtained from φ by decomposing only formula operators, but not program operators. For example, a is a subformula of $\langle b?\rangle a$, while b is not.

DEFINITION 6 (CLOSURE). — *Let φ be a APDL$^{(\neg)}$-formula. The set $\mathsf{cl}(\varphi)$ is the smallest set which is closed under the following conditions:*

(C1) $\varphi \in \mathsf{cl}(\varphi)$

(C2) if ψ *is a subformula of* $\psi' \in \mathsf{cl}(\varphi)$*, then* $\psi \in \mathsf{cl}(\varphi)$

(C3) if $\psi \in \mathsf{cl}(\varphi)$*, then* $\dot{\neg}\psi \in \mathsf{cl}(\varphi)$

(C4) if $\langle\alpha\rangle\psi \in \mathsf{cl}(\varphi)$*, then* $\psi' \in \mathsf{cl}(\varphi)$ *for all* $\psi'? \in \Sigma_\alpha$

(C5) if $\langle\alpha\rangle\psi \in \mathsf{cl}(\varphi)$*, then* $\langle\alpha_q\rangle\psi \in \mathsf{cl}(\varphi)$ *for all* $q \in Q_\alpha$

(C6) if $[\alpha]\psi \in \mathsf{cl}(\varphi)$*, then* $\psi' \in \mathsf{cl}(\varphi)$ *for all* $\psi'? \in \Sigma_\alpha$

(C7) if $[\alpha]\psi \in \mathsf{cl}(\varphi)$*, then* $[\alpha_q]\psi \in \mathsf{cl}(\varphi)$ *for all* $q \in Q_\alpha$

It is standard to verify that the cardinality of $\mathsf{cl}(\varphi)$ is polynomial in the length of φ; see e.g. [HAR 00]. We generally assume the diamond formulas (i.e. formulas of the form $\langle\alpha\rangle\psi$) in $\mathsf{cl}(\varphi)$ to be linearly ordered and use ϵ_i to denote the i-th diamond formula in $\mathsf{cl}(\varphi)$, with ϵ_1 being the first one. Note that a changed initial state of an automaton results in a different diamond formula.

To define *Hintikka-trees*, the tree-shaped abstraction of models underlying our decision procedure, we proceed in three steps. First, we introduce *Hintikka-sets* that will be used as (parts of) node labels. Intuitively, each node in the tree describes a world of the corresponding model, and its label contains the formulas from the closure of the input formula φ that are true in this world. Second, we introduce a *matching relation* that describes the possible "neighborhoods" that we may find in Hintikka-trees, where a neighborhood consists of a labeled node and its labeled successors. And third, we use these ingredients to define Hintikka-trees.

DEFINITION 7 (HINTIKKA-SET). — *Let $\psi \in \Phi^{(\neg)}$ be an APDL$^{(\neg)}$-formula, and $\alpha \in A\Pi^{(\neg)}$ a program automaton. The set $\Psi \subseteq \mathsf{cl}(\varphi)$ is a* Hintikka-set for φ *if*

(H1) if $\psi_1 \wedge \psi_2 \in \Psi$, then $\psi_1 \in \Psi$ and $\psi_2 \in \Psi$

(H2) if $\psi_1 \vee \psi_2 \in \Psi$, then $\psi_1 \in \Psi$ or $\psi_2 \in \Psi$

(H3) $\psi \in \Psi$ iff $\dot{\neg}\psi \notin \Psi$

(H4) if $[\alpha]\psi \in \Psi$ and $q_\alpha \in F_\alpha$, then $\psi \in \Psi$

(H5) if $[\alpha]\psi \in \Psi$ then, for any state $q \in Q_\alpha$ and test $\theta? \in \Sigma_\alpha$,
$q \in \Delta_\alpha(q_\alpha, \theta?)$ implies that $\dot{\neg}\theta \in \Psi$ or $[\alpha_q]\psi \in \Psi$

The set of all Hintikka-sets for φ is designated by $\mathcal{H}_\varphi$.

The conditions (H1) to (H3) are standard, with one exception: (H3) is stronger than usual since it enforces maximality of Hintikka-sets by stating that, for each formula $\psi \in \mathsf{cl}(\varphi)$, either ψ or $\dot{\neg}\psi$ must be in the Hintikka-set. This will be used later on to deal with negated programs. The last two conditions (H4) and (H5) deal with the "local" impact of box formulas.

Next, we define the matching relation. The purpose of this relation can be understood as follows: in the Hintikka-tree, each node has exactly one successor for every diamond formula in $\mathsf{cl}(\varphi)$. The matching relation helps to ensure that all diamond formulas in a node's label can be satisfied "via" the corresponding successor in the Hintikka-tree, and that none of the box formulas is violated via any successors. We talk of "via" here since going to an immediate successor corresponds to travelling along a *single* program literal. Since programs in APDL$^{(\neg)}$ are automata that may only accept words of length greater one, in general we cannot satisfy diamonds by going only to the immediate successor, but rather we must perform a sequence of such moves.

Before we define the matching relation formally, let us fix the structure of node labels of Hintikka-trees. For reasons that will be discussed below, node labels not only contain a Hintikka-set, but also two additional components. More precisely, if φ is an APDL$^{(\neg)}$-formula and $\mathsf{cl}(\varphi)$ contains k diamond formulas, then we use

– $\Pi_\varphi^{(\neg)}$ to denote the set of all program literals occurring in φ; and

– Λ_φ to abbreviate $\mathcal{H}_\varphi \times (\Pi_\varphi^{(\neg)} \cup \{\bot\}) \times \{0, \ldots, k\}$, i.e. the set of triples containing a Hintikka-set for φ, a program literal of φ or $\bot$, and a number at most k.

The elements of Λ_φ will be used as node labels in Hintikka-trees. Intuitively, the first component lists the formulas that are true at a node, the second component fixes the program literal with which the node can be reached from its predecessor (or $\bot$ if this information is not important), and the third component will help to ensure that diamond formulas are eventually satisfied when moving through the tree. For a triple $\lambda \in \Lambda_\varphi$, we refer to the first, second and third triple component with λ^1, λ^2, and

λ^3, respectively. For the following definition, recall that we use ϵ_i to denote the i-th diamond in $\mathsf{cl}(\varphi)$.

DEFINITION 8 (MATCHING). — *Let φ be a formula and k the number of diamond formulas in $\mathsf{cl}(\varphi)$. A $k+1$-tuple of Λ_φ-triples $(\lambda, \lambda_1, \ldots, \lambda_k)$ is* matching *if, for $1 \leq i \leq k$ and all automata $\alpha \in A\Pi^{(\neg)}$, the following holds:*

(M1) if $\epsilon_i = \langle\alpha\rangle\psi \in \lambda^1$, then there is a word $w = \psi_1? \cdots \psi_n? \in \Sigma_\alpha^*$, $n \geq 0$,

and a state $q_1 \in Q_\alpha$ such that $\{\psi_1, \ldots, \psi_n\} \subseteq \lambda^1$, $q_1 \in \Delta_\alpha(q_\alpha, w)$,

and one of the following holds:

(a) q_1 is a final state, $\psi \in \lambda^1$, $\lambda_i^2 = \bot$, and $\lambda_i^3 = 0$

(b) there is a program literal $\pi \in \Sigma_\alpha$ and a state $q_2 \in Q_\alpha$ such that

$q_2 \in \Delta_\alpha(q_1, \pi)$, $\epsilon_j = \langle\alpha_{q_2}\rangle\psi \in \lambda_i^1$, $\lambda_i^2 = \pi$, and $\lambda_i^3 = j$.

(M2) if $[\alpha]\psi \in \lambda^1$, $q \in Q_\alpha$, and $\pi \in \Sigma_\alpha$ a program literal such that

$q \in \Delta_\alpha(q_\alpha, \pi)$, then $\pi = \lambda_i^2$ implies $[\alpha_q]\psi \in \lambda_i^1$.

As already noted, the purpose of the matching relation is to describe the possible neighborhoods in Hintikka-trees. To this end, think of λ as the label of a node, and of $\lambda_1, \ldots, \lambda_k$ as the labels of its successors. The purpose of Conditions (M1) and (M2) is to ensure that diamonds are satisfied and that boxes are not violated, respectively. Let us consider only (M1). If a diamond $\epsilon_i = \langle\alpha\rangle\psi$ is in the first component of λ, it can either be satisfied in the node labeled with λ itself (Condition (a)) or we can "delay" its satisfaction to the i-th successor node that is reserved specifically for this purpose (Condition (b)). In Case (a), it is not important over which program literal we can reach the i-th successor, and thus the second component of λ_i can be set to $\bot$. In the second case, we must choose a suitable program literal π and a suitable state q of α, make sure that the i-th successor is reachable over π via its second λ_i-component, and guarantee that the first component of λ_i contains the diamond under consideration with the automata α "advanced" to initial state q.

The remaining building block for ensuring that diamonds are satisfied is to enforce that the satisfaction of diamonds is not delayed forever. This is one of the two core parts of the definition of Hintikka-trees, the other being the proper treatment of negation. Before we can discuss the prevention of infinitely delayed diamonds in some more detail, we have to introduce some basic notions.

Let M be a set and $k \in \mathbb{N}$. An *(infinite) k-ary M-tree* T is a mapping $T : [k]^* \to M$, where $[k]$ is used (now and in the following) as an abbreviation for the set $\{1, \ldots, k\}$. Intuitively, the node αi is the i-th child of α. We use ε to denote the empty word (corresponding to the root of the tree). An infinite *path* in a k-ary M-tree is an infinite word γ over the alphabet $[k]$. We use $\gamma[n]$, $n \geq 0$, to denote the prefix of γ up to the n-th element of the sequence (with $\gamma[0]$ yielding the empty sequence).

Now back to the prevention of infinitely delayed diamonds. Given a formula φ with k diamond formulas in $\mathsf{cl}(\varphi)$, a Hintikka-tree will be defined as a k-ary Λ_φ-tree in which every neighborhood is matching and some additional conditions are satisfied. To detect infinite delays of diamonds in such trees, it does *not* suffice to simply look for infinite sequences of nodes that all contain the same diamond: firstly, diamonds are evolving while being "pushed" through the tree since their initial state might be changed. Secondly, such a sequence does not necessarily correspond to an infinite delay of diamond satisfaction: it could as well be the case that the diamond is satisfied an infinite number of times, but always immediately "regenerated" by some other formula. Also note that we cannot use the standard technique from [VAR 86] since it only works for deterministic variants of PDL.

Precisely for this purpose, the easy detection of infinitely delayed diamonds, we have introduced the third component of node labels in Hintikka-trees: if a diamond was pushed to the current node x from its predecessor, then by (M1) the third component of x's label contains the number of the pushed diamond. Moreover, if the pushed diamond is not satisfied in x, we again use the third component of x: it contains the number of the successor of x to which the diamond's satisfaction is (further) delayed. If no diamond was pushed to x, its third component is simply zero. Thus, the following definition captures our intuitive notion of infinitely delayed diamonds.

DEFINITION 9 (DIAMOND STARVATION). — *Let φ be an APDL$^{(\neg)}$-formula with k diamond formulas in $\mathsf{cl}(\varphi)$, T a k-ary Λ_φ-tree, $x \in [k]^*$ a node in T, and $\epsilon_i = \langle\alpha\rangle\psi \in T(x)^1$. Then the diamond formula $\langle\alpha\rangle\psi$ is called* starving in x *if there exists a path $\gamma = \gamma_1\gamma_2\cdots \in [k]^\omega$ such that*

1) $\gamma_1 = i$,

2) $T(x\gamma[n])^3 = \gamma_{n+1}$ for $n \geq 1$.

We have now gathered all ingredients to define Hintikka-trees formally.

DEFINITION 10 (HINTIKKA-TREE). — *Let φ be an APDL$^{(\neg)}$-formula with k diamond formulas in $\mathsf{cl}(\varphi)$. A k-ary Λ_φ-tree T is a* Hintikka-tree for φ *if T satisfies, for all nodes $x, y \in [k]^*$, the following conditions:*

(T1) $\varphi \in T(\varepsilon)^1$

(T2) the $k+1$-tuple $(T(x), T(x1), \dots, T(xk))$ is matching

(T3) no diamond formula from $\mathsf{cl}(\varphi)$ is starving in x

(T4) if $[\alpha]\psi, [\beta]\theta \in T(x)^1$, $\pi \in \Pi_0^{(\neg)}$, $q'_\alpha \in Q_\alpha$, and $q'_\beta \in Q_\beta$ such that

$q'_\alpha \in \Delta_\alpha(q_\alpha, \pi)$ and $q'_\beta \in \Delta_\beta(q_\beta, \overline{\pi})$, then

$[\alpha_{q'_\alpha}]\psi \notin T(y)^1$ implies $[\beta_{q'_\beta}]\theta \in T(y)^1$.

Conditions (T1) to (T3) are easily understood. The purpose of Condition (T4) is to deal with negated atomic programs. In particular, for each atomic program a we have to ensure that any pair of nodes x, y of a Hintikka-tree T can be related by one of a and $\neg a$ without violating any boxes. This is done by (T4) together with (H3)—indeed, this is the reason for formulating (H3) stronger than usual. Intuitively, the treatment of negation can be understood as follows: suppose that $[\alpha]\psi \in T(x)^1$, let $q \in \Delta_\alpha(q_\alpha, a)$ for some atomic program a, and let y be a node. By (H3), we have either $[\alpha_q]\psi \in T(y)^1$ or $\dot\neg[\alpha_q]\psi \in T(y)^1$. In the first case, x and y can be related by a. In the second case, (T4) ensures that they can be related by $\neg a$. This technique is inspired by [LUT 01], but generalized to program automata. Note that the treatment of negated atomic programs in (T4) allows the representation of non-tree models as (Hintikka-)trees.

We now show that Hintikka-trees are indeed proper abstractions of models. In order to do this, we need two new notions. First, we use Σ_φ to denote the smallest set containing (i) all program literals occurring in φ (i.e. $\Pi_\varphi^{(\neg)}$) and (ii) all tests $\psi?$ occurring in φ. Second, let $\mathcal{M}$ be a Kripke structure and u a world of $\mathcal{M}$. Then a word $w = w_1 \cdots w_m \in \Sigma^*$ is said to *accomplish* a diamond formula $\langle\alpha\rangle\varphi$ at u if $w \in \mathcal{L}(\alpha)$, and there are worlds $u_0, \ldots, u_m$ of $\mathcal{M}$ such that $u = u_0 R(w_1) u_1 R(w_2) \cdots u_{m-1} R(w_m) u_m$ and $\mathcal{M}, u_m \models \varphi$.

PROPOSITION 11. — *An APDL$^{(\neg)}$-formula φ is satisfiable iff it has a Hintikka-tree.*

PROOF 12. — Let φ be an $APDL^{(\neg)}$-formula and k the number of diamond formulas in $\mathsf{cl}(\varphi)$.

“$\Rightarrow$”. Suppose the Kripke structure $\mathcal{M} = (W, R, V)$ is a model of φ, i.e., there is a world $u_\varphi \in W$ such that $\mathcal{M}, u_\varphi \models \varphi$. Fix a linear order $\prec$ on Σ_φ^* such that

– $w \prec w'$ if $|w| < |w'|$, and
– $ww' \prec ww''$ if $w' \prec w''$.

Then define a partial function $\ell : \mathsf{cl}(\varphi) \times W \to \mathbb{N}$ as follows: for each $\langle\alpha\rangle\psi \in \mathsf{cl}(\varphi)$, and $u \in W$ such that $\mathcal{M}, u \models \langle\alpha\rangle\psi$, $\ell(\langle\alpha\rangle\psi, u)$ denotes the length of the word $w \in \Sigma_\varphi^*$ that accomplishes $\langle\alpha\rangle\psi$ at u and is minimal (w.r.t. $\prec$) with this property.

In the following, we construct a Hintikka-tree for φ. More precisely, we simultaneously define

– a k-ary $(2^{\mathsf{cl}(\varphi)} \times (\Pi_\varphi^{(\neg)} \cup \{\bot\}) \times \{0, \ldots, k\})$-tree T_φ and
– a mapping $\tau : [k]^* \to W$

such that, for all $x \in [k]^*$, we have

$$\psi \in T_\varphi(x)^1 \text{ iff } \mathcal{M}, \tau(x) \models \psi. \qquad (*)$$

The construction of τ and T_φ is inductive. For the induction base, set

$$\begin{array}{rcl}
\tau(\varepsilon) & := & u_\varphi, \\
T_\varphi(\varepsilon)^1 & := & \{\psi \in \mathsf{cl}(\varphi) \mid \mathcal{M}, u_\varphi \models \psi\}, \\
T_\varphi(\varepsilon)^2 & := & \bot, \text{ and} \\
T_\varphi(\varepsilon)^3 & := & 0.
\end{array}$$

For the induction step, let $x \in [k]^*$ be a node such that $\tau(x)$ is already defined, but $\tau(x1), \ldots, \tau(xk)$ are not. For each $i \in [k]$, we distinguish two cases:

1) $\epsilon_i = \langle\alpha\rangle\psi \in T_\varphi(x)^1$. Then ($*$) yields $\mathcal{M}, \tau(x) \models \langle\alpha\rangle\psi$. By the semantics, there thus exists a word that accomplishes $\langle\alpha\rangle\psi$ at $\tau(x)$. Let $w = w_1 \cdots w_m \in \Sigma_\varphi^*$ be the minimal such word w.r.t. the ordering $\prec$. Then there are worlds $u_0, \ldots, u_m \in W$ such that $\tau(x) = u_0 R(w_1) u_1 R(w_2) u_2 \cdots u_{m-1} R(w_m) u_m$ and $\mathcal{M}, u_m \models \psi$. Distinguish two subcases:

(a) The word w contains only tests or is empty. This implies that $\tau(x) = u_m$, and thus $\mathcal{M}, \tau(x) \models \psi$. By ($*$), we get $\psi \in T_\varphi(x)^1$. Set

$$\begin{array}{rcl}
\tau(xi) & := & u_\varphi, \\
T_\varphi(xi)^1 & := & \{\psi \in \mathsf{cl}(\varphi) \mid \mathcal{M}, u_\varphi \models \psi\}, \\
T_\varphi(xi)^2 & := & \bot, \text{ and} \\
T_\varphi(xi)^3 & := & 0.
\end{array}$$

(b) Otherwise, fix an run $q_0 \cdots q_m \in Q_\alpha^*$, of α on w. Take the least $p \in [m]$ such that w_p is not a test but a program literal. Let $j \in [k]$ be such that $\epsilon_j = \langle\alpha_{q_p}\rangle\psi$. Set

$$\begin{array}{rcl}
\tau(xi) & := & u_p, \\
T_\varphi(xi)^1 & := & \{\psi \in \mathsf{cl}(\varphi) \mid \mathcal{M}, u_p \models \psi\}, \\
T_\varphi(xi)^2 & := & w_p, \text{ and} \\
T_\varphi(xi)^3 & := & j.
\end{array}$$

2) $\epsilon_i = \langle\alpha\rangle\psi \notin T_\varphi(x)^1$. Then set

$$\begin{array}{rcl}
\tau(xi) & := & u_\varphi, \\
T_\varphi(xi)^1 & := & \{\psi \in \mathsf{cl}(\varphi) \mid \mathcal{M}, u_\varphi \models \psi\}, \\
T_\varphi(xi)^2 & := & \bot, \text{ and} \\
T_\varphi(xi)^3 & := & 0.
\end{array}$$

To show that T_φ is indeed a Hintikka-tree for φ, we first prove the following claim which will be helpful in showing that our Hintikka-tree does not contain starving diamonds.

Claim. Let $x, y \in [k]^*$ be nodes such that $xi = y$ for some $i \in [k]$. If $T_\varphi(y)^3 = j \neq 0$, then (i) $\epsilon_i \in T_\varphi(x)^1$, (ii) $\epsilon_j \in T_\varphi(y)^1$, and (iii) $\ell(\epsilon_j, \tau(y)) < \ell(\epsilon_i, \tau(x))$.

Proof. Let x and y be as in the claim. Suppose $T_\varphi(y)^3 = j \neq 0$. Then $\tau(y)$ and $T_\varphi(y)$ have been set in Case 1b of the induction step. Hence $\epsilon_i \in T_\varphi(x)^1$ and thus Point (i) is proved. Let $w = w_1 \cdots w_m$, $u_1 \cdots u_m$, $q_0 \cdots q_m$, and p be as in Case 1b. Clearly, $w \in \mathcal{L}(\alpha)$ implies $w_{p+1} \cdots w_m \in \mathcal{L}(\alpha_{q_p})$. Since

$$\tau(y) = u_p R(w_{p+1}) u_{p+1} R(w_{p+2}) \cdots u_{m-1} R(w_m) u_m \qquad (\dagger)$$

and $\mathcal{M}, u_m \models \psi$, we have $\mathcal{M}, \tau(y) \models \langle \alpha_{q_p} \rangle \psi = \epsilon_j$, which shows Point (ii). As shown by $w_{p+1} \cdots w_m \in \mathcal{L}(\alpha_{q_p})$, (†), and $\mathcal{M}, u_m \models \psi$, $w_{p+1} \cdots w_m$ accomplishes ϵ_j at $\tau(y)$. Clearly $w_{p+1} \cdots w_m$ is minimal with this property: under the assumption of non-minimality, it is easy to derive a contradiction to the minimality of the word w for accomplishing ϵ_i at x. Hence, $\ell(\epsilon_j, \tau(y)) = |w_{p+1} \cdots w_m| < m = |w| = \ell(\epsilon_i, \tau(x))$ finishing the proof of the claim.

We now prove that T_φ is a Hintikka-tree for φ. First, it is shown that, for each node $x \in [k]^*$, $T_\varphi(x)^1$ is a Hintikka-set. Observe that Conditions (H1), (H2), and (H3) easily follow from the semantics and the definition of the closure cl. Now consider the Condition (H4). Suppose that $[\alpha]\psi \in T_\varphi(x)^1$, and $q_\alpha \in F_\alpha$. We get $\mathcal{M}, x \models [\alpha]\psi$ (by $(*)$), $(\tau(x), \tau(x)) \in R(\alpha)$, and thus also $\mathcal{M}, \tau(x) \models \psi$. Hence, $\psi \in T_\varphi(x)^1$ by $(*)$. Finally, look at the remaining Condition (H5). Suppose that $[\alpha]\psi \in T_\varphi(x)^1$, and let $q \in Q_\alpha$ be a state, and $\theta? \in \Sigma_\alpha$ a test such that $q \in \Delta_\alpha(q_\alpha, \theta?)$. By (H3), $\dot\neg\theta \in T_\varphi(x)^1$ or $\theta \in T_\varphi(x)^1$. In the former case, (H4) follows immediately. Consider the latter case. By $(*)$, it holds that $\mathcal{M}, \tau(x) \models [\alpha]\psi$, and $\mathcal{M}, \tau(x) \models \theta$. From the semantics it follows that $\mathcal{M}, \tau(x) \models [\alpha_q]\psi$. Thus, $[\alpha_q]\psi \in T_\varphi(x)^1$ by $(*)$.

Since the first triple components of the node labels in tree T_φ are Hintikka-sets, T_φ is a k-ary Λ_φ-tree. It remains to show that T_φ additionally satisfies the conditions for Hintikka-trees (T1) to (T4). Let $x, y \in [k]^*$ be nodes.

(T1) Holds by definition of T_φ; see induction start.

(T2) It is to show that the $k+1$-tuple $(T_\varphi(x), T_\varphi(x1), \ldots, T_\varphi(xk))$ is matching. We first consider the matching condition (M1). Suppose $\epsilon_i = \langle \alpha \rangle \psi \in T_\varphi(x)^1$. By $(*)$, $\mathcal{M}, \tau(x) \models \langle \alpha \rangle \psi$. Thus there exists a word that accomplishes ϵ_i at $\tau(x)$. Let $w = w_1 \cdots w_m$ be the minimal such word w.r.t. $\prec$. If w comprises only tests, then $\mathcal{M}, \tau(x) \models \psi$ and $\tau(xi)$ and $T_\varphi(xi)$ are defined in Case 1a of the induction step. It is thus readily checked that Case (a) of (M1) is satisfied: since $w \in \mathcal{L}(\alpha)$, we find a final state q_1 as required. Since $\mathcal{M}, \tau(x) \models \psi$ and we are in Case 1a, we obtain $\psi \in T_\varphi(x)^1$, $T_\varphi(xi)^2 = \bot$, and $T_\varphi(xi)^3 = 0$. Now assume that w does contain a program literal and let $p \in [m]$ be minimal such that w_p is a program literal. Then $\tau(xi)$ and $T_\varphi(xi)$ are defined in Case 1b of the induction step. As in the proof of the claim, it follows that $\epsilon_j = \langle \alpha_q \rangle \psi \in T_\varphi(xi)^1$, where $q \in \Delta_\alpha(q_\alpha, w_1 \cdots w_p)$. Moreover, we have $T_\varphi(xi)^2 = w_p$ and $T_\varphi(xi)^3 = j$. Thus, Case (b) of (M1) is fulfilled.
Consider the remaining Condition (M2). Suppose $[\alpha]\psi \in T_\varphi(x)^1$. By $(*)$, it holds that $\mathcal{M}, \tau(x) \models [\alpha]\psi$. Let $q \in Q_\alpha$ be a state of α, and $\pi \in \Sigma_\alpha$ a program literal such that $q \in \Delta_\alpha(q_\alpha, \pi)$. Assume $\pi = T_\varphi(xi)^2$ for some $i \in [k]$. Then $\tau(xi)$ and $T_\varphi(xi)$ have been set in Case (1b) of the induction step. Thus $(\tau(x), \tau(xi)) \in R(\pi)$. Together with the fact that $\mathcal{M}, \tau(x) \models [\alpha]\psi$, it follows that $\mathcal{M}, \tau(xi) \models [\alpha_q]\psi$. Consequently, $[\alpha_q]\psi \in T_\varphi(xi)^1$ by $(*)$.

(T3) Suppose by contradiction that the diamond formula $\epsilon_i = \langle \alpha \rangle \psi \in T_\varphi(x)^1$ is starving in node x, i.e., there is a path $\gamma = \gamma_1\gamma_2 \cdots \in [k]^\omega$ such that $\gamma_1 = i$, and $T_\varphi(x\gamma[n])^3 = \gamma_{n+1}$ for $n \geq 1$. Label x with the natural number $\ell(\epsilon_{\gamma_1}, \tau(x))$, and the nodes $x\gamma[n]$, $n \geq 1$, with $\ell(\epsilon_{T_\varphi(x\gamma[n])^3}, \tau(x\gamma[n]))$. By the above claim, these numbers are strictly decreasing along the path γ: a contradiction to the fact that the range of the function ℓ is $\mathbb{N}$.

(T4) Suppose that $[\alpha]\psi, [\beta]\theta \in T_\varphi(x)^1$. Let $\pi \in \Pi_\varphi^{(\neg)}$ be a program literal, and $q'_\alpha \in Q_\alpha$, $q'_\beta \in Q_\beta$ be states such that $q'_\alpha \in \Delta_\alpha(q_\alpha, \pi)$, and $q'_\beta \in \Delta_\beta(q_\beta, \overline{\pi})$. Further suppose that $[\alpha_{q'_\alpha}]\psi \notin T(y)^1$. We show that $(\tau(x), \tau(y)) \in R(\overline{\pi})$. Assume by contradiction that $(\tau(x), \tau(y)) \in R(\pi)$. Since $\mathcal{M}, \tau(x) \models [\alpha]\psi$ by ($*$), the semantics yields, $\mathcal{M}, \tau(y) \models [\alpha_{q'_\alpha}]\psi$. Then ($*$) yields $[\alpha_{q'_\alpha}]\psi \in T(y)^1$, a contradiction. Consequently, $(\tau(x), \tau(y)) \in R(\overline{\pi})$. Since $\mathcal{M}, \tau(x) \models [\beta]\theta$ by ($*$), the semantics yields, $\mathcal{M}, \tau(y) \models [\beta_{q'_\beta}]\theta$. Then, again by ($*$), we have $[\beta_{q'_\beta}]\theta \in T(y)^1$ as required by (T4).

"$\Leftarrow$". Suppose T is a Hintikka-tree for φ. Construct a Kripke Structure $\mathcal{M} = (W, R, V)$ in the following way:

– $W := [k]^*$;

– For each atomic program a, set

$$\begin{array}{rcl} R_\Diamond(a) & := & \{(x,y) \in W^2 \mid y = xi \text{ for some } i \in [k], \text{ and } T(y)^2 = a\}, \text{ and} \\ R_\Box(a) & := & \{(x,y) \in W^2 \mid \text{there is a } [\alpha]\psi \in T(x)^1 \text{ and a } q \in \Delta_\alpha(q_\alpha, \neg a) \\ & & \qquad \text{such that } [\alpha_q]\psi \notin T(y)^1\} \end{array}$$

Now we can define the accessibility relation for a: $R(a) := R_\Diamond(a) \cup R_\Box(a)$;

– The valuation function $V(p)$ for propositional variables p is defined as

$$V(p) := \{x \in W \mid p \in T(x)^1\}.$$

To show that $\mathcal{M}$ is a model of φ, we prove two claims. The first is concerned with the relational structure:

Claim 1. For each program literal $\pi \in \Pi_\varphi^{(\neg)}$, node $x \in [k]^*$, and $i \in [k]$, we have that $T(xi)^2 = \pi$ implies $(x, xi) \in R(\pi)$.

Proof: If π is an atomic program, then the claim is an immediate consequence of the definition of $R(a)$. Thus suppose that $\pi = \neg a$ is a negated atomic program. Assume that $T(x)^2 = \neg a$ and $(x, xi) \notin R(\neg a)$. By the semantics, we obtain $(x, xi) \in R(a)$. Since $T(x)^2 = \neg a$, in particular we have $(x, xi) \in R_\Box(a)$. Thus there is a $[\alpha]\psi \in T(x)^1$ and a $q \in \Delta_\alpha(q_\alpha, \neg a)$ such that $[\alpha_q]\psi \notin T(y)^1$. Since $T(x)^2 = \neg a$, however, this is in contradiction to (M2).

The second claim is concerned with the truth of formulas.

Claim 2. For each $\psi \in \mathsf{cl}(\varphi)$ and $x \in [k]^*$, $\psi \in T(x)^1$ implies $\mathcal{M}, x \models \psi$.

Proof: Let ψ and x be as in the claim. Inductively define the *norm* $||\cdot||$ of $APDL^{(\neg)}$-formulas in NNF as follows:

$$\begin{array}{lclcl} ||p|| & := & ||\neg p|| & := & 0 \text{ for } p \in \Phi_0 \\ ||\psi_1 \wedge \psi_2|| & := & ||\psi_1 \vee \psi_2|| & := & 1 + ||\psi_1|| + ||\psi_2|| \\ ||\langle\alpha\rangle\psi|| & := & ||[\alpha]\psi|| & := & 1 + ||\psi|| + \sum\limits_{\theta? \in \Sigma_\alpha} ||\theta|| \end{array}$$

The proof is by induction on the norm $||\cdot||$. The induction base has two cases:

– ψ is a propositional variable. Immediate by definition of $\mathcal{M}$.

– $\psi = \neg p$. We have $p \in \Phi_0$ due to NNF. With (H3), $\psi \in T(x)^1$ thus implies that $p \notin T(x)^1$. By definition of $\mathcal{M}$, $x \notin V(p)$. Hence, $\mathcal{M}, x \models \neg p$.

The induction step consists of several cases:

– $\psi = \psi_1 \vee \psi_2$ or $\psi = \psi_1 \wedge \psi_2$. Since $T(x)^1$ is a Hintikka-set, these cases are straightforward by Conditions (H1) and (H2) and the induction hypothesis.

– $\psi = \epsilon_i = \langle\alpha\rangle\theta$. Inductively define a (potentially infinite) path $\gamma = \gamma_1\gamma_2\cdots \in ([k]^\omega \cup [k]^*)$ as follows:

- $\gamma_1 := i$
- if $T(x\gamma[n])^3 \neq 0$, then $\gamma_{n+1} := T(x\gamma[n])^3$;

otherwise, γ_n is the last node in the path.

By (T3), the diamond formula $\langle\alpha\rangle\theta$ is not starving in x. It is thus readily checked that the path γ is finite, i.e. $\gamma = \gamma_1\cdots\gamma_m$ and $T(x\gamma)^3 = 0$. By (M1), there thus exist two sequences of states $q_0, \ldots, q_m \in Q_\alpha$ and $q'_0, \ldots, q'_m \in Q_\alpha$, a sequence of program literals $\pi_1, \ldots, \pi_m \in \Sigma_\alpha$, and a sequence of sequences of tests $t_0, \ldots, t_m \in \Sigma^*_\alpha$ such that, for $n \leq m$, we have

1) $q_0 = q_\alpha$;

2) if $t_n = \psi_1?\cdots\psi_\ell?$, then $\psi_1?, \ldots, \psi_\ell? \in T(x\gamma[n])^1$;

3) $q'_n \in \Delta_\alpha(q_n, t_n)$;

4) $T(x\gamma[n-1])^2 = \pi_n$ if $n > 0$;

5) $q_{n+1} \in \Delta_\alpha(q'_n, \pi_{n+1})$ if $n < m$;

6) q'_m is a final state;

7) $\theta \in T(x\gamma)^1$.

By induction hypothesis, we obtain the following from Point 2 and 7: for $n \leq m$, we have

8) if $t_n = \psi_1?\cdots\psi_\ell?$, then $\mathcal{M}, x\gamma[n] \models \psi_j?$ for $1 \leq j \leq \ell$;

9) $\mathcal{M}, x\gamma \models \theta$.

Together with Claim 1, we obtain from Point 4:

10) for $n \in [m]$, we have $(x\gamma[n-1], x\gamma[n]) \in R(\pi)$.

Using Points 1,8,3,10,5, and 6, it is readily checked that $(x, x\gamma) \in R(\alpha)$. Together with Point 9, we thus obtain $\mathcal{M}, x \models \langle\alpha\rangle\theta$ as required.

– $\psi = [\alpha]\theta$. By the semantics, it needs to be shown that, for any world $y \in W$, $(x, y) \in R(\alpha)$ implies $\mathcal{M}, y \models \theta$. Thus suppose $(x, y) \in R(\alpha)$ for some $y \in W$, i.e., there is a word $w = w_1\cdots w_m \in \mathcal{L}(\alpha)$, $m \geq 0$, and worlds $x_0, \ldots, x_m \in W$ with $x = x_0 R(w_1) x_1 R(w_2) \cdots x_{m-1} R(w_m) x_m = y$. Fix an accepting run $q_0 \cdots q_m \in Q^*_\alpha$ of the program automaton α on w. In the following, it is shown that $[\alpha_{q_i}]\theta \in$

$T(x_i)^1$ for $i \leq m$. We proceed by induction on i. The induction start is immediate. For the step, assume that $[\alpha_{q_i}]\theta \in T(x_i)^1$ has already been shown. We distinguish the following cases:

(i) $w_i = \delta?$ is a test. From $(x_i, x_{i+1}) \in R(\delta?)$ it follows by the semantics that $x_i = x_{i+1}$, and $\mathcal{M}, x_i \models \delta$. Suppose by contradiction that $\dot{\neg}\delta \in T(x_i)^1$. The induction hypothesis yields $\mathcal{M}, x_i \models \dot{\neg}\delta$, a contradiction to $\mathcal{M}, x_i \models \delta$. Thus $\dot{\neg}\delta \notin T(x_i)^1$, and by (H5) we obtain $[\alpha_{q_{i+1}}]\theta \in T(x_i)^1 = T(x_{i+1})^1$.

(ii) w_i is an atomic program a in $\Pi_\varphi^{(\neg)}$. By definition of $R(a)$, $(x_i, x_{i+1}) \in R_\Diamond(a) \cup R_\Box(a)$. Firstly, suppose that $(x_i, x_{i+1}) \in R_\Diamond(a)$. The definition of $R_\Diamond(a)$ yields $x_{i+1} = x_i j$ for some $j \in [k]$, and $T(x_{i+1})^2 = a$. Consequently, by Condition (M2), $[\alpha_{q_{i+1}}]\theta \in T(x_{i+1})^1$.
Secondly, suppose that $(x_i, x_{i+1}) \in R_\Box(a)$. By definition of $R_\Box(a)$, there is a $[\beta]\delta \in T(x_i)^1$, and $[\beta_q]\delta \notin T(x_{i+1})^1$ for some $q \in \Delta_\beta(q_\beta, \overline{a})$. Since $[\alpha_{q_i}]\theta \in T(x_i)^1$ by assumption, the Condition (T4) yields $[\alpha_{q_{i+1}}]\theta \in T(x_{i+1})^1$.

(iii) w_i is a negated atomic program $\neg a$ in $\Pi_\varphi^{(\neg)}$. By the semantics, we have $R(\neg a) = W^2 \setminus R(a)$. By definition of $R(a)$, this yields $(x_i, x_{i+1}) \in W^2 \setminus (R_\Diamond(a) \cup R_\Box(a))$. The fact that $(x_i, x_{i+1}) \notin R_\Box(a)$ implies that, for any box formula $[\beta]\delta$ and state $q \in Q_\beta$ with $q \in \Delta_\beta(q_\beta, \neg a)$, $[\beta]\delta \in T(x_i)^1$ implies $[\beta_q]\delta \in T(x_{i+1})^1$. Consequently, $[\alpha_{q_{i+1}}]\theta \in T(x_{i+1})^1$.

It thus holds that $[\alpha_{q_m}]\theta \in T(x)^1$. Since $q_m \in F_\alpha$, by Condition (H4) it follows that $\theta \in T(x)^1$ as required.

This finishes the proof of Claim 2. By Condition (T1), we have $\varphi \in T(\varepsilon)^1$. Thus, it follows directly from Claim 2 that $\mathcal{M}$ is a model of φ. ■

5. Büchi Automata for Hintikka-trees

In this section, we show that it is possible to construct, for every APDL$^{(\neg)}$-formula φ, a Büchi tree automaton $\mathcal{B}_\varphi$ that accepts exactly the Hintikka-trees for φ. By Proposition 11, since the size of $\mathcal{B}_\varphi$ is at most exponential in the length of φ, and since the emptiness of Büchi-tree automata can be verified in quadratic time [VAR 86], this yields an EXPTIME decision procedure for the satisfiability of APDL$^{(\neg)}$-formulas. We start with introducing Büchi tree automata.

DEFINITION 13 (BÜCHI TREE AUTOMATON). — *A* Büchi tree automaton $\mathcal{B}$ for k-ary M-trees *is a quintuple* (Q, M, I, Δ, F), *where*

– Q *is a finite set of* states,

– M *is a finite* alphabet,

– $I \subseteq Q$ *is the set of* initial states,

– $\Delta \subseteq Q \times M \times Q^k$ *is the* transition relation, *and*

– $F \subseteq Q$ *is the set of* accepting states.

Let M be a set of labels, and T a k-ary M-tree. Then, a run *of $\mathcal{B}$ on T is a k-ary Q-tree r such that*

1) $r(\varepsilon) \in I$, and

2) $(r(x), T(x), r(x1), \ldots, r(xk)) \in \Delta$ for all nodes $x \in [k]^*$.

Let $\gamma \in [k]^\omega$ be a path. The set $inf_r(\gamma)$ contains the states in Q that occur infinitely often in run r along path γ. A run r of $\mathcal{B}$ on T is accepting *if, for each path $\gamma \in [k]^\omega$, we have $inf_r(\gamma) \cap F \neq \emptyset$. The* language *accepted by $\mathcal{B}$ is the set $\mathcal{L}(\mathcal{B}) = \{T \mid$ there is an accepting run of $\mathcal{B}$ on $T\}$.*

Given a Büchi automaton $\mathcal{B}$, the problem whether its language is empty, i.e., whether it holds that $\mathcal{L}(\mathcal{B}) = \emptyset$, is called the *emptiness problem*. This problem is solvable in time quadratic in the size of the automaton [VAR 86].

We now give the translation of APDL$^{(\neg)}$-formulas φ into Büchi-automata $\mathcal{B}_\varphi$. To simplify the notation, we write

$$\mathcal{P}_\square(\varphi) \text{ for } \{\{[\alpha]\psi, [\beta]\theta\} \mid [\alpha]\psi, [\beta]\theta \in \mathsf{cl}(\varphi)\}.$$

We first introduce our automata formally and then explain the intuition.

DEFINITION 14. — *Let φ be an APDL$^{(\neg)}$-formula with $\mathsf{cl}(\varphi)$ containing k diamond formulas. The Büchi tree automaton $\mathcal{B}_\varphi = (Q, \Lambda_\varphi, I, \Delta, F)$ on k-ary Λ_φ-trees is defined as follows:*

– Q contains those triples

$$((\Psi, \pi, \ell), P, d) \in \Lambda_\varphi \times 2^{\mathcal{P}_\square(\varphi)} \times \{\oslash, \uparrow\}$$

that satisfy the following conditions:

(1) if $\{[\alpha]\psi, [\beta]\theta\} \subseteq \Psi$, then $\{[\alpha]\psi, [\beta]\theta\} \in P$

(2) if $\{[\alpha]\psi, [\beta]\theta\} \in P$, $\pi \in \Pi^{(\neg)}$, $q'_\alpha \in \Delta_\alpha(q_\alpha, \pi)$, $q'_\beta \in \Delta_\beta(q_\beta, \overline{\pi})$, and $[\alpha_{q'_\alpha}]\psi \notin \Psi$, then $[\beta_{q'_\beta}]\theta \in \Psi$

– $I := \{((\Psi, \pi, \ell), P, d) \in Q \mid \varphi \in \Psi$, and $d = \oslash\}$.

– $((\lambda_0, P_0, d_0), (\Psi, \pi, \ell), (\lambda_1, P_1, d_1), \ldots, (\lambda_k, P_k, d_k)) \in \Delta$ if and only if, for each $i \in [k]$, the following holds:

1) $\lambda_0 = (\Psi, \pi, \ell)$,

2) $P_0 = P_i$,

3) the tuple $(\lambda_0, \ldots, \lambda_k)$ is matching,

4) $d_i = \begin{cases} \uparrow & \text{if } d_0 = \oslash,\ \lambda_i^3 \neq 0 \text{ and } \epsilon_i \in \Psi \\ \uparrow & \text{if } d_0 = \uparrow,\ \lambda_0^3 = i, \text{ and } \lambda_i^3 \neq 0 \\ \oslash & \text{otherwise}. \end{cases}$

– *The set F of accepting states is* $F := \{(\lambda, P, d) \in Q \mid d = \oslash\}$.

While it is not hard to see how the set of initial states enforces (T1) of Hintikka-trees and how the transition relation enforces (T2), Conditions (T3) and (T4) are more challenging. In the following, we discuss them in detail.

Condition (T3) is enforced with the help of the third component of states, which may take the values "$\oslash$" and "$\uparrow$". Intuitively, the fourth point in the definition of Δ ensures that, whenever the satisfaction of a diamond is delayed in a node x and r is a run, then r assigns states with third component $\uparrow$ to all nodes on the path that "tracks" the diamond delay. Note that, for this purpose, the definition of Δ refers to the third component of Λ_φ-tuples, which is "controlled" by (M1) in the appropriate way. All nodes that do not appear on delayed diamond paths are labeled with $\oslash$. Then, the set of accepting states ensures that there is no path that, from some point on, is constantly labeled with $\uparrow$. Thus, we enforce that no diamonds are delayed infinitely in trees accepted by our automata, i.e. no starvation occurs.

There is one special case that should be mentioned. Assume that a node x contains a diamond $\epsilon_i = \langle\alpha\rangle\psi$ that is not satisfied "within this node" (Case (a) of (M1) does not apply). Then there is a potential starvation path for ϵ_i that starts at x and goes through the node xi: (M1) "advances" the automaton α to α_q, and ensures that $\epsilon_j = \langle\alpha_q\rangle\psi \in T(xi)^1$ and that $T(xi)^3 = j$. Now suppose that $T(xi)^1$ contains another diamond $\epsilon_k = \langle\beta\rangle\theta$ with $\epsilon_j \neq \epsilon_k$. If ϵ_k is not satisfied within xi, there is a potential starvation path for ϵ_k starting at xi and going through xik. Since the starvation path for ϵ_i and the starvation path for ϵ_k are for different diamonds, we must be careful to separate them—failure in doing this would result in some starvation-free Hintikka-trees to be rejected. Thus, the definition of Δ ensures that runs label xik with $\oslash$, and the constant $\uparrow$-labeling of the starvation path for ϵ_k is delayed by one node: it starts only at the *successor* of xik on the starvation path for ϵ_k.

Now for Condition (T4). In contrast to Conditions (T1) and (T2), this condition has a global flavor in the sense that it does not only concern a node and its successors. Thus, we need to employ a special technique to enforce that (T4) is satisfied: we use the second component of states as a "bookkeeping component" that allows to propagate global information. More precisely, Point (1) of the definition of Q and Point (1) of the definition of Δ ensure that, whenever two boxes appear in a Hintikka-set labeling a node x in a Hintikka-tree T, then this joint occurrence is recorded in the second component of the state that any run assigns to x. Via the definition of the transition relation (second point), we further ensure that all states appearing in a run share the same second component. Thus, we may use Point (2) of the definition of Q and Point (1) of the definition of Δ to ensure that any node y satisfies the property stated by Condition (T4).

The following proposition shows that the Büchi tree automaton $\mathcal{B}_\varphi$ indeed accepts precisely the Hintikka-trees for APDL$^{(\neg)}$-formula φ.

PROPOSITION 15. — *Let φ be an APDL$^{(\neg)}$-formula and T a k-ary Λ_φ-tree. Then T is a Hintikka-tree for φ iff $T \in \mathcal{L}(\mathcal{B}_\varphi)$.*

PROOF 16. — Let φ be a APDL$^{(\neg)}$-formula and k the number of diamond formulas in $\mathsf{cl}(\varphi)$.

"$\Rightarrow$". Suppose T is a Hintikka-tree for φ. In the following, we show that $T \in \mathcal{L}(\mathcal{B}_\varphi)$, i.e. that there is an accepting run of the Büchi automaton $\mathcal{B}_\varphi$ on T. Set

$$P_\Box(T) := \{\{[\alpha]\psi, [\beta]\theta\} \mid \text{ there is an } x \in [k]^* \text{ such that } [\alpha]\psi, [\beta]\theta \in T(x)^1\}.$$

Inductively define a $\Lambda_\varphi \times \{P_\Box(T)\} \times \{\oslash, \uparrow\}$-tree r as follows:

– set $r(\varepsilon) := (T(\varepsilon), P_\Box(T), \oslash)$;

– let $x \in [k]^*$ such that $r(x) = (T(x), P_\Box(T), d_x)$ is already defined, and let $i \in [k]$. Set $r(xi) := (T(xi), P_\Box(T), d_{xi})$ where

$$d_{xi} := \begin{cases} \uparrow & \text{if } d_x = \oslash, T(xi)^3 \neq 0, \text{ and } \epsilon_i \in T(x)^1 \\ \uparrow & \text{if } d_x = \uparrow, T(x)^3 = i, \text{ and } T(xi)^3 \neq 0 \\ \oslash & \text{otherwise} \end{cases} .$$

We claim that the tree r is an accepting run of $\mathcal{B}_\varphi$ on T. We start with showing that r is a Q-tree. To this end, let $x \in [k]^*$ be a node. We have to show that $r(x) \in Q$. Since T is a Hintikka-tree for φ, we have $T(x) \in \Lambda_\varphi$. Moreover $P_\Box(T) \subseteq \mathcal{P}_\Box(\varphi)$, and thus $r(x) = (T(x), P_\Box(T), d_x) \in \Lambda_\varphi \times 2^{\mathcal{P}_\Box(\varphi)} \times \{\oslash, \uparrow\}$. We still have to show that $r(x)$ satisfies Properties (1) and (2) of the definition of Q.

(1) Holds by definition of $P_\Box(T)$ and since $r(x)^1 = T(x)$;

(2) Suppose that $\{[\alpha]\psi, [\beta]\theta\} \in P_\Box(T)$, and $[\alpha_{q'_\alpha}]\psi \notin T(x)^1$ where $q'_\alpha \in \Delta_\alpha(q_\alpha, \pi)$ for some program literal $\pi \in \Pi^{(\neg)}$. Let $q'_\beta \in \Delta_\beta(q_\beta, \overline{\pi})$. By definition of $P_\Box(T)$, there is a node $y \in [k]^*$ with $\{[\alpha]\psi, [\beta]\theta\} \subseteq T(y)^1$. Then, the Condition (T4) yields $[\beta_{q'_\beta}]\theta \in T(x)^1$.

Thus, r is a Q-tree. To show that r is a run of $\mathcal{B}_\varphi$ on T, it remains to verify the two conditions from Definition 13.

– $r(\varepsilon) \in I$. Since $\varphi \in T(\varepsilon)^1$ by Condition (T1), $r(\varepsilon) = (T(\varepsilon), P_\Box(T), \oslash) \in I$.

– $(r(x), T(x), r(x1), \ldots, r(xk)) \in \Delta$ for all nodes $x \in [k]^*$. According to the definition of Δ, there are four conditions that need to be checked. Conditions 1 and 2 are trivial by definition of r. By (T2), we have that the tuple $(T(x), T(x1), \ldots, T(xk))$ is matching. Thus, Condition 3 is satisfied. Finally, using the definition of r it is easily checked that Condition 4 is also satisfied.

It remains to show that the run r is accepting. Suppose that it is not, i.e. that there is a path $\gamma = \gamma_1\gamma_2 \cdots \in [k]^\omega$ such that $inf_r(\gamma) \cap F = \emptyset$. By definition of F, the set $inf_r(\gamma)$ contains only states q with third component $\uparrow$. Consequently, there is a position p in the sequence γ such that, for all $m \geq p$, the third component of $r(\gamma[m])$ is $\uparrow$. Let p be the minimal such position. By definition of $r(\varepsilon)$, we know that $p > 0$. Minimality of p thus yields that the third component of $r(\gamma[p-1])$ is $\oslash$. Together with the fact that the third component of $r(\gamma[p])$ is $\uparrow$ and by definition of r (the third component), this implies that the diamond formula ϵ_ℓ with $\ell = \gamma_p$ is in $T(\gamma[p-1])^1$. We show that the diamond ϵ_ℓ is starving in $\gamma[p-1]$, in this way obtaining a contradiction to the fact that T is a Hintikka-tree satisfying (T3).

To show that ϵ_ℓ is starving in $\gamma[p-1]$, we show that the path $\gamma_p\gamma_{p+1}\cdots$ satisfies Properties 1 and 2 from Definition 9:

– $\ell = \gamma_p$. Satisfied by choice of ℓ.

– for $m \geq p$, $T(\gamma[m])^3 = \gamma_{m+1}$. Fix an $m \geq p$. Since the third component of $r(\gamma[m])$ and of $r(\gamma[m+1])$ is $\uparrow$, we obtain $T(\gamma[m])^3 = \gamma_{m+1}$ by definition of r (third component).

"$\Leftarrow$". Suppose $T \in \mathcal{L}(\mathcal{B}_\varphi)$, i.e., there is an accepting run r of $\mathcal{B}_\varphi$ on T. We show that T is a Hintikka-tree for φ. Since $\mathcal{B}_\varphi$ is a Büchi-automaton on k-ary Λ_φ-trees, T is a k-ary Λ_φ-tree. Let $x, y \in [k]^*$ be a nodes in T such that $r(x) = (T_x, P_x, d'_x)$ and $r(y) = (T_y, P_y, d'_y)$. In the following, we check that T fulfills the conditions for Hintikka-trees (T1) to (T4).

(T1) Let $r(\varepsilon) = (T(\varepsilon), P, d'_\varepsilon)$. By definition of a run, it holds that $r(\varepsilon) \in I$. Thus, $\varphi \in T(\varepsilon)^1$ by definition of I.

(T2) By definition of a run, $(r(x), T(x), r(x1), \ldots, r(xk)) \in \Delta$. Thus, it follows by definition of Δ that the tuple $(T(x), T(x1), \ldots, T(xk))$ is matching.

(T3) Assume, to the contrary of what is to be shown, that there is a diamond formula $\epsilon_\ell \in T(x)^1$ that is starving in x. Then there exists a path $\gamma = \gamma_1\gamma_2\cdots \in [k]^\omega$ satisfying Conditions 1 and 2 from Definition 9. We show that the third component of $r(x\gamma[i])$ is $\uparrow$ for all $i \geq 2$. Thus, there are no states in $inf_r(x\gamma)$ whose third component is $\oslash$. By definition of F, this is a contradiction to the fact that r is an accepting run.

Hence, let us show that the third component of $r(x\gamma[i])$ is $\uparrow$ for all $i \geq 2$. This is done by induction on i.

- $i = 2$. We distinguish two cases: First, the third component of $r(x\gamma_1)$ is $\oslash$. By Condition 1 of starvation, $\gamma_1 = \ell$. Thus, we have $\epsilon_{\gamma_1} \in T(x)^1$. Condition 2 of starvation yields $T(x\gamma_1)^3 = \gamma_2$. Since we have already proved (T2), we may use (M1) to derive $\epsilon_{\gamma_2} \in T(x\gamma_1)^1$. Together with the fact that the third component of $r(x\gamma_1)$ is $\oslash$, this yields that the third component of $r(x\gamma_1\gamma_2)$ is $\uparrow$ by Condition 4 of the definition of Δ.

Second, the third component of $r(x\gamma_1)$ is $\uparrow$. By Condition 2 of starvation, $T(x\gamma_1) = \gamma_2 \neq 0$. Thus, the third component of $r(x\gamma_1\gamma_2)$ is $\uparrow$ by Condition 4 of the definition of Δ.

- $i > 2$. By induction hypothesis, the third component of $r(x\gamma[i-1])$ is $\uparrow$. Due to Condition 2 of starvation. we have $T(x\gamma[i-1])^3 = \gamma_i$ and $T(x\gamma[i])^3 = \gamma_{i+1} \neq 0$. Thus, the definition of Δ implies that the third component of $r(x\gamma[i])$ is $\uparrow$.

(T4) Suppose $[\alpha]\psi, [\beta]\theta \in T(x)^1$. Let $\pi \in \Pi_0^{(\neg)}$ be a program literal and $q'_\alpha \in Q_\alpha$, $q'_\beta \in Q_\beta$ states such that $q'_\alpha \in \Delta_\alpha(q_\alpha, \pi)$, and $q'_\beta \in \Delta_\beta(q_\beta, \overline{\pi})$. Suppose $[\alpha_{q'_\alpha}]\psi \notin$

$T(y)^1$ for some node $y \in [k]^*$. By Condition (1) in the definition of Q, we have $\{[\alpha]\psi, [\beta]\theta\} \in P_x$. By the definition of Δ, this implies $\{[\alpha]\psi, [\beta]\theta\} \in P_y$. Thus, Condition (2) in the definition of Q yields $[\beta_{q'_\beta}]\theta \in T(y)^1$.

■

Putting together Propositions 11 and 15, it is now easy to establish decidability and EXPTIME-complexity of APDL$^{(\neg)}$ and thus also of PDL$^{(\neg)}$.

THEOREM 17. — *Satisfiability and validity of PDL$^{(\neg)}$-formulas are* EXPTIME-*complete.*

PROOF 18. — From Propositions 11 and 15, it follows that an APDL$^{(\neg)}$-formula φ is satisfiable if and only if $\mathcal{L}(\mathcal{B}_\varphi) \neq \emptyset$. The emptiness problem for Büchi automata is decidable in time quadratic in the size of the automaton [VAR 86]. To show that APDL$^{(\neg)}$-formula satisfiability is in EXPTIME, it thus remains to show that the size of $\mathcal{B}_\varphi = (Q, \Lambda_\varphi, I, \Delta, F)$ is at most exponential in φ.

Let n be the length of φ. Since the cardinality of $\mathsf{cl}(\varphi)$ is polynomial in n, the cardinality of $\mathcal{H}_\varphi$ (the set of Hintikka-sets for φ) is at most exponential in n. Thus, it is readily checked that the same holds for Λ_Φ and Q. The exponential upper bound on the cardinalities of I and F is trivial. It remains to determine the size of Δ: since the size of Q is exponential in n and the out-degree of trees accepted by automata is polynomial in n, we obtain an exponential bound.

Thus, APDL$^{(\neg)}$-formula satisfiability and hence also PDL$^{(\neg)}$-formula satisfiability are in EXPTIME. For the lower bound, it suffices to recall that PDL-formula satisfiability is already EXPTIME-hard [FIS 79]. ■

6. Conclusion

This paper introduces the propositional dynamic logic PDL$^{(\neg)}$, which extends standard PDL with negation of atomic programs. We were able to show that this logic extends PDL in an interesting and useful way, yet retaining its appealing computational properties. There are some natural directions for future work. For instance, it should be simple to further extend PDL$^{(\neg)}$ with the converse operator without destroying the EXPTIME upper bound. It would be more interesting, however, to investigate the interplay between (full) negation and PDL's program operators in some more detail. For example, to the best our our knowledge it is unknown whether the fragment of PDL$^{\neg}$ that has only the program operators "$\neg$" and ";" is decidable.

7. References

[AND 01] ANDRÉKA H., NÉMETI I., SAIN I., "*Handbook of Philosophical Logic*", vol. 2, chapter Algebraic Logic, p. 133–247, Kluwer Academic Publishers, second edition, 2001.

[BAA 03] BAADER F., MCGUINESS D. L., NARDI D., PATEL-SCHNEIDER P., *The Description Logic Handbook: Theory, implementation and applications*, Cambridge University Press, 2003.

[BRO 03] BROERSEN J., "Relativized Action Complement for Dynamic Logics", BALBIANI P., SUZUKI N.-Y., WOLTER F., ZAKHARYASCHEV M., Eds., *Advances in Modal Logics Volume 4*, King's College Publications, 2003, p. 51–69.

[DAN 84] DANECKI S., "Nondeterministic Propositional Dynamic Logic with intersection is decidable", SKOWRON A., Ed., *Proceedings of the Fifth Symposium on Computation Theory*, vol. 208 of *LNCS*, Springer, 1984, p. 34–53.

[De 95] DE GIACOMO G., LENZERINI M., "PDL-based Framework for Reasoning about Actions", *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence (AI*IA'95)*, vol. 992, Springer, 1995, p. 103–114.

[EHR 74] EHRENFEUCHT A., ZEIGER P., "Complexity measures for regular expressions", *Sixth annual ACM Symposium on Theory of Computing*, New York, USA, 1974, ACM Press, p. 75–79.

[FAG 95] FAGIN R., HALPERN J. Y., MOSES Y., VARDI M. Y., *Reasoning About Knowledge*, MIT Press, 1995.

[FIS 77] FISCHER M. J., LADNER R. E., "Propositional modal logic of programs", *Conference record of the ninth annual ACM Symposium on Theory of Computing*, ACM Press, 1977, p. 286–294.

[FIS 79] FISCHER M. J., LADNER R. E., "Propositional Dynamic Logic of Regular Programs", *JCSS*, vol. 18, 1979, p. 194–211.

[GAR 87] GARGOV G., PASSY S., TINCHEV T., "Modal Environment for Boolean Speculations", SKORDEV D., Ed., *Mathematical Logic and Applications*, New York, USA, 1987, Plenum Press, p. 253–263.

[GIA 94] GIACOMO G. D., LENZERINI M., "Boosting the Correspondence between Description Logics and Propositional Dynamic Logics", *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94). Volume 1*, AAAI Press, 1994, p. 205–212.

[GOR 90] GORANKO V., "Modal Definability in Enriched Languages", *Notre Dame Journal of Formal Logic*, vol. 31, num. 1, 1990, p. 81–105.

[GOR 92] GORANKO V., PASSY S., "Using the Universal Modality: Gains and Questions", *Journal of Logic and Computation*, vol. 2, num. 1, 1992, p. 5–30.

[HAR 78] HAREL D., PRATT V., "Nondeterminism in logics of programs", *Proceedings of the Fifth Symposium on Principles of Programming Languages*, ACM, 1978, p. 203–213.

[HAR 84] HAREL D., "Dynamic Logic", GABBAY D. M., GUENTHNER F., Eds., *Handbook of Philosophical Logic, Volume II*, p. 496–604, D. Reidel Publishers, 1984.

[HAR 00] HAREL D., KOZEN D., TIURYN J., *Dynamic Logic*, MIT Press, 2000.

[HUM 83] HUMBERSTONE I. L., "Inaccessible Worlds", *Notre Dame Journal of Formal Logic*, vol. 24, num. 3, 1983, p. 346–352.

[KLE 56] KLEENE S., "Representation of events in nerve nets and finite automata", C.E.SHANNON, J.MCCARTHY, Eds., *Automata Studies*, p. 3–41, Princeton University Press, 1956.

[LUT 00] LUTZ C., SATTLER U., "Mary likes all Cats", BAADER F., SATTLER U., Eds., *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, num. 33

CEUR-WS (http://ceur-ws.org/), 2000, p. 213–226.

[LUT 01] LUTZ C., SATTLER U., "The Complexity of Reasoning with Boolean Modal Logics", WOLTER F., WANSING H., DE RIJKE M., ZAKHARYASCHEV M., Eds., *Advances in Modal Logics Volume 3*, CSLI Publications, Stanford, CA, USA, 2001.

[LUT 04] LUTZ C., WALTHER D., "PDL with Negation of Atomic Programs", *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR'04)*, Lecture Notes in Artifical Intelligence, Springer-Verlag, 2004.

[MAT 67] MATIJASEVICH Y., "Simple examples of undecidable associative calculi", *Soviet mathematics (Doklady)*, vol. 2, num. 2, 1967, p. 555-557.

[PAS 91] PASSY S., TINCHEV T., "An Essay in Combinatory Dynamic Logic", *Information and Computation*, vol. 93, num. 2, 1991.

[PRA 76] PRATT, "Considerations on Floyd-Hoare Logic", *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1976.

[PRE 96] PRENDINGER H., SCHURZ G., "Reasoning about Action and Change: A Dynamic Logic Approach", *Journal of Logic, Language, and Information*, vol. 5, num. 2, 1996, p. 209–245.

[SCH 91] SCHILD K. D., "A correspondence theory for terminological logics: Preliminary report", MYLOPOULOS J., REITER R., Eds., *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, Morgan Kaufmann, 1991, p. 466-471.

[VAR 85] VARDI M. Y., "The Taming of Converse: Reasoning about Two-way Computations", PARIKH R., Ed., *Proceedings of the Conference on Logic of Programs*, vol. 193 of *LNCS*, Springer, 1985, p. 413–424.

[VAR 86] VARDI M. Y., WOLPER P., "Automata-theoretic techniques for modal logic of programs", *Journal of Computer and System Sciences*, vol. 32, 1986, p. 183–221.