Propositional Dynamic Logic with Negation on Atomic Programs

Dirk Walther

January 15, 2004

A thesis submitted to the Dresden University of Technology
for the degree of Master of Computer Science within the
International Masters Program in Computational Logic
at the Department of Computer Science

Supervisor: Dr. Carsten Lutz
Supervising University Professor: Prof. Franz Baader
Institute for Theoretical Computer Science
Department of Computer Science
Dresden University of Technology

Contents

Abstract

Propositional Dynamic Logic (PDL) is a very successful variant of modal logic. In the past, many extensions of PDL have been considered to make this logic even more suitable for applications. Unfortunately, the very interesting extension of PDL with negation of programs is undecidable. This work extends PDL with negation on atomic programs only. The resulting logic is called $\mathrm{PDL}^{(\neg)}$ and has still an interesting expressive power. It is shown that the satisfiability problem of $\mathrm{PDL}^{(\neg)}$ is decidable and ExpTime-complete by using Büchi tree automata.

Declaration

Herewith I confirm that I independently prepared this thesis. No further references or auxiliary means as the ones indicated in this work were used for the preparation.

Signature: __________________________

Dirk Walther

Chapter 1

Introduction

Propositional dynamic logic (PDL) is a variant of modal logic and has been developed as a formal system for reasoning about the dynamic behavior of programs. PDL was first introduced in [FL79] and is known to have a ExpTime-complete satisfiability problem [FL79, Pra79]. For a survey see, e.g., [Har84, HKT00].

Since PDL was introduced, this logic became important for many application areas which could made use of it quite successfully. For instance, among others reasoning about knowledge [FHMV95], reasoning about actions [DL95, PS96], and description logics [GGLM94]. Furthermore, many extensions of PDL have been suggested to make this logic more suitable for applications [PT91, Har84, HKT00]. One extension is the *halt* predicate which was specifically developed for stating program termination in the field of reasoning about programs [HP78]. Contrary to that predicate, most extensions are more general and therefore have been used in many application areas, e.g. PDL extended with the *converse* operator [Var85].

Two interesting extensions of PDL are program intersection "$\cap$" and program negation "$\neg$" [Dan84, Har84, HKT00]. Intersection on programs can be used to model concurrent execution of programs. In [Dan84] it is shown that the resulting logic $\text{PDL}^{\cap}$ is decidable by providing a 2-ExpTime-decision procedure. Negation on programs is more general than program intersection since in $\text{PDL}^{\neg}$, which extends PDL with negation on programs, the intersection $\pi_1 \cap \pi_2$ can be written as $\neg(\neg\pi_1 \cup \neg\pi_2)$ using the program operator choice "$\cup$". Moreover, with negation on programs the universal modality $\Box_U\varphi$ becomes expressible by writing $[a]\varphi \wedge [\neg a]\varphi$ where a is an arbitrary atomic program. The formula $\Box_U\varphi$ holds if and only if the formula φ holds at every world. Universal modality as extension of modal logics is useful for many applications [GP92]. Additionally, due to program negation, the window operator $\boxbar_a$ [Hum83, GPT87, Gor90] can also be expressed in PDL$^\neg$. The formula $\boxbar_a \varphi$ holds at a world w if and only if the formula φ holding at a world w' implies that w' is a-accessible from w. In PDL$^\neg$, the formula $\boxbar_a \varphi$ can be written as $[\neg a]\neg\varphi$. In contrast to the standard box operator in modal logic, which can be seen to express necessity, the window operator expresses sufficiency. Furthermore, the window operator is quite important for applications in description logics [LS00]. Unfortunately, adding negation on programs to PDL yields a bad computational behavior: PDL$^\neg$ is undecidable [Har84].

Due to the appealing and useful expressive power of PDL$^\neg$, it seems to be a reasonable attempt to identify fragments of this logic which still have some interesting properties of program negation, but which are computationally better behaved. From a technical viewpoint, it is interesting to explore the borderline of decidability, i.e., to investigate decidable fragments of PDL$^\neg$ and to ask for their complexity. For instance, one such a fragment is the already mentioned logic PDL$^\cap$ which is known to be decidable in 2-ExpTime—a complexity upper bound. Facing the gap to the ExpTime-complete reasoning of PDL, it is supposed to be still an open problem whether this upper bound is tight. This work studies the fragment PDL$^{(\neg)}$ of PDL$^\neg$ where program negation is restricted to atomic programs only. The resulting theorem of this work states that the satisfiability problem of PDL$^{(\neg)}$ is decidable and ExpTime-complete, i.e. as hard as decidability of PDL. The technique to obtain this complexity result employs finite automata on infinite trees; so-called Büchi tree automata. For a given formula, a Büchi tree automaton is constructed which accepts a tree if and only if this formula is satisfiable. In this way, a decision procedure for PDL$^{(\neg)}$ is constructed which extends standard automata-based decision procedures for PDL [VW86] and for Boolean modal logic [LS01]. On the one hand, program intersection cannot be expressed in PDL$^{(\neg)}$, but on the other hand this logic is still of interest since the universal modality and the window operator remain available.

A more practical motivation for investigating PDL$^{(\neg)}$ is due to its correspondence to Description Logics (DLs). A DL is a formal language to represent and to reason about knowledge; see [BCM$^+$03] for an overview. DLs are closely related to modal languages when regarding concepts as formulas, and roles as PDL programs. In fact, many DLs are notational variants of modal logics [Sch91, Sch94, GGLM94]. For instance, the DL $\mathcal{ALC}_{\text{reg}}$, which extends the basic DL $\mathcal{ALC}$ with regular expression on roles, is a notational variant of PDL [GGLM94]. Adding negation of atomic roles to $\mathcal{ALC}_{\text{reg}}$ yields $\mathcal{ALC}_{\text{reg}}^{(\neg)}$, which in turn is a notational variant of PDL$^{(\neg)}$.

The following illustrates the use of the combination of negation of atomic roles and regular expressions in $\mathcal{ALC}_{\text{reg}}^{(\neg)}$. For this, consider the real-world example in the context of university administration which can be modelled with an $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concept: Private universities commonly prefer to admit students whose ancestors donated money to the university. First, consider the notion of donating ancestors for which building the transitive closure of roles is essential. The transitive closure of a given role parent can be written as $\mathsf{parent;(parent^*)}$ (short parent^+). Clearly, the $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concept $\exists\mathsf{parent}^+.\mathsf{Donator}$ denotes the class of students with donating ancestors. The disadvantage of just introducing a new transitive role $\mathsf{ancestor}$ instead of having the transitive closure of parent is that the relation between the roles $\mathsf{ancestor}$ and parent would be lost. One might want to widen the notion of ancestors by including also parents-in-law, and stepparents. This can be achieved by taking the transitive closure of the union of the roles parent, $\mathsf{parent\text{-}in\text{-}law}$, and $\mathsf{stepparent}$, i.e., then, the notion of ancestor is described by the role $(\mathsf{parent} \cup \mathsf{parent\text{-}in\text{-}law} \cup \mathsf{stepparent})^+$. Using the window operator for DL, the fact that all students with donating ancestors are preferred can be modelled by the $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concept $\forall\neg\mathsf{prefer}.\neg(\exists\mathsf{parent}^+.\mathsf{Donator})$.

In the past, due to the correspondence of DL and modal logics, the research in the complexity of DLs profited from already known complexity results for modal logics. Accordingly, this work contributes to the research of DLs by providing a complexity result for PDL$^{(\neg)}$. More precisely, the satisfiability problem of $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concepts is ExpTime-complete.

This work is organized as follows. Section 2 is about propositional dynamic logic with negation on programs. First, the logic PDL$^{\neg}$ is formally defined, and shown to be undecidable. Then the logics PDL and PDL$^{(\neg)}$ are introduced as fragments of PDL$^{\neg}$. Second, in order to show the correspondence between DLs and modal logics, the definition of the DL $\mathcal{ALC}_{\text{reg}}^{(\neg)}$ is given together with a linear translation of $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concepts into formulas of PDL$^{(\neg)}$. Last, in preparation for the automata-based decision procedure, the variant APDL$^{(\neg)}$ of PDL$^{(\neg)}$ is introduced where programs as modal parameters are replaced by finite automata. Additionally, a polynomial translation of PDL$^{(\neg)}$ into APDL$^{(\neg)}$ is provided. In Section 3, it is shown how models of APDL$^{(\neg)}$-formulas can abstractly be represented as infinite trees; so-called Hintikka-trees. Moreover, a first central lemma of this work is proven which states that an APDL$^{(\neg)}$-formula is satisfiable if and only if this formula has a Hintikka-tree. Section 4 provides first a general definition of Büchi tree automata, and then a construction of a specific Büchi automaton which will be used to check for existence of Hintikka-trees. More precisely, for a given APDL$^{(\neg)}$-formula a Büchi tree automaton is constructed which, in another central lemma, is proven to accept precisely the Hintikka-trees for this formula. Then, checking for emptiness of such automata solves APDL$^{(\neg)}$-satisfiability. Finally, complexity issues are discussed resulting in the ExpTime-completeness of the satisfiability problem for PDL$^{(\neg)}$. In Section 5, the work is concluded by giving possible directions for future research.

Chapter 2

Propositional Dynamic Logic with Negation

2.1 Variants of PDL with Program Negation

In this section, several variants of propositional dynamic logic (PDL) are introduced. First, the definition of $\mathrm{PDL}^{(\neg)}$ allowing for full negation of possibly complex programs is given. Then, the logics PDL and $\mathrm{PDL}^{(\neg)}$ are defined as fragments of $\mathrm{PDL}^{\neg}$. Subsequently, $\mathrm{PDL}^{(\neg)}$ is shown to be undecidable.

Definition 2.1.1 ($\mathrm{PDL}^{\neg}$ - Syntax) Let Φ be a countably infinite set of *propositional variables*, and Π_0 be a countably infinite set of *atomic programs*. The sets $\Pi^{\neg}$ of programs and $\mathrm{PDL}^{\neg}$ of formulas are defined by simultaneous induction, i.e., $\Pi^{\neg}$ and $\mathrm{PDL}^{\neg}$ are the smallest sets such that:

- $\Phi \subseteq \mathrm{PDL}^{\neg}$
- $\Pi_0 \subseteq \Pi^{\neg}$
- if $\varphi, \psi \in \mathrm{PDL}^{\neg}$, then $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathrm{PDL}^{\neg}$
- if $\pi_1, \pi_2 \in \Pi^{\neg}$, then $\neg\pi_1$, $\pi_1 \cup \pi_2$, $\pi_1; \pi_2$, $\pi_1^* \in \Pi^{\neg}$
- if $\pi \in \Pi^{\neg}$, and $\varphi \in \mathrm{PDL}^{\neg}$, then $\langle\pi\rangle\varphi$, $[\pi]\varphi \in \mathrm{PDL}^{\neg}$
- if $\varphi \in \mathrm{PDL}^{\neg}$, then $\varphi? \in \Pi^{\neg}$

The symbol $\top$ is an abbreviation for an arbitrary propositional tautology, and $\bot$ for $\neg\top$. Moreover, for $\pi, \pi' \in \Pi^{\neg}$, $\pi \cap \pi'$ is the 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). $\lhd$

For any formula $\psi \in$ PDL$^{\neg}$, ψ? is called *test*. The program operators "$\neg$", "$\cup$", "$;$", "$*$" are called *negation*, *choice*, *composition*, and *iteration*, respectively. Note that in the fragment PDL$^{(\neg)}$ negated atomic programs can occur nested inside other (complex) programs.

Definition 2.1.2 (PDL$^{\neg}$ - Semantics) Let $\mathfrak{M} = (W, \{R_a \mid a \in \Pi_0\}, V)$ be a *Kripke Structure* where W is the set of worlds, $\{R_a \mid a \in \Pi_0\}$ is the set of *accessibility relations* $R_a \subseteq W^2$ over the set of worlds W for atomic programs $a \in \Pi_0$, and $V : \Phi \to 2^W$ is a valuation function which maps propositional variables to sets of worlds. In the following, the definition of accessibility relations of compound programs and the consequence relation $\models$ is given by simultaneous induction. Let $u \in W$ be a world in $\mathfrak{M}$, $\varphi \in$ PDL$^{\neg}$ a formula, and $\pi \in \Pi^{\neg}$ a program.

$$\begin{array}{lcl}
R_{\varphi?} & := & \{(u,u) \in W^2 \mid \mathfrak{M}, u \models \varphi\} \\
R_{\neg\pi} & := & W^2 \backslash R_\pi, \text{ the complement of } R_\pi \\
R_{\pi_1 \cup \pi_2} & := & R_{\pi_1} \cup R_{\pi_2} \text{ for } \pi_1, \pi_2 \in \Pi^{\neg}, \text{ the union of } R_{\pi_1} \text{ and } R_{\pi_2} \\
R_{\pi_1 ; \pi_2} & := & R_{\pi_1} \circ R_{\pi_2} \text{ for } \pi_1, \pi_2 \in \Pi^{\neg}, \text{ the composition of } R_{\pi_1} \text{ and } R_{\pi_2} \\
R_{\pi^*} & := & (R_\pi)^* \text{ for } \pi \in \Pi^{\neg}, \text{ the reflexive transitive closure of } R_\pi
\end{array}$$

$$\begin{array}{lll}
\mathfrak{M}, u \models p & \text{iff} & u \in V(p) \text{ for any } p \in \Phi \\
\mathfrak{M}, u \models \neg\varphi & \text{iff} & \mathfrak{M}, u \not\models \varphi \\
\mathfrak{M}, u \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathfrak{M}, u \models \varphi_1 \text{ or } \mathfrak{M}, u \models \varphi_2 \\
\mathfrak{M}, u \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathfrak{M}, u \models \varphi_1 \text{ and } \mathfrak{M}, u \models \varphi_2 \\
\mathfrak{M}, u \models \langle\pi\rangle\varphi & \text{iff} & \text{there is a } v \in W \text{ with } (u,v) \in R_\pi \text{ and } \mathfrak{M}, v \models \varphi \\
\mathfrak{M}, u \models [\pi]\varphi & \text{iff} & \text{for all } v \in W, (u,v) \in R_\pi \text{ implies } \mathfrak{M}, v \models \varphi
\end{array}$$

If for some world $u \in W$ it holds $\mathfrak{M}, u \models \varphi$, then formula φ is *true* at u in $\mathfrak{M}$, and $\mathfrak{M}$ is called *model* of φ. A formula is *satisfiable* if it has a model. A formula is *valid* if it is true at all worlds in any Kripke structure. $\lhd$

Note that the semantics for both fragments, $\mathrm{PDL}^{(\neg)}$ and PDL, is the same as for $\mathrm{PDL}^{\neg}$.

Remark. The logic PDL is non-deterministic. The deterministic version DPDL differs in the semantics, i.e., formulas of DPDL are only interpreted over Kripke structures whose accessibility relations for atomic programs are functions that assign maximally one successor to each world. Note that talking about the deterministic versions of $\mathrm{PDL}^{\neg}$ and $\mathrm{PDL}^{(\neg)}$ does not make much sense. One cannot w.l.o.g. assume that both, the accessibility relation for atomic programs and their negation, are functional since this would restrict the size of Kripke structures to only two worlds.

In [Har84] it is shown that the logic $\mathrm{PDL}^{\neg}$ is undecidable. Since this undecidability result can be established quite easily, a simple proof is given in the following. The proof is done by reduction of the undecidable word-problem of finitely represented semigroups [Pos47] to the satisfiability problem of $\mathrm{PDL}^{\neg}$. Let the tuple $(A, ;, a_1, \ldots, a_n)$ be a semigroup, i.e., the carrier set A is closed under the binary operation ";" and the constants $a_1, \ldots, a_n$. Think of the constants $a_1, \ldots, a_n$ as smallest possible words, and of ";" as composition such that applying it produces more complex words. Each word gets assigned to an element of the carrier A. An equation of two words $w \approx w'$ is *true* if both words get assigned to the same element of A. Then the word-problem is as follows: Given a set of word-equations $\{w_1 \approx w_1', \ldots, w_k \approx w_k'\}$, the problem is to decide whether this implies another word-equation $w \approx w'$. In order to reduce this problem to $\mathrm{PDL}^{\neg}$-satisfiability, the universal modality $\Box_U \varphi$ is needed, which has the following semantics:

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

As mentioned in the introduction, the universal modality $\Box_U \varphi$ is expressible in $\mathrm{PDL}^{\neg}$ with its equivalent $[a]\varphi \wedge [\neg a]\varphi$ where $a \in \Pi_0$ is an arbitrary atomic program. Assume that for every constant $a_1, \ldots, a_n$ there is an atomic program with the same name. Then the reduction is as follows: $\{w_1 \approx w_1', \ldots, w_k \approx w_k'\}$ implies $w \approx w'$ if and only if the following formula is unsatisfiable:

$$(\langle w \cap \neg w' \rangle \top \vee \langle \neg w \cap w' \rangle \top) \wedge \Box_U (\bigwedge_{1 \le i \le k} [w_i \cap \neg w_i'] \bot \wedge [\neg w_i \cap w_i'] \bot).$$

This shows that $\mathrm{PDL}^{\neg}$ is undecidable.

On the other hand, it is well known that satisfiability for PDL is ExpTime-complete where ExpTime-decidability is shown in [Pra79], and [FL79] shows its ExpTime-hardness. According to the introduction, the aim was to identify decidable fragments of PDL$^{\neg}$ which extend PDL in an useful way. The main objective of this work is to show that PDL$^{(\neg)}$ is such a fragmemnt, i.e., it is proven that PDL$^{(\neg)}$-satisfiability is decidable and ExpTime-complete.

2.2 Correspondence of PDL$^{(\neg)}$ to Description Logic

This section provides a formal definition of the DL $\mathcal{ALC}^{(\neg)}_{\text{reg}}$. Moreover, the correspondence between PDL$^{(\neg)}$ and $\mathcal{ALC}^{(\neg)}_{\text{reg}}$ is shown by defining a function which translates $\mathcal{ALC}^{(\neg)}_{\text{reg}}$-concepts into formulas of PDL$^{(\neg)}$.

First, consider the definition of the syntax and the semantics of $\mathcal{ALC}^{(\neg)}_{\text{reg}}$.

Definition 2.2.1 ($\mathcal{ALC}^{(\neg)}_{\text{reg}}$ - Syntax, Semantics) Let $\mathcal{N}_C$ be a set of *concept names*, and $\mathcal{N}_R$ a set of *role names*. The set $\mathcal{R}^{(\neg)}$ is the set of *role literals*, i.e., the set of role names in $\mathcal{N}_R$ together with their negation. Define the $\mathcal{ALC}^{(\neg)}$-concepts and the set $\mathcal{R}^{(\neg)}_{\text{reg}}$ of roles simultaneously as the smallest sets such that:

- $\mathcal{N}_C \subseteq \mathcal{ALC}^{(\neg)}_{\text{reg}}$
- $\mathcal{R}^{(\neg)} \subseteq \mathcal{R}^{(\neg)}_{\text{reg}}$
- if $R, S \in \mathcal{R}^{(\neg)}_{\text{reg}}$, then $R \circ S$, $R \sqcup S$, $R^* \in \mathcal{R}^{(\neg)}_{\text{reg}}$
- if $C, D \in \mathcal{ALC}^{(\neg)}_{\text{reg}}$, and $R \in \mathcal{R}^{(\neg)}_{\text{reg}}$, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, $\exists R.C \in \mathcal{ALC}^{(\neg)}_{\text{reg}}$
- if $C \in \mathcal{ALC}^{(\neg)}_{\text{reg}}$, then $id(C) \in \mathcal{R}^{(\neg)}_{\text{reg}}$

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an *interpretation* where $\Delta^{\mathcal{I}}$ is a set, the *domain* of $\mathcal{I}$, and $\cdot^{\mathcal{I}}$ is a function mapping every atomic concept in $\mathcal{N}_C$ to a subset of $\Delta^{\mathcal{I}}$, and every atomic role in $\mathcal{N}_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Inductively augment the function $\cdot^{\mathcal{I}}$ to all concepts in $\mathcal{ALC}^{(\neg)}_{\text{reg}}$ and all roles in $\mathcal{R}^{(\neg)}_{\text{reg}}$ as follows:

$$\begin{array}{llll}
(\neg C)^{\mathcal{I}} & = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} & (\neg R)^{\mathcal{I}} & = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \backslash R^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = C^{\mathcal{I}} \cup D^{\mathcal{I}} & (R \sqcup S)^{\mathcal{I}} & = R^{\mathcal{I}} \cup S^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = C^{\mathcal{I}} \cap D^{\mathcal{I}} & (R \circ S)^{\mathcal{I}} & = R^{\mathcal{I}} \circ S^{\mathcal{I}} \\
(id(C))^{\mathcal{I}} & = \{(x,x) \mid x \in C^{\mathcal{I}}\} & (R^*)^{\mathcal{I}} & = (R^{\mathcal{I}})^*
\end{array}$$

$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{there is a } y \in \Delta^{\mathcal{I}} \text{ with } (x,y) \in R^{\mathcal{I}}, \text{ and } y \in C^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for all } y \in \Delta^{\mathcal{I}}, \text{ if } (x,y) \in R^{\mathcal{I}}, \text{ then } y \in C^{\mathcal{I}}\}$$

A concept C is *satisfiable* if $C^{\mathcal{I}} \neq \emptyset$. An interpretation $\mathcal{I}$ is a *model* of a concept C if C is *satisfiable.* $\lhd$

In the following, the correspondence of the DL $\mathcal{ALC}^{(\neg)}_{\text{reg}}$ and the modal logic PDL$^{(\neg)}$ is discussed, and a translation of $\mathcal{ALC}^{(\neg)}_{\text{reg}}$-concepts into formulas of PDL$^{(\neg)}$ defined.

First, look at the similarity in the syntax of both logics which can be seen when regarding $\mathcal{ALC}^{(\neg)}_{\text{reg}}$-concepts and -roles as formulas and programs of PDL$^{(\neg)}$, respectively. Atomic concepts correspond to propositional variables, and role literals to program literals. Moreover, all operators for building complex concepts and roles in $\mathcal{ALC}^{(\neg)}_{\text{reg}}$ have a counterpart in PDL$^{(\neg)}$.

The semantical correspondence of both logics is due to similar interpretation structures. Observe the similarity of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and a Kripke structure $\mathfrak{M} = (W, \{R_\pi \mid \pi \in \Pi_0\}, V)$: The domain $\Delta^{\mathcal{I}}$ corresponds to the set of worlds W, an atomic concept $C \in \mathcal{N}_C$ and a propositional variable $p \in \Phi$ are interpreted as a set of individuals $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and as a set of worlds $V(p) \subseteq W$, respectively, and an atomic role $R \in \mathcal{N}_R$ and an atomic program $\pi \in \Pi_0$ are interpreted as binary relations $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $R_\pi \subseteq W \times W$, respectively. Moreover, the constructs for building complex concepts and roles, and formulas and programs are interpreted similarly.

In fact, both logics are just notational variants of each other. To illustrate this, define a function t_1 which maps $\mathcal{ALC}^{(\neg)}_{\text{reg}}$-concepts and -roles to formulas and programs of PDL$^{(\neg)}$ as follows:

$$
\begin{array}{lcl}
t_1(C) & := & p_c \text{ where } p_c \in \Phi, \text{ for atomic concept } C \in \mathcal{N}_C; \\
t_1(\neg C) & := & \neg(t_1(C)); \\
t_1(C \wedge D) & := & t_1(C) \wedge t_1(D); \\
t_1(C \vee D) & := & t_1(C) \vee t_1(D); \\
t_1(\forall R.C) & := & [t_1(R)]t_1(C); \\
t_1(\exists R.C) & := & \langle t_1(R)\rangle t_1(C); \\
t_1(R) & := & \pi_R \text{ where } \pi_R \in \Pi_0^{(\neg)}, \text{ for role literals } R \in \mathcal{R}^{(\neg)}; \\
t_1(R_1 \sqcup R_2) & := & t_1(R_1) \cup t_1(R_2); \\
t_1(R_1 \circ R_2) & := & t_1(R_1); t_1(R_2); \\
t_1(R^*) & := & (t_1(R))^*;
\end{array}
$$

where $\Pi_0^{(\neg)}$ is the set of program literals containing atomic programs together with their negation.

Since $\mathcal{ALC}_{\text{reg}}^{(\neg)}$ and PDL$^{(\neg)}$ are notational variants of each other, it is easy to see that a concept C is satisfiable if and only if the formula $t_1(C)$ is satisfiable.

Now, with the help of function t_1 it is possible to translate the DL-example for $\mathcal{ALC}_{\text{reg}}^{(\neg)}$ in the introduction into PDL$^{(\neg)}$. Let the atomic role prefer correspond to the atomic program π_{prefer}, and the atomic concept Donator to the propositional variable p_{Donator}, respectively. Then, the $\mathcal{ALC}_{\text{reg}}^{(\neg)}$-concept of the introduction is translated into a PDL$^{(\neg)}$-formula as follows:

$$t_1(\forall\neg\mathsf{prefer}.\neg(\exists\mathsf{parent};(\mathsf{parent}^*).\mathsf{Donator})) = [\neg\pi_{\mathsf{prefer}}]\neg(\langle\pi_{\mathsf{parent}};(\pi^*_{\mathsf{parent}})\rangle p_{\mathsf{Donator}}).$$

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

The decision procedure for PDL$^{(\neg)}$ uses Büchi tree automata which accept infinite trees. In order to prove the result, it is more convenient to use a variant of PDL$^{(\neg)}$ where programs are represented by finite automata, rather than by regular expressions. A similar approach where finite automata replace regular expressions can be found in [VW86].

In the following, a variant of PDL$^{(\neg)}$, called APDL$^{(\neg)}$, is defined replacing regular expressions as modal parameter with finite automata. But first, finite automata are introduced.

Definition 2.3.1 (Finite automata) A *(nondeterministic) finite automaton* $\mathcal{A}$ is defined as $\mathcal{A} := (Q, \Sigma, q_0, \Delta, F)$ where Q is a finite set of states, Σ a finite alphabet, q_0 an initial state, $\Delta : Q \times \Sigma \to 2^Q$ a transition function, and $F \subseteq Q$ is the set of accepting states. Note that for a state $q \in Q$ and letter $a \in \Sigma$ the set $\Delta(q, a)$ contains the possible successors of q.
Inductively extend Δ to $Q \times \Sigma^*$ as follows: for any $q \in Q$, $\Delta(q, \varepsilon) := \{q\}$ where ε is the empty word, and $\Delta(q, wa) := \{q'' \in Q \mid q'' \in \Delta(q', a)$ for some $q' \in \Delta(q, w)\}$ where $w \in \Sigma^*$, and $a \in \Sigma$.
A sequence $p_0 \cdots p_n \in Q^*$, $n \geq 0$, of states 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 any $0 < i \leq n$, 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 $\mathcal{A}\}$. $\lhd$

Now, the syntax and the semantics of the variant APDL$^{(\neg)}$ is defined.

Definition 2.3.2 (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 APDL$^{(\neg)}$ of formulas are defined by simultaneous induction, i.e., $A\Pi^{(\neg)}$ and APDL$^{(\neg)}$ are the smallest sets such that:

- $\Phi \subseteq$ APDL$^{(\neg)}$
- if $\varphi, \psi \in$ APDL$^{(\neg)}$, then $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi \in$ APDL$^{(\neg)}$
- if $\alpha \in A\Pi^{(\neg)}$ and $\varphi \in$ APDL$^{(\neg)}$, then $\langle\alpha\rangle\varphi$, $[\alpha]\varphi \in$ APDL$^{(\neg)}$
- if α is a finite automaton with alphabet $\Sigma \subseteq \Pi_0^{(\neg)} \cup \{\psi? \mid \psi \in ADPL^{(\neg)}\}$, then $\alpha \in A\Pi^{(\neg)}$

$\lhd$

For any formula $\psi \in$ APDL$^{(\neg)}$, $\psi?$ is called *test*. Note that the alphabet of an program automaton is a finite set of atomic programs, negated atomic programs, and tests.

Definition 2.3.3 (APDL$^{(\neg)}$ - Semantics) Let $\mathfrak{M} = (W, \{R_a \mid a \in \Pi_0\}, V)$ be a Kripke structure as in Definition 2.1.2. Inductively define a relation R which maps each program literal, each test, and each program automaton to a binary relation over W. This is done simultaneously with the inductive definition of the consequence relation $\models$. Let $\alpha \in A\Pi^{(\neg)}$ be a program automaton, $u \in W$ a world in $\mathfrak{M}$, and $\varphi \in \text{APDL}^{(\neg)}$ a formula.

$$\begin{array}{lcl}
R(a) & := & R_a \text{ for any } a \in \Pi_0 \\
R(\neg a) & := & W^2 \backslash R_a \text{ for any } a \in \Pi_0 \\
R(\psi?) & := & \{(u,u) \in W^2 \mid \mathfrak{M}, u \models \psi\} \\
R(\alpha) & := & \{(u,v) \in W^2 \mid \text{there is a word } w = w_1 \cdots w_m \in \mathcal{L}(\alpha),\ m \geq 0, \\
 & & \text{and worlds } u_0, \ldots, u_m \in W \text{ with} \\
 & & u = u_0 R(w_1) u_1 R(w_2) \cdots u_{m-1} R(w_m) u_m = v\}
\end{array}$$

$$\begin{array}{lll}
\mathfrak{M}, u \models p & \text{iff} & u \in V(p) \text{ for any } p \in \Phi, \\
\mathfrak{M}, u \models \neg\varphi & \text{iff} & \mathfrak{M}, u \not\models \varphi, \\
\mathfrak{M}, u \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathfrak{M}, u \models \varphi_1 \text{ or } \mathfrak{M}, u \models \varphi_2, \\
\mathfrak{M}, u \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathfrak{M}, u \models \varphi_1 \text{ and } \mathfrak{M}, u \models \varphi_2, \\
\mathfrak{M}, u \models \langle\alpha\rangle\varphi & \text{iff} & \text{there is a } u' \in W \text{ with } (u,u') \in R(\alpha) \text{ and } \mathfrak{M}, u' \models \varphi, \\
\mathfrak{M}, u \models [\alpha]\varphi & \text{iff} & \text{for all } u' \in W, (u,u') \in R(\alpha) \text{ implies } \mathfrak{M}, u' \models \varphi.
\end{array}$$

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

Since a regular set represented by a regular expression is equivalent to the language of a finite automaton [Kle56], obviously both logics, $\text{PDL}^{(\neg)}$ and $\text{APDL}^{(\neg)}$, have the same expressive power.

2.4 Translation of $\text{PDL}^{(\neg)}$ into $\text{APDL}^{(\neg)}$

This section provides a method of how $\text{PDL}^{(\neg)}$-formulas can be translated into formulas of $\text{APDL}^{(\neg)}$ with no essential grow in size. In fact, the translation is linear.

Definition 2.4.1 (Translation) By simultaneous induction define the function $t_2 : \text{PDL}^{(\neg)} \to \text{APDL}^{(\neg)}$, and the finite automaton $\mathcal{A}^{\pi} := (Q^{\pi}, \Sigma^{\pi}, q_i^{\pi}, \Delta^{\pi}, \{q_f^{\pi}\})$

for programs $\pi \in \Pi^{(\neg)}$. First, consider the induction on the structure of φ.

$$\begin{array}{lcl} t_2(p) & := & p \text{ for } p \in \Phi \\ t_2(\neg\psi) & := & \neg t_2(\psi) \\ t_2(\psi \wedge \theta) & := & t_2(\psi) \wedge t_2(\theta) \\ t_2(\psi \vee \theta) & := & t_2(\psi) \vee t_2(\theta) \\ t_2(\langle\pi\rangle\psi) & := & \langle\mathcal{A}^\pi\rangle t_2(\psi) \\ t_2([\pi]\psi) & := & [\mathcal{A}^\pi]t(\psi) \\ t_2(\psi?) & := & t_2(\psi)? \end{array}$$

Then, consider the induction on the structure of π. In the induction base, π is a program literal, or π is a test $\psi?$.

$$\begin{array}{lcl} Q^\pi & := & \{q_i, q_f\} \\ \Sigma^\pi & := & \begin{cases} \{t_2(\psi?)\} & \text{if } \pi = \psi? \\ \{\pi\} & \text{if } \pi \in \Pi_0^{(\neg)} \end{cases} \\ q_i^\pi & := & q_i \\ \Delta^\pi & := & \{(q_i, \pi, \{q_f\})\} \\ q_f^\pi & := & q_f \end{array}$$

In the first two cases of the induction step, π is $\pi_1 \cup \pi_2$, or $\pi_1; \pi_2$.

$$\begin{array}{lcl} Q^\pi & := & Q^{\pi_1} \cup Q^{\pi_2} \\ \Sigma^\pi & := & \Sigma^{\pi_1} \cup \Sigma^{\pi_2} \\ q_i^\pi & := & q_i^{\pi_1} \\ \Delta^\pi & := & \Delta^{\pi_1} \cup \Delta^{\pi_2} \\ q_f^\pi & := & q_f^{\pi_2} \end{array}$$

In the case where $\pi = \pi_1 \cup \pi_2$, the initial (final) state $q_i^{\pi_1}$ ($q_f^{\pi_1}$) of $\mathcal{A}^{\pi_1}$ is set equivalent to the initial (final) state $q_i^{\pi_2}$ ($q_f^{\pi_2}$) of $\mathcal{A}^{\pi_2}$, i.e., $q_i^{\pi_1} = q_i^{\pi_2}$ and $q_f^{\pi_1} = q_f^{\pi_2}$. In the case where $\pi = \pi_1; \pi_2$, the final state $q_f^{\pi_1}$ of $\mathcal{A}^{\pi_1}$ is set equivalent to the initial state $q_i^{\pi_2}$ of $\mathcal{A}^{\pi_2}$, i.e., $q_f^{\pi_1} = q_i^{\pi_2}$.

In the last case of the induction step, π is σ^* where $\sigma \in \Pi^{(\neg)}$.

$$\begin{aligned}
Q^\pi &:= Q^\sigma \cup \{q_i^{\sigma^*}, q_f^{\sigma^*}\} \\
\Sigma^\pi &:= \Sigma^\sigma \\
q_i^\pi &:= q_i^{\sigma^*} \\
\Delta^\pi &:= \Delta^\sigma \cup \{(q_i^{\sigma^*}, \varepsilon, q_i^\sigma), (q_f^\sigma, \varepsilon, q_f^{\sigma^*})\} \\
q_f^\pi &:= q_f^{\sigma^*}
\end{aligned}$$

$\lhd$

The function t_2 translates PDL$^{(\neg)}$-formulas into formulas of APDL$^{(\neg)}$ by replacing regular expressions $\pi \in \Pi^{(\neg)}$, which are modal parameters in PDL$^{(\neg)}$-formulas, by finite automata $\mathcal{A}^\pi$. In order to maintain the truth of translated formulas, the language of the finite automaton $\mathcal{A}^\pi$ is the same regular set as represented by the regular expression $\pi \in \Pi^{(\neg)}$ which is easy to verify. Note that actually the language of $\mathcal{A}^\pi$ is still different because of the tests. Formulas used for tests are translated by t_2 as well.

The translation is linear in the size of the input formula, i.e., for a PDL$^{(\neg)}$-formula φ the length of APDL$^{(\neg)}$-formula $t_2(\varphi)$ is linear in the length of φ. To see this, observe that in the inductive definition of t_2 most cases are obviously linear, only the modal cases where a regular expression $\pi \in \Pi^{(\neg)}$ is replaced by the automaton $\mathcal{A}^\pi$ need a closer look. In the definition of $\mathcal{A}^\pi$, only the cases in the induction step are interesting for investigating complexity. The first two cases for choice and composition yield no grow in size of the resulting automaton since neither states, alphabet symbols, nor transitions are added. The last case, for program iteration, is also linear, because it just augments the automaton with two new states and two new transitions which yields a constant grow in size.

Note that due to this linear translation, APDL$^{(\neg)}$ is more general than PDL$^{(\neg)}$ in the sense that any upper complexity bound for APDL$^{(\neg)}$-satisfiability automatically holds for PDL$^{(\neg)}$ as well. The converse is not true since APDL$^{(\neg)}$-formulas can be exponentially more succinct than formulas of PDL$^{(\neg)}$. This is because it is not possible to translate program automata back to regular expression without an exponential blowup.

Chapter 3

Hintikka-Trees

In this chapter, models of APDL$^{(\neg)}$-formulas are abstractly represented as infinite trees; so-called Hintikka-trees. Moreover, it is proven in a first central lemma that an APDL$^{(\neg)}$-formula is satisfiable if and only if there exists a Hintikka-tree for this formula.

The approach for deciding APDL$^{(\neg)}$'s satisfiability problem makes use of the notion of Hintikka-trees as follows: An APDL$^{(\neg)}$-formula φ is translated into a Büchi tree automaton $\mathcal{B}_\varphi$ which accepts precisely the Hintikka-trees for φ. That is, the language of $\mathcal{B}_\varphi$ is non-empty if and only if φ has a model. To decide satisfiability of φ, it remains to perform an emptiness-test on the automaton $\mathcal{B}_\varphi$ which is computationally easy.

An obstacle in representing APDL$^{(\neg)}$-models as trees is the fact that APDL$^{(\neg)}$ does not have the *tree model property*. For instance, the PDL$^{(\neg)}$-formula

$$p \wedge [\neg a]\neg p, \; a \in \Pi_0,$$

which can be translated in to APDL$^{(\neg)}$, has no tree model. But nevertheless, it is possible to construct a tree-like representation of these (possibly non-tree) models which is done in the following.

3.1 Negation Normal Form and Notation

Further on, it is assumed that all formulas in APDL$^{(\neg)}$ are in negation normal form (NNF), i.e., negation only occurs in front of propositional variables. The following notation for negation of APDL$^{(\neg)}$-formulas and program literals in $\Pi_0^{(\neg)}$ is used:

- for each $\varphi \in \text{APDL}^{(\neg)}$, $\overline{\varphi}$ denotes the NNF of $\neg\varphi$,
- for each $\pi \in \Pi_0^{(\neg)}$, $\overline{\pi}$ denotes $\neg\pi$ if $\pi \in \Pi_0$, and σ if $\pi = \neg\sigma$ where $\sigma \in \Pi_0$.

For identification purposes, the tuple components of a program automaton $\alpha \in A\Pi^{(\neg)}$ are labelled with an identifying subscript: $\alpha = (Q_\alpha, \Sigma_\alpha, q_\alpha, \Delta_\alpha, F_\alpha)$. For each program automaton α, and each state $q \in Q_\alpha$, let $\alpha_q := (Q_\alpha, \Sigma_\alpha, q, \Delta_\alpha, F_\alpha)$ be the automaton α with new initial state q.

3.2 Closure $cl(\varphi)$

As a first step, a *closure* for APDL$^{(\neg)}$-formulas is introduced, analogous to [FL79, VW86] for PDL. The closure is a set $cl(\varphi)$ that contains those formulas which are relevant for deciding satisfiability of φ. That is, the set $cl(\varphi)$ contains subformulas of φ, their negation, tests in the alphabet of φ's automata, and formulas with automata whose initial states are changed.

In the following, with a *subformula* ψ of a formula φ it is meant that ψ can be obtained from φ by decomposing only the formula operators, but not decomposing the program automata in φ. For instance, φ is a subformula of $\langle\alpha\rangle\varphi$, but test-formulas in the alphabet Σ_α are not.

Definition 3.2.1 Let φ be an APDL$^{(\neg)}$-formula. The set $cl(\varphi)$ is the smallest set which is closed under the following conditions:

(C1) $\varphi \in cl(\varphi)$

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

(C3) if $\psi \in cl(\varphi)$, then $\overline{\psi} \in cl(\varphi)$

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

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

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

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

$\lhd$

In the sequel, it is assumed that all diamond formulas in $cl(\varphi)$ are linearly ordered, and ϵ_i with $i \geq 1$ denotes the i-th diamond formula in $cl(\varphi)$. Note that a changed initial state of an program automaton results in a different diamond formula.

In order to get the ExpTime-complete complexity result for APDL$^{(\neg)}$--satisfiability in the end, it is important that the cardinality of the closure $cl(\varphi)$ is at most polynomial in the length of φ. For this, it becomes necessary to assume that APDL$^{(\neg)}$-formulas are not represented too succinct. Otherwise, for instance when program automata are encoded exponentially succinct, it would be possible to extract exponentially many formulas with the Conditions (C4) to (C7). To avoid this, it is further on assumed that any program automaton $\alpha \in A\Pi^{(\neg)}$ is encoded by just separately writing down the automaton components Q_α, q_α, Σ_α, Δ_α, F_α where sets are represented element by element.

With the help of this assumption, the cardinality of the closure $cl(\varphi)$ can be proven to be polynomial in the length of φ.

Claim 1 Let φ be an APDL$^{(\neg)}$-formula. The size of the set $cl(\varphi)$ is polynomial in the length of φ.

Proof of Claim Concerning the first three Conditions (C1) to (C3), the size of $cl(\varphi)$ stays polynomial w.r.t. the length of φ. To see this, note that there are only linearly many subformulas of an APDL$^{(\neg)}$-formula, and adding the NNF of the negation of $cl(\varphi)$-formulas to the closure increases its cardinality only by a constant factor two. The Conditions (C4) to (C7) deal with modal formulas where (C4) and (C6) add formulas used for tests, and (C5) and (C7) add modal formulas with changed initial state of their program automata to the closure.

Consider the Conditions (C4) and (C6). From the assumption above, it follows that the size of a program automaton α is polynomial in the length of all test-formulas in its alphabet Σ_α together. Then, w.r.t. to the length of φ, at most polynomially many formulas used for tests are added to $cl(\varphi)$. Note that these test-formulas in their turn contribute at most polynomially many formulas w.r.t. their length to the closure.

Consider the Conditions (C5) and (C7). By assumption, the number of states in Q_α is polynomial in the length of α. Consequently, changing the initial state of α in box and diamond formulas yields only polynomially many different formulas which are added to $cl(\varphi)$. Note that these formulas with different initial state of α share the same subformulas. ◀

3.3 Representation of Models as Hintikka-Trees

In order to define Hintikka-trees as tree-like representation of APDL$^{(\neg)}$-models, infinite trees are employed. This is done in the following way. First, *Hintikka-sets* are defined as consistent subsets of the closure $cl(\varphi)$. Hintikka-sets are used to label nodes in infinite trees. The intuition is that every node in such a tree is meant to represent a world in the corresponding model. A Hintikka-set labelling a node contains exactly those $cl(\varphi)$-formulas which are true at the world represented by this set. As a second step, a *matching relation* is introduced which regulates the relations between adjacent nodes in the tree. By "pushing" box and diamond formulas to the Hintikka-sets of their immediate successors, the matching relation models the "local" influence of modal formulas on their successors. Next, *diamond starvation* is defined. With the help of this notion the "non-local" impact of diamond formulas whose program automata might allow for arbitrary long words is captured. More precisely, with this definition it can be made sure that diamond formulas cannot be pushed infinitely far along the tree, i.e. that they get satisfied at some node. Last, by putting all these notions together, *Hintikka-trees* are defined.

Definition 3.3.1 (Hintikka-set) Let $\psi \in$ APDL$^{(\neg)}$ be a formula, and $\alpha \in A\Pi^{(\neg)}$ a program automaton. The set $\Psi \subseteq cl(\varphi)$ 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 $\overline{\psi} \notin \Psi$

(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 $\overline{\theta} \in \Psi$ or $[\alpha_q]\psi \in \Psi$

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

The Conditions (H1) and (H2) deal with boolean cases and are straightforward. Condition (H3) forces maximality of Hintikka-sets by stating that, for each $cl(\varphi)$-formula, either the formula itself or its negation must be in the Hintikka-set. Later, this condition will be used to deal with negated atomic programs. The last two Conditions (H4) and (H5) deal with the "local" impact of box formulas within the same Hintikka-set. Note that, in (H5), the program automaton α_q corresponds to the automaton α with new initial state q.

The matching relation models how box and diamond formulas are pushed to successor nodes. Consider the following intuition: In a Hintikka-tree, every diamond formula in a Hintikka-set of a node has its own successor node. Pushing a diamond formula to the Hintikka-set of its successor corresponds to travelling along a single program literal such that the program automaton can "advance" to a next state. Then, this program literal connects the node with this successor. Generally, in order to satisfy a diamond formula, it can be necessary to perform a sequence of such moves since the program automaton of the diamond might allow for arbitrary long words. The matching relation is necessary (but not sufficient) to ensure that diamond formulas get satisfied, either in the node where they occur, or, if necessary, by pushing them to their corresponding successors node by node along the tree. Moreover, this relation makes sure that box formulas are not violated by any successor.

In order to define the matching relation, fix the structure for the node labels. Let φ be a APDL$^{(\neg)}$-formula, and denote with k the number of diamond formulas in the closure $cl(\varphi)$. Let $\Pi_\varphi^{(\neg)}$ be the set of all program literals occurring in φ. Then, the set of possible node-labels for Hintikka-trees for φ is defined as a set of triples Λ_φ with

$$\Lambda_\varphi := \mathcal{H}_\varphi \times \Pi_\varphi^{(\neg)} \cup \{\bot\} \times \{0, \ldots, k\}.$$

In a Λ_φ-triple, which labels a node, the first component is a Hintikka-set containing those $cl(\varphi)$-formulas which are true at the corresponding world in the model. The second component contains the program literal of φ connecting the node with its predecessor; the second component is $\bot$ when this information is irrelevant. The third component is a number at most k which indicates a diamond formula in the Hintikka-set of this label. Later, this information is used to trace diamonds through the tree and to make sure they eventually get satisfied. For any triple $\lambda \in \Lambda_\varphi$ refer to the first, second, and third triple component with λ^1, λ^2, and λ^3, respectively.

Definition 3.3.2 (Matching) Let $\varphi \in$ APDL$^{(\neg)}$ be a formula, and k the number of diamond formulas in $cl(\varphi)$. A $k+1$-tuple of Λ_φ-triples $(\lambda, \lambda_1, \ldots, \lambda_k)$ is *matching* if, for all $1 \leq i \leq k$ and all automata $\alpha \in A\Pi^{(\neg)}$, it 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 $\psi_1, \ldots, \psi_n \in \lambda^1$, $q_1 \in \Delta_\alpha(q_\alpha, w)$ and one of the following conditions hold:

(a) $q_1 \in F_\alpha$ 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 $\pi \in \Sigma_\alpha$ is a program literal such that $q \in \Delta_\alpha(q_\alpha, \pi)$, then $\pi = \lambda_i^2$ implies $[\alpha_q]\psi \in \lambda_i^1$

$\lhd$

The matching relation contains possible candidates of labels for nodes and their immediate successors. Think of λ as a label of a node, and of $\lambda_1, \ldots, \lambda_k$ as the labels of its k successors. Condition (M1) deals with diamond-pushing. Suppose the diamond formula $\epsilon_i = \langle\alpha\rangle\psi$ is in the Hintikka-set λ^1. Now, there are two possibilities: either ϵ_i can be satisfied inside λ^1 as it is considered in (M1)(a), or the satisfaction of ϵ_i is "delayed" by pushing it to the Hintikka-set λ_i^1 of the i-th successor which is regulated by (M1)(b). In the case of (M1)(a), the program automaton α can reach a final state by accepting a possible empty sequence of tests such that these test-formulas are also contained in λ^1. Then, the diamond formula ϵ_i is not pushed and the formula ψ without the diamond is forced to be in λ^1. In this case, the information how the current node is connected with its i-th successor is irrelevant, therefore λ_i^2 is set to $\bot$. There is also no need to indicate further diamond formulas in λ_i^1, thus λ_i^3 is set to 0. This is different in the case of (M1)(b), where ϵ_i is pushed to its i-th successor using a program literal π such that α "advances" to a state q. Then, λ_i^2 is set to π since the current node is connected with its i-th successor by this program literal. The pushed diamond formula $\epsilon_j = \langle\alpha_q\rangle\psi$ is forced to be in the Hintikka-set λ_i^1 of the i-th successor. Note that diamond formulas change while being pushed. Moreover, the pushed diamond formula ϵ_j in λ_i^1 is indicated by setting λ_i^3 to j. This indication makes it possible to trace the path along which the changing diamond formulas are pushed.

The Condition (M2) is about box-pushing. More precisely, a box formula $[\alpha]\psi$ in λ^1 is pushed to the Hintikka-set of a successor, when this successor is connected with a program literal which is applicable to α such that α can advance to a state q. Then, the box formula $[\alpha_q]\psi$ with changed initial state of its program automaton is guaranteed to be in the Hintikka-set of this successor.

In order to define Hintikka-trees, the notion of infinite trees is made precise as follows. Further on, abbreviate the set $\{1, \dots, n\}$, $n \geq 0$, with $[n]$. Let M be a set of labels, and k a natural number. An *(infinite) k-ary M-tree* T is a mapping $T : [k]^* \to M$ where $[k]^*$ is the set of finite sequences of numbers $1, \dots, k$. Note that a k-ary tree is a tree where each node has k many successors, and that $x \in [k]^*$ uniquely determines a node in a k-ary tree: $x = \varepsilon$, where ε denotes the empty sequence, corresponds to the root of the tree, and xi, $i \in [k]$, denotes the i-th successor of node x. An infinite *path* in a k-ary M-tree is an infinite sequence $\gamma \in [k]^\omega$. Denote with $\gamma[n]$, $n \geq 0$, the prefix of γ up to the n-th element of the sequence ($\gamma[0]$ yields the empty sequence).

With the notion of an infinite tree in hand, a Hintikka-tree for an APDL$^{(\neg)}$-formula φ will be a k-ary Λ_φ-tree where k is the number of diamond formulas in the closure $cl(\varphi)$. Moreover, such a tree will have to meet further conditions, such as that the labels of adjacent nodes need to be matching.

As already mentioned, the matching relation is not sufficient to ensure diamond satisfaction. That is because a program automaton of a diamond formula might accept arbitrary long words such that it can become necessary to push this diamond several times to corresponding successors. Since the matching relation only considers nodes and their immediate successors, it cannot avoid that diamond formulas are pushed infinitely far along the tree and that they therefore never get satisfied. Now, the third component of a Λ_φ-triple comes into the play. These third triple components of the node labels make it possible to trace the path along which diamonds are pushed. Since diamond formulas change while being pushed, the Condition (M1)(b) indicates the respective next diamond formula in the third label component at the corresponding successor. Is the diamond satisfied at a node, (M1)(a) sets this third component to zero and a “diamond-push-path” ends. Consequently, to ensure that all diamonds eventually get satisfied, it remains to guarantee that there are no infinite such paths where diamonds can “starve”. These paths can be identified by the notion of diamond starvation.

Definition 3.3.3 (Diamond Starvation) Let $\varphi \in$ APDL$^{(\neg)}$ be a formula, and k

the number of diamond formulas in $cl(\varphi)$. Let T be a k-ary Λ_φ-tree, and $x \in [k]^*$ a node in T. A diamond formula $\epsilon_i = \langle\alpha\rangle\psi \in T(x)^1$ is *starving in x* if there is a path $\gamma = \gamma_1\gamma_2 \cdots \in [k]^\omega$ such that

(1) $\gamma_1 = i$, and

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

$\lhd$

Finally, Hintikka-trees are defined by putting together the notions of Hintikka-sets, matching relation, and diamond starvation.

Definition 3.3.4 (Hintikka-Tree) Let $\varphi \in \text{APDL}^{(\neg)}$ be a formula, and k the number of diamond formulas in $cl(\varphi)$. 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 four conditions:

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

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

(T3) no diamond formula of $cl(\varphi)$ is starving in node 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 $[\beta_{q'_\beta}]\theta \in T(y)^1$.

$\lhd$

Condition (T1) states that the formula φ, for which an abstract model representation should be constructed, must be contained in the Hintikka-set of the root of a Hintikka-tree for φ. The labels of adjacent nodes must be matching by Condition (T2). With (T3), infinite diamond-pushing is prevented by forcing that there is no diamond formula starving in any node of the tree. Condition (T4) captures the influence of negated atomic programs in the language. This is done analogous to [LS01], but generalized to program automata. By the semantics, any two worlds in an $\text{APDL}^{(\neg)}$-model are connected by the accessibility relation for either a or $\neg a$ where a is an atomic program. Consequently, any two nodes in a Hintikka-tree, being an abstract representation of such models, are connected by either a or $\neg a$ as well. For diamond formulas this makes no difference since at each node they already have their own successor. But, due to such global connections, box formulas can be pushed to nodes far away in the tree. This is modelled by the Conditions (T4) and (H3) as follows. Consider two nodes x and y in a Hintikka-tree T. Suppose there is a box formula $[\alpha]\psi$ in $T(x)^1$ such that $q'_\alpha \in \Delta_\alpha(q_\alpha, a)$ where a is an atomic program. By (H3), it holds that either $[\alpha_{q'_\alpha}]\psi$ or $\overline{[\alpha_{q'_\alpha}]\psi}$ is in $T(y)^1$. In the former case, the nodes x and y are connected by a, and in the latter one, by $\neg a$. Note that Condition (T4) is applicable in the second case and ensures that, if necessary, box formulas in $T(x)^1$ are pushed via a $\neg a$-connection to $T(y)^1$.

3.4 Satisfiability and Existence of Hintikka-Trees

In this section, it is proven that Hintikka-trees indeed properly represent APDL$^{(\neg)}$-models.

First, consider some auxiliary notions. For a given APDL$^{(\neg)}$-formula φ, denote with Σ_φ the set $\Pi_\varphi^{(\neg)}$ of all program literals in φ together with all tests occurring in φ. Let $\mathfrak{M}$ be a Kripke structure, and u a world in $\mathfrak{M}$. A word $w = w_1 \cdots w_m \in \Sigma_\varphi^*$, $m \geq 0$, *accomplishes* a diamond formula $\langle\alpha\rangle\varphi$ at a world u if $w \in \mathcal{L}(\alpha)$, and there are worlds $u_0, \ldots, u_m \in W$ with $u = u_0 R(w_1) \cdots R(w_m) u_m$, and $\mathfrak{M}, u_m \models \varphi$.

Lemma 3.4.1 *Let $\varphi \in APDL^{(\neg)}$ be a formula. Then φ is satisfiable iff φ has a Hintikka-tree.*

Proof. Let $\varphi \in$ APDL$^{(\neg)}$ be a formula and k the number of diamond formulas in $cl(\varphi)$.

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

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

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

In the following, a Hintikka-tree for φ is constructed. Define a k-ary W-tree T_W, and a k-ary $(2^{cl(\varphi)} \times \Pi_\varphi^{(\neg)} \cup \{\bot\} \times \{0,\dots,k\})$-tree T_φ by simultaneous induction such that it holds, for all $x \in [k]^*$

$$\psi \in T_\varphi(x)^1 \quad \text{iff} \quad \mathfrak{M}, T_W(x) \models \psi. \tag{$*$}$$

For the induction base, set

$$\begin{aligned} T_W(\varepsilon) &:= u_\varphi, \\ T_\varphi(\varepsilon)^1 &:= \{\psi \in cl(\varphi) \mid \mathfrak{M}, u_\varphi \models \psi\}, \\ T_\varphi(\varepsilon)^2 &:= \bot, \text{ and} \\ T_\varphi(\varepsilon)^3 &:= 0. \end{aligned}$$

For the induction step, let $x \in [k]^*$ be a node such that $T_W(x)$ is already defined. For each $i \in [k]$ consider the following cases:

(1) $\epsilon_i = \langle\alpha\rangle\psi \in T_\varphi(x)^1$. By $(*)$, it holds that $\mathfrak{M}, T_W(x) \models \langle\alpha\rangle\psi$. Then there is a word that accomplishes $\alpha\rangle\psi$ at $T_W(x)$ by the semantics. Among the words which accomplish $\langle\alpha\rangle\psi$ at $T_W(x)$ let $w = w_1 \cdots w_m \in \Sigma_\varphi^*$, $m \geq 0$, be the minimal one w.r.t. $\prec$. Then there are worlds $u_0, \dots, u_m \in W$ such that $T_W(x) = u_0 R(w_1) u_1 R(w_2) \cdots u_{m-1} R(w_m) u_m$ and $\mathfrak{M}, u_m \models \psi$. Distinguish two more cases:

   (a) The word w contains only tests or is empty. This implies that $T_W(x) = u$, and thus, $\mathfrak{M}, T_W(x) \models \psi$. Consequently, $\psi \in T_\varphi(x)^1$ by $(*)$. Define

$$\begin{aligned} T_W(xi) &:= u_\varphi, \\ T_\varphi(xi)^1 &:= \{\psi \in cl(\varphi) \mid \mathfrak{M}, u_\varphi \models \psi\}, \\ T_\varphi(xi)^2 &:= \bot, \text{ and} \\ T_\varphi(xi)^3 &:= 0. \end{aligned}$$

   (b) Otherwise, fix a run $q_0 \cdots q_m \in Q_\alpha^*$ of α on word w. Take the least $p \in [m]$ such that w_p is not a test but a program literal in $\Pi_\varphi^{(\neg)}$. Let

$j \in [k]$ be such that $\epsilon_j = \langle \alpha_{q_p} \rangle \psi$. Then, define

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

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

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

Claim 2 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 $\epsilon_i \in T_\varphi(x)^1$, $\epsilon_j \in T_\varphi(y)^1$, and $\ell(\epsilon_j, T_W(y)) < \ell(\epsilon_i, T_W(x))$.

Proof of Claim Let x and y be as in the claim. Suppose $T_\varphi(y)^3 = j \neq 0$. The value of $T_\varphi(y)^3$ can only be set to different from zero in the Case (1)(b) of the induction step. Thus, $\epsilon_i = \langle \alpha \rangle \psi \in T_\varphi(x)^1$. Let the word $w = w_1 \cdots w_m$, the worlds $u_0, \ldots, u_m$, the states $q_0 \cdots q_n$, and p be as in the Case (1)(b). Since w accomplishes $\langle \alpha \rangle \psi$ at $T_W(x)$, it holds that $w \in \mathcal{L}(\alpha)$ which implies $w_{p+1} \cdots w_m \in \mathcal{L}(\alpha_{q_p})$. Then from $T_W(y) = u_p R(w_{p+1}) u_{p+1} R(w_{p+2}) \cdots u_{m-1} R(w_m) u_m$ and $\mathfrak{M}, u_m \models \psi$, it follows by the semantics that $\mathfrak{M}, T_W(y) \models \langle \alpha_{q_p} \rangle \psi$ where $\epsilon_j = \langle \alpha_{q_p} \rangle \psi$. Consequently, $\epsilon_j \in T_\varphi(y)^1$ by $(*)$. Obviously, $w_{p+1} \cdots w_m$ accomplishes ϵ_j at $T_W(y)$. Since w accomplishes ϵ_i at $T_W(x)$ and is minimal w.r.t. $\prec$ with this property, it follows from the definition of the ordering $\prec$ that the rest word $w_{p+1} \cdots w_m$ is the minimal word with the property to accomplish ϵ_j at $T_W(y)$. Hence, $\ell(\epsilon_j, T_W(y)) = |w_{p+1} \cdots w_m| < m = |w| = \ell(\epsilon_i, T_W(x))$.

◀

The following proves 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. Next, consider the Condition (H4). Suppose that $[\alpha]\psi \in T_\varphi(x)^1$, and $q_\alpha \in F_\alpha$. Then, $\mathfrak{M}, T_W(x) \models \psi$ by $(*)$. It obviously holds that $(T_W(x), T_W(x)) \in R(\alpha)$, and

$\mathfrak{M}, T_W(x) \models \psi$. Hence, $\psi \in T_\varphi(x)^1$ by $(*)$. 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), $\overline{\theta} \in T_\varphi(x)^1$ or $\theta \in T_\varphi(x)^1$. In the former case, (H4) follows immediately. Consider the latter case. By $(*)$, it holds that $\mathfrak{M}, T_W(x) \models [\alpha]\psi$, and $\mathfrak{M}, T_W(x) \models \theta$. From the semantics, it follows that $\mathfrak{M}, T_W(x) \models [\alpha_q]\psi$. Thus, $[\alpha_q]\psi \in T_\varphi(x)^1$.

Since the first triple components of the node labels in tree T_φ are Hintikka-sets, T_φ can be seen as a 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. First consider the matching condition (M1). Suppose $\epsilon_i = \langle\alpha\rangle\psi \in T_\varphi(x)^1$. By $(*)$, it holds that $\mathfrak{M}, T_W(x) \models \langle\alpha\rangle\psi$. By the semantics, there is a word that accomplishes ϵ_i at $T_W(x)$. Let $w = w_1 \cdots w_m$ be the minimal such word w.r.t. $\prec$. To check Condition (M1)(a) suppose w contains only tests or is empty. Then Case (1)(a) of the induction step applies which yields $\psi \in T_\varphi(x)^1$, $T_\varphi(xi)^2 = \bot$, and $T_\varphi(x)^3 = 0$. Clearly, the program automaton α can reach a final state since $w \in \mathcal{L}(\alpha)$. Consequently, (M1)(a) is satisfied. For checking the Condition (M1)(b), assume that w contains a program literal. Let $p \in [m]$ be the smallest number such that w_p is a program literal. Then from Case (1)(b) of the induction step, it follows that $\epsilon_j = \langle\alpha_q\rangle\psi \in T_\varphi(xi)^1$, where $q \in \Delta_\alpha(q_\alpha, w_1 \cdots w_p)$ (as in the proof of Claim 2). Furthermore, $T_\varphi(xi)^2 = w_p$ and $T_\varphi(xi)^3 = j$. Thus, (M1)(b) is fulfilled.
Consider the remaining Condition (M2). Suppose $[\alpha]\psi \in T_\varphi(x)^1$. By $(*)$, it holds that $\mathfrak{M}, T_W(x) \models [\alpha]\psi$. Let $q \in Q_\alpha$ be a state of α, and $\pi \in \Sigma_\alpha$ be a program literal such that $q \in \Delta_\alpha(q_\alpha, \pi)$. Assume $\pi = T_\varphi(xi)^2$ for some $i \in [k]$. Since $T_\varphi(xi)^2 \neq \bot$, $T_\varphi(xi)^2$ and $T_W(xi)$ must have been set in Case (b) of the induction step. This implies $(T_W(x), T_W(xi)) \in R(\pi)$. Together with the fact that $\mathfrak{M}, T_W(x) \models [\alpha]\psi$, it follows that $\mathfrak{M}, T_W(xi) \models [\alpha_q]\psi$. Consequently, $[\alpha_q]\psi \in T_\varphi(xi)^1$.

(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

$\ell(\epsilon_{\gamma_1}, T_W(x))$, and the nodes $x\gamma[n]$, $n \geq 1$, with $\ell(\epsilon_{T_\varphi(x\gamma[n])^3}, T_W(x\gamma[n]))$. By the Claim 2, these numbers are strictly decreasing with increasing n along the path γ. Since the range of function ℓ is $\mathbb{N}$, there cannot be such a path γ; a contradiction.

(T4) It is to show that T_φ satisfies extended box-pushing w.r.t. node x. Suppose $[\alpha]\psi, [\beta]\theta \in T_\varphi(x)^1$. 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 $[\alpha_{q'_\alpha}]\psi \notin T(y)^1$. To show $(T_W(x), T_W(y)) \in R(\overline{\pi})$, assume by contradiction that $(T_W(x), T_W(y)) \in R(\pi)$. Since $\mathfrak{M}, T_W(x) \models [\alpha]\psi$ by $(*)$, the semantics yields, $\mathfrak{M}, T_W(y) \models [\alpha_{q'_\alpha}]\psi$. Then, by $(*)$, $[\alpha_{q'_\alpha}]\psi \in T(y)^1$; a contradiction. Consequently, $(T_W(x), T_W(y)) \in R(\overline{\pi})$. Since $\mathfrak{M}, T_W(x) \models [\beta]\theta$ by $(*)$, the semantics yields, $\mathfrak{M}, T_W(y) \models [\beta_{q'_\beta}]\theta$. Then, by $(*)$, it holds that $[\beta_{q'_\beta}]\theta \in T(y)^1$ which satisfies (T4).

" $\Leftarrow$": Suppose T is a Hintikka-tree for φ. Construct a Kripke Structure $\mathfrak{M} = (W, R, V)$ from tree T in the following way. The set of worlds W is defined as $W := [k]^*$, the set of nodes in a k-ary tree. For an atomic program $a \in \Pi_0$ define the accessibility relation $R(a) \subseteq W^2$ in terms of relations $R_\diamond(a)$ and $R_\Box(a)$ as $R(a) := R_\diamond(a) \cup R_\Box(a)$ where

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

The valuation function $V(p)$ for any propositional variable $p \in \Phi$ is defined as

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

In order to prove that $\mathfrak{M}$ is a model of φ, consider the following claim.

Claim 3 Let $\psi \in cl(\varphi)$ be a APDL$^{(\neg)}$-formula, and $x \in [k]^*$ a world in $\mathfrak{M}$. Then, $\psi \in T(x)^1$ implies $\mathfrak{M}, x \models \psi$.

Proof of Claim Let ψ and x be as in the claim. Define a *norm* $\| \cdot \|$ of

APDL$^{(\neg)}$-formulas in NNF inductively as follows:

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

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

- ψ is a propositional variable $p \in \Phi$. This is immediate by definition of $\mathfrak{M}$.
- $\psi = \neg p$. By assumption in section 3.1, ψ is in NNF, i.e., negation occurs only in front of propositional variables. Thus $p \in \Phi$. It follows from $\neg p \in T(x)^1$ by (H3) that $p \notin T(x)^1$. By definition of $\mathfrak{M}$, $x \notin V(p)$. Hence, $\mathfrak{M}, x \models \neg p$.

The induction step is as follows:

- $\psi = \psi_1 \vee \psi_2$ 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 by the induction hypothesis.
- $\psi = \epsilon_i = \langle\alpha\rangle\theta$. Inductively define a path $\gamma = \gamma_1\gamma_2 \cdots \in [k]^\omega$ as:

  - $\gamma_1 := i$
  - $\gamma_{n+1} := \begin{cases} T(x\gamma[n])^3 & \text{if } T(x\gamma[n])^3 \neq 0 \\ n' \in [k] \text{ arbitrary} & \text{otherwise} \end{cases}$.

  By (T3), the diamond formula $\langle\alpha\rangle\theta$ is not starving in x. Consequently, $T(x\gamma[n])^3 = 0$ for some $n \geq 1$. Let n be the smallest such number.
  From Condition (M1), it follows that there are states $q_0, \ldots, q_{n-1} \in Q_\alpha$, and a word $w = t_1\pi_1 \cdots t_{n-2}\pi_{n-2}t_{n-1} \in \Sigma_\alpha^*$, such that, for $m \in [n-2]$,

  (a) $\pi_m = T(x\gamma[m])^2$,

  (b) there are formulas $\psi_1, \ldots, \psi_\ell \in T(x\gamma[m-1])^1$, $\ell \geq 0$, such that $t_m = \psi_1? \cdots \psi_\ell?$ is a sequence of tests, for $m \in [n-1]$

  (c) $q_0 = q_\alpha$, $q_m \in \Delta_\alpha(q_{m-1}, t_m\pi_m)$, $q_{n-1} \in \Delta_\alpha(q_{n-2}, t_{n-1})$, and $q_{n-1} \in F_\alpha$

Moreover, $\theta \in T(x\gamma[n-1])^1$ by (M1). The induction hypothesis yields $\mathfrak{M}, x\gamma[n-1] \models \theta$. From (b), it follows by the induction hypothesis that, for each $m \in [n-1]$, $\mathfrak{M}, x\gamma[m-1] \models \psi_1, \ldots, \mathfrak{M}, x\gamma[m-1] \models \psi_\ell$.
In the following, it is shown that $(x\gamma[m-1], x\gamma[m]) \in R(\pi_m)$. Depending on π_m, distinguish two cases: First, if π_m is an atomic program a, then $(x\gamma[m-1], x\gamma[m]) \in R(a)$ holds easily by definition of $R(a)$. Second, suppose that π_m is an negated atomic program $\neg a$. Assume by contradiction that $(x\gamma[m-1], x\gamma[m]) \notin R(\neg a)$. Then, $(x\gamma[m-1], x\gamma[m]) \in R(a)$ by the semantics. By definition of $R_\diamond(a)$, $(x\gamma[m-1], x\gamma[m]) \notin R_\diamond(a)$ since $T(x\gamma[m])^2 = \neg a$; see (a). Consequently, $(x\gamma[m-1], x\gamma[m]) \in R_\Box(a)$ which yields that there is a $[\alpha]\psi \in T(x)^1$ and $q \in \Delta_\alpha(q_\alpha, \neg a)$ such that $[\alpha_q]\psi \notin T(y)^1$; a contradiction to (M2).
By (c) it holds that $w \in \mathcal{L}(\alpha)$. Thus, it follows by the semantics that $(x, x\gamma[n-1]) \in R(\alpha)$. Consequently, $\mathfrak{M}, x \models \langle\alpha\rangle\theta$.

- $\psi = [\alpha]\theta$. By the semantics, it needs to be shown that, for any world $y \in [k]^*$, $(x, y) \in R(\alpha)$ implies $\mathfrak{M}, y \models \theta$. Suppose $(x, y) \in R(\alpha)$ for some world $y \in [k]^*$, i.e., there is a word $w = w_1 \cdots w_m \in \mathcal{L}(\alpha)$, $m \geq 0$, and worlds $x_0, \ldots, x_m \in [k]^*$ with $x = x_0 R(w_1) x_1 R(w_2) \cdots x_{m-1} R(w_m) x_m = y$. Fix a run $q_0 \cdots q_m \in Q_\alpha^*$ of program automaton α on w. In the following, it is shown by induction on i that $[\alpha_{q_i}]\theta \in T(x_i)^1$ for $i \leq m$. The induction start is immediate. For the induction step, assume that $[\alpha_{q_i}]\theta \in T(x_i)^1$. Depending on w_i distinguish the following three cases:

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

  (2) 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 the 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 $[\alpha_{q_{i+1}}]\theta \in$

$T(x_{i+1})^1$.

(3) w_i is a negated atomic program $\neg a$ in $\Pi_\varphi^{(\neg)}$. It follows from the semantics that $R(\neg a) = W^2 \backslash R(a)$, and by definition of $R(a)$, $(x_i, x_{i+1}) \in W^2 \backslash (R_\Diamond(\neg a) \cup R_\Box(\neg a))$. By definition of $R_\Box$, the fact that $(x_i, x_{i+1}) \notin R_\Box(a)$ implies that, for any box formula $[\beta]\delta$ and state $q \in Q_\beta$ such that $q \in \Delta_\beta(q_\beta, \overline{a})$, $[\beta]\delta \in T(x_i)^1$ implies $[\beta_q]\delta \in T(x_{i+1})^1$. Then, since $[\alpha_{q_i}]\theta \in T(x_i)^1$ by assumption, $[\alpha_{q_{i+1}}]\theta \in T(x_{i+1})^1$.

Consequently, it holds that $[\alpha_{q_m}]\theta \in T(x)^1$. Since $q_m \in F_\alpha$, Condition (H4) yields $\theta \in T(x)^1$.

◀

By Condition (T1), it holds that $\varphi \in T(\varepsilon)^1$. Then it follows directly from Claim 3 that $\mathfrak{M}$ is a model of φ. QED

Chapter 4

Büchi Automata decide Satisfiability of PDL$^{(\neg)}$

In the previous chapter, an one-to-one correspondence between APDL$^{(\neg)}$-models and Hintikka-trees has been established. In the following, an elegant technique is shown how to check for existence of Hintikka-trees by Büchi tree automata, and thus, to solve the satisfiability problem of APDL$^{(\neg)}$. More precisely, for a given APDL$^{(\neg)}$-formula φ a Büchi tree automaton $\mathcal{B}_\varphi$ is constructed such that it accepts precisely the Hintikka-trees for φ. Then, checking for emptiness of $\mathcal{B}_\varphi$'s language corresponds to decide satisfiability of φ. Subsequently, complexity issues are discussed yielding that satisfiability of APDL$^{(\neg)}$, and thus of PDL$^{(\neg)}$, is ExpTime-complete.

4.1 Büchi Tree Automata

In this section, Büchi tree automata are introduced which will be employed for devising a decision procedure for APDL$^{(\neg)}$.

Definition 4.1.1 (Büchi Tree Automaton, Run, Accepting) Let k be a natural number. A *Büchi Tree Automaton $\mathcal{B}$ for k-ary trees* is defined as $\mathcal{B} := (Q, M, I, \Delta, F)$ where Q is a finite set of states, M an alphabet, $I \subseteq Q$ the set of initial states, $\Delta \subseteq Q \times M \times Q^k$ a transition relation, and $F \subseteq Q$ is a set of accepting states.
Let M be a set of labels, and T be 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 $\mathit{inf}_r(\gamma)$ of the states in Q that occur infinitely often in run r along path γ is defined as: $\mathit{inf}_r(\gamma) := \{q \in Q \mid \forall n.\exists m.r(\gamma[n]) = q \text{ implies } r(\gamma[m]) = q\}$.
A run r of $\mathcal{B}$ on T is *accepting* if, for each path $\gamma \in [k]^\omega$, it holds that $\mathit{inf}_r(\gamma) \cap F \neq \emptyset$. The *language* accepted by $\mathcal{B}$ is the set $\mathcal{L}(\mathcal{B}) = \{T \mid$ there is an accepting run of $\mathcal{B}$ on $T\}$.

$\lhd$

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*. The emptiness problem for Büchi automata is solvable in quadratic time in the size of the automata; see [VW86].

4.2 Automata Construction

This section provides the construction of the Büchi tree automaton $\mathcal{B}_\varphi$ for an APDL$^{(\neg)}$-formula φ. This construction is aiming at that $\mathcal{B}_\varphi$ accepts exactly the Hintikka-trees for φ.

Denote with $\mathcal{P}_\Box(\varphi)$ the set of sets of box formulas $\{\{[\alpha]\psi, [\beta]\theta\} \mid [\alpha]\psi, [\beta]\theta \in cl(\varphi)\}$.

Definition 4.2.1 Let $\varphi \in$ APDL$^{(\neg)}$ be a formula, and k the number of diamond formulas in $cl(\varphi)$. Then the Büchi tree automaton for k-ary trees $\mathcal{B}_\varphi$ is defined as follows: $\mathcal{B}_\varphi := (Q, \Lambda_\varphi, I, \Delta, F)$ such that

- The set Q of states is $Q \subseteq \Lambda_\varphi \times 2^{\mathcal{P}_\Box(\varphi)} \times \{\oslash, \uparrow\}$ such that each state $((\Psi, \pi, \ell), P, d) \in Q$ satisfies the following two conditions:
  (1) if $[\alpha]\psi, [\beta]\theta \in \Psi$, 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 $[\beta_{q'_\beta}]\theta \in \Psi$
- The set I of initial states is $I := \{((\Psi, \pi, \ell), P, d) \in Q \mid \varphi \in \Psi, \text{ and } d = \oslash\}$.

- The transitions in $\Delta \subseteq Q \times \Lambda_\varphi \times Q^k$ are defined as follows: $((\lambda_0, P_0, d_0), (\Psi, \pi, \ell), (\lambda_1, P_1, d_1), \dots, (\lambda_k, P_k, d_k)) \in \Delta$ iff, for all $i \in [k]$, the following holds:

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

  (2) $P_0 = P_i$,

  (3) the $k+1$-tuple $(\lambda_0, \dots, \lambda_k)$ is matching, and

  (4) $d_i = \begin{cases} \uparrow & \text{if } d_0 = \oslash,\ \lambda_i^3 \neq 0, \text{ and } \epsilon_i \in \Psi_0 \\ \uparrow & \text{if } d_0 = \uparrow,\ \lambda_0^3 = i, \text{ and } \lambda_i^3 \neq 0 \\ \oslash & \text{otherwise} \end{cases}$.

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

$\triangleleft$

In order to achieve that the Büchi tree automaton $\mathcal{B}_\varphi$ accepts precisely the Hintikka-trees for φ, the definition of $\mathcal{B}_\varphi$ needs to incorporate the conditions for Hintikka-trees (T1) to (T4). In the following, it is explained how this is done.

Automaton $\mathcal{B}_\varphi$ is a Büchi tree automaton for k-ary trees, where k is the number of diamond formulas in the closure $cl(\varphi)$. Since Λ_φ is used as alphabet for $\mathcal{B}_\varphi$, $\mathcal{B}_\varphi$ accepts k-ary Λ_φ-trees. In order to see that these trees in the language $\mathcal{L}(\mathcal{B}_\varphi)$ satisfy the Conditions (T1) to (T4), look at the accepting runs of $\mathcal{B}_\varphi$. By definition of a run, the root of a run is labelled with an initial state in I. Such an initial state contains φ in its Hintikka-set. Then, by Condition (1) in the definition of Δ, the Hintikka-set in the root of an accepted tree also contains φ, which captures (T1). The node labels of a node and its immediate successors in an accepted tree are matching by Condition (1) and (2) in the definition of the transition relation Δ. Thus (T2) is satisfied.

The incorporation of Condition (T3) is more intrigue; it involves the third components of states and the accepting condition for a run. Intuitively, whenever a diamond formula is not satisfied at a current node in a Hintikka-tree, it is pushed further to the next node, where it may be satisfied. The paths along which diamonds are pushed should not be infinite, otherwise the diamond is never satisfied—it "starves". Therefore, Condition (4) in the definition of Δ marks nodes of a run, which are on a "diamond-push-path", with $\uparrow$ in the third component of states, and the other nodes with $\oslash$. To trace and mark such paths the information in the third component of Λ_φ-triples is used. This component is set by (M1)(b) indicating the next node in the path, or by (M1)(a) to zero when the path ends. Infinite "diamond-push-paths" are avoided using the acceptance condition for a run. This is done by defining the third component of all accepting states in F to be $\oslash$. Consequently, no path in an accepting run contains infinitely many consecutive nodes marked with $\upharpoonleft$. Then, by Condition (1) in the definition of Δ, no diamond is starving in a tree accepted by $\mathcal{B}_\varphi$ which captures (T3).

In the following, the labelling of states in a run with $\upharpoonleft$ and $\oslash$ controlled by Point (4) in the definition of Δ is illustrated in more detail. Suppose there is an unsatisfied diamond formula $\epsilon_i = \langle\alpha\rangle\psi$ at a $\oslash$-labelled node x in a run; see Figure 4.1. Then (M1)(b) pushes $\langle\alpha\rangle\psi$ to xi yielding $\epsilon_j = \langle\alpha_q\rangle\psi$. When ϵ_j is also not satisfied at xi, it is pushed further to xij, and so on. The nodes xi, $xij, \ldots$ on the path along which the diamond $\langle\alpha\rangle\psi$ is pushed are marked with $\upharpoonleft$. Now suppose that there is another unsatisfied diamond formula $\epsilon_k = \langle\beta\rangle\theta$, $k \neq j$, at xi. Clearly, $\langle\beta\rangle\theta$ gets pushed to the node xik. Thus, another "$\upharpoonleft$-path" starts out of an already existing one. Observe that the first node xik in the "$\upharpoonleft$-path" of $\langle\beta\rangle\theta$ is marked with $\oslash$ in order to separate the two "$\upharpoonleft$-paths". Otherwise, if two such paths are not separated by a node marked with $\oslash$, this could yield an infinite "$\upharpoonleft$-path" by jumping from one "$\upharpoonleft$-path" to the next along r. Consequently, run r would not be accepting. In this way, Hintikka-trees satisfying (T3) could be rejected by $\mathcal{B}_\varphi$.

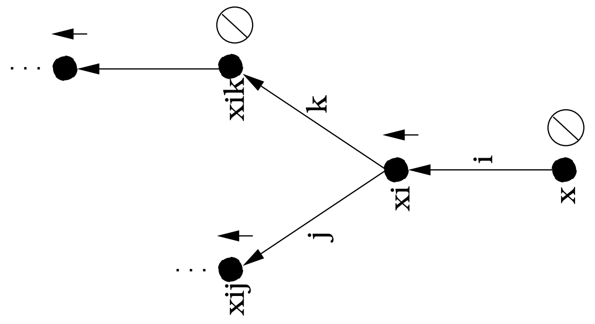

Figure 4.1: Run of $\mathcal{B}_\varphi$ where nodes are labelled with $\upharpoonleft$ and $\oslash$

Finally, consider how Condition (T4) is modelled by using the second component of states. Remember that (T4) has a global character since it relates any two nodes in a Hintikka-tree with each other. For this, the second component of a state is used as a set of pairs of box formulas. By Point (1) in the definition of Q and (1) in the definition of Δ, such a set contains each pair of box formulas at a node in a tree accepted by $\mathcal{B}_\varphi$. Point (2) in the definition of Δ ensures that every node in a run is labelled with the same such set. Thus, the second component of a state in a run makes any pair of box formulas occurring in this run available for each node. Then, Condition (T4) is realized in an tree accepted by $\mathcal{B}_\varphi$ with the help of Point (2) in the definition of Q and (1) in the definition of Δ.

In preparation for a subsequent discussion about the complexity of APDL$^{(\neg)}$'s decision procedure, the following claim determines the size of the Büchi tree automaton $\mathcal{B}_\varphi$. Preceding this, assume that $\mathcal{B}_\varphi = (Q, \Lambda_\varphi, I, \Delta, F)$ is encoded by just separately writing down its components as sets element by element.

Claim 4 The size of $\mathcal{B}_\varphi$ is exponential in the length of φ.

Proof of Claim The size of the Büchi automaton $\mathcal{B}_\varphi = (Q, \Lambda_\varphi, I, \Delta, F)$ is the length of its coding. By the assumption above, the sum $|\mathcal{B}_\varphi| = |Q| + |\Lambda_\varphi| + |I| + |\Delta| + |F|$ of the cardinality of $\mathcal{B}_\varphi$'s components is polynomial in the length of this coding.

Denote with $n = |cl(\varphi)| + |\Pi_\varphi^{(\neg)}|$ an upper bound for the size of the closure $cl(\varphi)$, and the size of the set $\Pi_\varphi^{(\neg)}$ of program literals in φ. Let k denote the number of diamond formulas in $cl(\varphi)$. Then n is also an upper bound for k.

In the following, it is determined that $n = |cl(\varphi)| + |\Pi_\varphi^{(\neg)}|$ is polynomial in the length of φ. This amounts to show that the cardinality of the sets $cl(\varphi)$ and $\Pi_\varphi^{(\neg)}$ is polynomial in the length of φ. The case for $cl(\varphi)$ simply holds by Claim 1. Consider the set $\Pi_\varphi^{(\neg)}$. By assumption in Section 3.2, any program automaton α in φ is represented in such a way that the number of program literals in Σ_α is polynomial in the size of α. Then the size of $\Pi_\varphi^{(\neg)}$, which contains all program literals in φ, is polynomial in the length of φ.

Consider the upper bounds for the cardinality of the following sets:

$$\begin{array}{lcll}
\mathcal{P}_\Box \subseteq cl(\varphi) \times cl(\varphi) & \rightsquigarrow & |\mathcal{P}_\Box| & \leq n^2 \\
\mathcal{H}_\varphi \subseteq 2^{cl(\varphi)} & \rightsquigarrow & |\mathcal{H}_\varphi| & \leq 2^n \\
\Lambda_\varphi = \mathcal{H}_\varphi \times \Pi_\varphi^{(\neg)} \cup \{\bot\} \times \{0,\dots,k\} & \rightsquigarrow & |\Lambda_\varphi| & = |\mathcal{H}_\varphi| \cdot (|\Pi_\varphi^{(\neg)}| + 1) \cdot (k+1) \\
 & & & \leq 2^{\mathcal{O}(n)} \cdot \mathcal{O}(n^2) \\
Q \subseteq \Lambda_\varphi \times 2^{\mathcal{P}_\Box} \times \{\oslash, \uparrow\} & \rightsquigarrow & |Q| & \leq |\Lambda_\varphi| \cdot 2^{|\mathcal{P}_\Box|} \cdot 2 \\
 & & & \leq 2^{\mathcal{O}(n^2)} \cdot \mathcal{O}(n^2) \\
I \subseteq Q & \rightsquigarrow & |I| & \leq |Q| \\
\Delta \subseteq Q \times \Lambda_\varphi \times Q^k & \rightsquigarrow & |\Delta| & \leq |Q| \cdot |\Lambda_\varphi| \cdot |Q|^n \\
 & & & \leq 2^{\mathcal{O}(n^3)} \cdot \mathcal{O}(n)^{\mathcal{O}(n)} \\
F \subseteq Q & \rightsquigarrow & |F| & \leq |Q|
\end{array}$$

Note that the set Δ has the highest upper bound. Then, $|\mathcal{B}_\varphi|$ is determined as:

$$\begin{aligned}
|\mathcal{B}_\varphi| &= |Q| + |\Lambda_\varphi| + |I| + |\Delta| + |F| \\
&= 2^{\mathcal{O}(n^3)} \cdot \mathcal{O}(n)^{\mathcal{O}(n)}
\end{aligned}$$

Consequently, the size of automaton $\mathcal{B}_\varphi$ is exponential in the length of φ. ◀

4.3 Recognition of Hintikka-Trees by Büchi Tree Automata

The following lemma shows that the Büchi tree automaton $\mathcal{B}_\varphi$ indeed accepts precisely the Hintikka-trees for φ.

Lemma 4.3.1 *Let $\varphi \in APDL^{(\neg)}$ be a formula, and T a k-ary Λ-tree. Then T is a Hintikka-tree for φ iff $T \in \mathcal{L}(\mathcal{B}_\varphi)$.*

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

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

Define the set P of pairs of box formulas in T as

$$P_{\Box}(T) := \{\{[\alpha]\psi, [\beta]\theta\} \mid [\alpha]\psi, [\beta]\theta \in T(x)^1 \text{ for some } x \in [k]^*\}.$$

Define a $\Lambda_\varphi \times \{P_\Box(T)\} \times \{\oslash, \uparrow\}$-tree r by induction on the sequence $x \in [k]^*$ as follows. In the induction base, set $r(\varepsilon) := (T(\varepsilon), P_\Box(T), \oslash)$. For the induction step, let $x \in [k]$ be such that $r(x) = (T(x), P_\Box(T), d_x)$ is already defined. For all $i \in [k]$, define $r(xi) := (T(xi), P_\Box(T), d_{xi})$ where

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

Claim 5 The tree r is an accepting run of $\mathcal{B}_\varphi$ on T.

Proof of Claim In order to verify that r is a run of $\mathcal{B}_\varphi$ on T, it is first checked that r is a Q-tree, i.e. that $r(x) \in Q$ for all nodes $x \in [k]^*$. Let $x \in [k]^*$ be a node. Since T is a Hintikka-tree for φ, it holds that $T(x) \in \Lambda_\varphi$. Moreover, $P_\Box(T) \subseteq \mathcal{P}_\Box(\varphi)$, and thus $r(x) = (T(x), P_\Box(T), d_x) \in \Lambda_\varphi \times 2^{\mathcal{P}_\Box(\varphi)} \times \{\oslash, \uparrow\}$. It still needs to be shown that $r(x)$ satisfies Properties (1) and (2) of the definition of Q.

(1) Holds by definition of set $P_\Box(T)$.

(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 two conditions in Definition 4.1.1 need to be checked.
First, $r(\varepsilon) \in I$. Since T is a Hintikka-tree for φ, Condition (T1) yields that $\varphi \in T(\varepsilon)^1$. Hence, $r(\varepsilon) = (T(\varepsilon), P_\Box(T), \oslash) \in I$.
Second, it is to check that $(r(x), T(x), r(x1), \ldots, r(xk)) \in \Delta$ 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 immediate by definition of r. Condition (3) holds since the tuple $(T(x), T(x1), \ldots, T(xk))$ is matching by (T2). Last, the Condition (4) is easily be checked using the definition of r.

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 $inf_r(\gamma)$ only contains states $r(x) = (T(x), P, d_x)$ where the third component is $d_x = \uparrow$. Consequently, there is a position p in γ such that, for all $m \geq p$, the third component of $r(\gamma[m])$ is $d_{\gamma[m]} = \uparrow$. Note that $p > 0$ since $d_\varepsilon = \oslash$ by definition. Let p be the minimal such position, i.e., $d_{\gamma[p]} = \uparrow$, and $d_{\gamma[p-1]} = \oslash$. Then from the definition of $d_{\gamma[p]}$, it follows that $\epsilon_{\gamma_p} \in T(\gamma[p-1])^1$.
In order to show that the diamond formula ϵ_{γ_p} is starving in $\gamma[p-1]$, check that the path $\gamma_p\gamma_{p+1} \cdots$ has the Properties (1) and (2) of Definition 3.3.3. Property (1) is immediate since $\epsilon_{\gamma_p} \in T(\gamma[p-1])^1$. With $d_{\gamma[m]} = \uparrow$ and $d_{\gamma[m+1]} = \uparrow$ for any $m \geq p$, the definition of $d_{\gamma[m+1]}$ yields $\gamma_{m+1} = T(x\gamma[m])^3$ which shows Property (2).
The fact that ϵ_{γ_p} is starving in $\gamma[p-1]$ is a contradiction to (T3) which is satisfied by Hintikka-tree T. ◀

The result $T \in \mathcal{L}(\mathcal{B}_\varphi)$ follows directly follows from Claim 5 above.

" $\Leftarrow$": Suppose $T \in \mathcal{L}(\mathcal{B}_\varphi)$, i.e. that there is an accepting run, say r, of $\mathcal{B}_\varphi$ on T. In the following, it is shown that T is a Hintikka-tree for φ. Since $\mathcal{B}_\varphi$ is a Büchi tree automaton for k-ary trees with alphabet Λ_φ, T is a k-ary Λ_φ-tree. Let $x \in [k]^*$ be a node such that $r(x) = (T_x, P_x, d'_x)$. In the following, it is checked that T fulfills the conditions for Hintikka-trees (T1) to (T4).

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

(T2) By Condition (2) in the definition of a run, $(r(x), T(x), r(x1), \ldots, r(xk)) \in \Delta$. Then it follows by (3) in the definition of Δ that the $k+1$-tuple $(T(x), T(x1), \ldots, T(xk))$ is matching.

(T3) Suppose by contradiction that there is a diamond formula $\epsilon_i \in T(x)^1$ that is starving in x. Then there exists a path $\gamma = \gamma_1\gamma_2 \cdots \in [k]^\omega$ such that $\gamma_1 = i$, and $\gamma_{n+1} = T(x\gamma[n])^3$ for $n \geq 1$. In the following, it is shown that $d_{x\gamma[n]} = \uparrow$ for $n \geq 2$. This is done by induction on n. In the induction base, $n = 2$. Distinguish two cases:

– $d'_{x\gamma[1]} = \uparrow$. By definition of diamond starvation it holds that $T(x\gamma[1])^3 = \gamma_2$ and $T(x\gamma[2])^3 = \gamma_3 \neq 0$. Hence, $d'_{x\gamma[2]} = \uparrow$ by Condition (4) in the definition of Δ.

- $d'_{x\gamma[1]} = \oslash$. By definition of diamond starvation it holds that $T(x\gamma[1])^3 = \gamma_2 \neq 0$ and $T(x\gamma[2])^3 = \gamma_3 \neq 0$. As shown above, (T2) holds for all nodes $x \in [k]^*$. Thus (M1)(b) applies at x since $\epsilon_i \in T(x)^1$, $i = \gamma_1$, and $T(x\gamma[1])^3 = \gamma_2 \neq 0$, and yields $\epsilon_j \in T(x\gamma[1])^1$ where $j = \gamma_2$. Then, Condition (4) in the definition of Δ yields $d'_{x\gamma[2]} = \uparrow$.

For the induction step, let $n > 2$ such that $d_{x\gamma[n]} = \uparrow$ is already defined. By definition of diamond starvation it holds that $T(x\gamma[n])^3 = \gamma_{n+1}$ and $T(x\gamma[n+1])^3 = \gamma_{n+2} \neq 0$. Then, by Condition (4) in the definition of Δ it follows $d'_{x\gamma[n+1]} = \uparrow$.
Consequently, there are only finitely many nodes in the path $x\gamma$ which are labelled with $\oslash$. This implies that $inf_r(x\gamma) \cap F = \emptyset$; a contradiction to r being an accepting run.

(T4) Suppose $[\alpha]\psi, [\beta]\theta \in T(x)^1$. Let $\pi \in \Pi_0^{(\neg)}$ be a program literal and $q'_\alpha \in Q_\alpha$, $q'_\beta \in Q_\beta$ be 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 $y \in [k]^*$ with $r(y) = (T_y, P_y, d'_y)$. By Condition (1) in the definition of Q, it holds that $\{[\alpha]\psi, [\beta]\theta\} \in P_x$. Since $P_x = P_y$ by (2) in the definition of Δ, Condition (2) in the definition of Q yields $[\beta_{q'_\beta}]\theta \in T(y)^1$.

QED

4.4 Complexity of PDL$^{(\neg)}$-Satisfiability

Theorem 4.4.1 *Satisfiability problem for PDL$^{(\neg)}$ is* ExpTime*-complete.*

Proof. From Lemma 3.4.1 and 4.3.1, it follows that, for all APDL$^{(\neg)}$-formulas φ, φ is satisfiable if and only if $\mathcal{L}(\mathcal{B}_\varphi) \neq \emptyset$. The emptiness problem for Büchi automata is decidable in quadratic time in the size of the automaton; see [VW86]. By Claim 4, the size of the automaton $\mathcal{B}_\varphi$ is exponential in the length of φ. Hence, satisfiability of APDL$^{(\neg)}$ can be decided in ExpTime. Since formulas of PDL$^{(\neg)}$ can be translated into APDL$^{(\neg)}$-formulas in linear time, satisfiability of PDL$^{(\neg)}$ is in ExpTime which fixes an upper bound. For the lower bound consider the fragment PDL of PDL$^{(\neg)}$ for which satisfiability is ExpTime-hard; see [FL79]. Thus the satisfiability problem for PDL$^{(\neg)}$ is ExpTime-complete. QED

Chapter 5

Conclusion

In this work, propositional dynamic logic is extended with negation on atomic programs resulting in the logic $\mathrm{PDL}^{(\neg)}$. The satisfiability problem for $\mathrm{PDL}^{(\neg)}$ was shown to be decidable and ExpTime-complete. Thus adding negation of atomic programs to PDL does not make reasoning more difficult, but increases PDL's expressiveness in an interesting and useful way. To obtain the complexity result, an ExpTime-decision procedure was constructed using Büchi automata on infinite trees. This technique extends standard automata-based decision procedures for PDL [VW86] and for Boolean modal logic [LS01]. Due to the correspondence of DLs and modal logics, the complexity result for $\mathrm{PDL}^{(\neg)}$ can be transferred to its corresponding DL $\mathcal{ALC}^{(\neg)}_{\mathrm{reg}}$. Hence, as a corollary it can be stated that satisfiability of $\mathcal{ALC}^{(\neg)}_{\mathrm{reg}}$-concepts is ExpTime-complete.

For future work, it seems interesting to study the complexity of further extensions of PDL. For instance, it should not be hard to show that adding the converse operator to $\mathrm{PDL}^{(\neg)}$ does not increase its complexity. In order to gain more of $\mathrm{PDL}^{\neg}$'s appealing expressiveness without destroying decidability, it is interesting to further investigate logics which allow for more negation on programs. From a technical viewpoint, this means identifying larger decidable fragments of the undecidable logic $\mathrm{PDL}^{\neg}$. A next interesting candidate for this could be the fragment of $\mathrm{PDL}^{\neg}$ that has only the program operators negation "$\neg$" and composition ";". Currently, it is supposed to be unknown whether this logic is decidable.

Bibliography

[BCM$^+$03] F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook - Theory, Implementation and Applications.* Cambridge University Press, 2003.

[Dan84] R. Danecki. Nondeterministic propositional dynamic logic with intersection is decidable. In *Proc. Workshop on Logics of Programs 1981*, volume 208 of *Lecture Notes in Computer Science*, pages 34–53. Spring-Verlag, 1984.

[DL95] G. De Giacomo and M. Lenzerini. PDL-based framework for reasoning about actions. In *Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence (AI*IA'95)*, volume 992, pages 103–114. Springer, 1995.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, April 1979.

[GGLM94] De Giacomo, Giuseppe, Lenzerini, and Maurizio. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 205–212. AAAI-Press/the MIT-Press, 1994.

[Gor90] Valentin Goranko. Modal definability in enriched languages. *Notre Dame Journal of Formal Logic*, 31(1):81–105, 1990.

[GP92] Valentin Goranko and Solomon Passy. Using the universal modality: Gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992.

[GPT87] G. Gargov, S. Passy, and T. Tinchev. Modal environment for Boolean speculations. In D. Skordev, editor, *Mathematical Logic and Applications*, pages 253–263, New York, USA, 1987. Plenum Press.

[Har84] David Harel. Dynamic logic. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic, Volume II*, pages 496–604. D. Reidel Publishers, 1984.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.

[HP78] D. Harel and V.R. Pratt. Nondeterminism in logics of programs. In *Proceedings of the Fifth Symposium on Principles of Programming Languages*, pages 203–213. ACM, 1978.

[Hum83] I.L. Humberstone. Inaccessible worlds. In *Notre Dame Journal of Formal Logic*, pages 24(3):346–352, 1983.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. Automata Studies, pages 3–41. Princeton University Press, 1956.

[LS00] C. Lutz and U. Sattler. Mary likes all cats. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS, pages 213–226, Aachen, Germany, August 2000. RWTH Aachen. Proceedings online available from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-33/.

[LS01] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In Frank Wolter, Heinrich Wansing, Maarten de Rijke, and Michael Zakharyaschev, editors, *Advances in Modal Logics Volume 3*. CSLI Publications, Stanford, 2001.

[Pos47] E. L. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[Pra79] V. R. Pratt. Models of program logics. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1979.

[PS96] Helmut Prendinger and Gerhard Schurz. Reasoning about action and change: A dynamic logic approach. *Journal of Logic, Language, and Information*, 5(2):209–245, 1996.

[PT91] Solomon Passy and Tinko Tinchev. An essay in combinatory dynamic logic. *Inf. Comput.*, 93(2):263–332, 1991.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sydney, AU, 1991.

[Sch94] Klaus Schild. Tractable reasoning in a universal description logic. In *Proc. of the 1st Workshop on Reasoning about Structured Objects: Knowledge Representation Meets Databases*, Saarbrcken, Germany, 1994.

[Var85] M. Y. Vardi. The taming of converse: Reasoning about two-way computations. In Rohit Parikh, editor, *Proceedings of the Conference on Logic of Programs*, volume 193 of *LNCS*, pages 413–424. Springer, 1985.

[VW86] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logic of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.