A Goal-Directed Decision Procedure for Hybrid PDL

Mark Kaminski and Gert Smolka

Saarland University

October 30, 2013

We present the first goal-directed decision procedure for hybrid PDL. The procedure is based on a modular approach that scales from basic modal logic with eventualities to hybrid PDL. The approach is designed so that nominals and eventualities are treated orthogonally. To deal with the complex programs of PDL, the approach employs a novel disjunctive program decomposition. In arguing the correctness of our approach, we employ the novel notion of support generalizing the standard notion of Hintikka sets.

1 Introduction

Propositional dynamic logic (PDL) [19, 23] is an expressive but decidable modal logic initially introduced for reasoning about program correctness. It extends basic modal logic with expressions called programs. Programs describe binary relations on states, and are used to define modalities. Complex programs are constructed from atomic programs with union, composition, and iteration (i.e., complex programs include regular expressions over atomic programs). Additionally, there are special programs called tests that allow to restrict the domain or range of a relation to states satisfying a certain formula. A particularly important class of formulas in PDL are diamond formulas of the form $\langle \alpha^* \rangle \varphi$, which hold for a given state if they can reach a state satisfying the formula φ via a finite iteration of the program α. Following a tradition in temporal logics [12, 14], we call such formulas eventualities.

Due to the inductive nature of eventualities, PDL is not compact (consider $\langle a^* \rangle \neg p$, $[a]p$, $[a][a]p, \ldots$). Still, the satisfiability problem for PDL is decidable and EXPTIME-complete. Fischer and Ladner [19] establish the decidability of PDL with a small model theorem, which states that every formula has a syntactic

model obtained over a finite formula universe known as the Fischer-Ladner closure. They also show that the satisfiability problem for PDL is EXPTIME-hard.

Based on Fischer and Ladner's model construction, Pratt [46] gives a decision procedure for PDL that runs in deterministic exponential time, thus establishing EXPTIME-completeness of the logic (see [23, 6, 32] for variants of Pratt's procedure with correctness proofs). The procedure starts with the set $\mathcal{H}$ of all Hintikka sets over the Fischer-Ladner closure of the input formula and prunes $\mathcal{H}$ by removing unsatisfiable sets until the remaining sets form a model satisfying all of the sets. The input formula is then satisfiable if and only if it is contained in one of the remaining sets. Pratt [47] devises a more practical version of the abstract procedure in [46] that works with an AND-OR graph constructed from the input formula. While initially developed for PDL, Pratt's methods scale to temporal logics as shown by Ben-Ari et al. [5], Wolper [55], and Emerson and Halpern [15]. An alternative formulation of Pratt's procedure [47] is given by Nguyen and Szałas [44].

Although worst-case optimal, Pratt's methods are not truly practical since they always construct data structures whose size is exponential in that of the input formula, and hence take exponential time. The first decision procedure for PDL that is both practical and worst-case optimal is devised by Goré and Widmann [20, 54]. Goré and Widmann's procedure interleaves the construction of the AND-OR graph with pruning. Unlike Pratt's procedures, it is goal-directed. By following the structure of the input formula, the procedure avoids computation steps that are redundant for determining the satisfiability of the formula. This way, many formulas can be decided efficiently despite the high worst-case complexity of the logic.

A different family of decision procedures for PDL is based on nondeterministic search for models as pioneered for modal logic by Kripke [40]. Such procedures stepwise grow (a syntactic representation of) a candidate model by adding states and transitions until the model satisfies the input formula. The first nondeterministic procedure for (a variant of) test-free PDL is due to Baader [4]. To check the satisfaction of eventualities, Baader develops a dedicated loop detection technique. Later, related procedures are also proposed for full PDL by De Giacomo and Massacci [13], and by Abate et al. [1]. Unlike Goré and Widmann's procedure, the procedures in [4, 13, 1] have suboptimal complexity: Baader's [4] procedure can be seen to run in NEXPTIME, the procedure by Abate et al. [1] requires 2EXPTIME, while de Giacomo and Massacci [13] claim NEXPTIME complexity for their procedure (without proof). Despite being suboptimal, the procedures improve on Pratt's procedures in terms of practical efficiency since they are goal-directed.

In this paper, we devise the first goal-directed procedure for PDL extended

with nominals. Nominals are predicates that hold for exactly one state. They allow to reason about equality of states in a model and are the characteristic feature of hybrid logic [3]. We call the extension of PDL with nominals hybrid PDL (HPDL). Similarly to ordinary PDL, HPDL is EXPTIME-complete [49, 32].

The present procedure builds on an earlier procedure for a simpler hybrid logic with eventualities [34] and thoroughly revises the procedure in [35]. A variant of the procedure is studied by the first author in his PhD thesis [31].

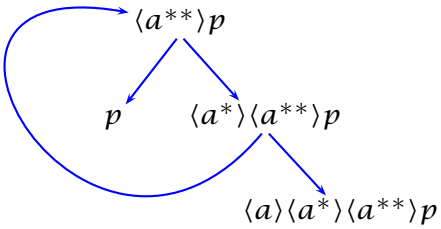
Our procedure is based on nondeterministic search (similarly to [4, 13, 1]) rather than pruning as used by Pratt [46, 47] and Goré and Widmann [20, 54]. The reason for choosing nondeterministic search is that pruning crucially depends on certain closure properties for models that do not hold in the presence of nominals [31] (the problem of extending Goré and Widmann's approach to nominals is also noted in [54]). On the other hand, nondeterministic search scales well to nominals, as is witnessed by diverse nondeterministic procedures for hybrid logic [28, 7, 29, 50, 33, 11, 25].

Our procedure differs considerably from existing nondeterministic approaches to hybrid logic in how it represents the search space for models. To deal with the equational constraints imposed by nominals, existing approaches [28, 7, 29, 50, 33, 11, 25] work with prefixed formulas and prefix-based propagation rules. Our procedure adopts a representation that does not involve prefixes. Instead, the procedure operates on sets of prefix-free formulas called clauses. The clausal representation yields a simple termination argument for our procedure that immediately follows from the finiteness of the Fischer-Ladner closure [19]. Moreover, the clausal representation is essential for our loop detection mechanism for eventualities. While variants of the clausal representation are used by pruning-based procedures for PDL and temporal logics [46, 47, 55, 15, 20, 32], our procedure is the first one to use clauses in combination with nondeterministic search.

In contrast to the procedures in [46, 32], which work with full Hintikka sets, we define our clausal model representation based on the notion of support, first introduced for a sublanguage of HPDL in [34]. This allows us to work with normal clauses, which are Hintikka sets that contain only elementary formulas we call literals. Similar to Hintikka sets, every normal clause can be seen as a state of a syntactic model. The satisfaction of arbitrary clauses is reduced to that of normal clauses via an abstract DNF computation. This eliminates the need for auxiliary clauses as used by Goré and Widmann [20, 54] or Pratt [47] in his graph-based procedure, which is important since auxiliary clauses cause problems for non-deterministic search in the presence of eventualities.

The most significant technical innovation in the present paper compared to our work in [34] is extending the notions of support and DNFs to full PDL. The

main challenge here is posed by iterations of the form α^* where α is a complex program. Such iterations can cause a naive disjunctive formula decomposition (as used, e.g., in [46, 47, 20, 54, 32]) to run into cycles. A simple example of such a cycle is as follows:



$$\begin{aligned}\langle a^{**}\rangle p &\equiv p \vee \langle a^{*}\rangle\langle a^{**}\rangle p\\ &\equiv p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p \vee \langle a^{**}\rangle p\end{aligned}$$

The graph on the left visualizes the decomposition of the formula $\langle a^{**}\rangle p$. The edges issuing from a formula φ to formulas ψ_1, ψ_2 indicate a decomposition step reducing φ to ψ_1 and ψ_2 such that $\varphi \equiv \psi_1 \vee \psi_2$. After two decomposition steps, the formula $\langle a^{**}\rangle p$ reduces to itself. The recursive equivalence obtained from the decomposition is shown on the right.

We deal with this complication by introducing a disjunctive decomposition of regular programs, called a program DNF. We define support and clausal DNF computation based on program DNFs. To compute program DNFs, we employ a novel algorithm whose correctness is argued independently from the correctness of clausal DNF computation or the overall procedure. This simplifies our correctness arguments as compared to Pratt [47] or Widmann [54].

To reflect the semantics of nominals in the clausal setting, we introduce an invariant that we call nominal coherence. Nominal coherence is maintained by a technique that we call nominal completion. Nominal completion, first introduced in [34], allows us to treat nominals in a way that is completely orthogonal to the treatment of the other features of HPDL.

Rather than starting with the full language, we develop the procedure in stages. We start with a procedure for the logic K* (an extension of basic modal logic with eventualities) and stepwise extend it to more complex logics. This way we can better explain how our approach deals with the individual syntactic features of HPDL. After introducing the basic procedure for K*, we consider its extension to PDL without tests. After that, we study the implications of allowing tests. Finally, we extend our approach to nominals, thus obtaining a decision procedure for full HPDL.

2 Formulas and Models

We begin by defining the syntax and semantics of HPDL. We assume a nonempty set of **atomic propositions** and a nonempty set of **actions**. We denote atomic propositions by the letters p and q, and actions by a and b. We partition the set of atomic propositions into ordinary atomic propositions and a special kind of atomic propositions called **nominals**. We denote nominals by the letters x and y.

For simplicity of presentation, we restrict ourselves to formulas in negation normal form (NNF). It is easily seen that every formula can be transformed into an equivalent formula in NNF in linear time. We define **programs** and **formulas** of HPDL by mutual recursion as follows:

$$\alpha ::= a \mid \varphi \mid \alpha + \alpha \mid \alpha\alpha \mid \alpha^* \qquad \textbf{programs}$$

$$\varphi ::= p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \qquad \textbf{formulas}$$

Note that in the definition, p ranges over both ordinary atomic propositions and nominals. Also note that in the literature (e.g., in [19, 47, 23]) one often uses a different notation for programs, writing $\alpha \cup \beta$ for $\alpha + \beta$, $\alpha;\beta$ for $\alpha\beta$, and $\varphi?$ for φ when φ acts as a program.

Formulas of the form $\langle\alpha\rangle\varphi$ are called **diamond formulas** and formulas $[\alpha]\varphi$ are called **box formulas**. Programs of the form φ are called **tests**. We call diamond formulas of the form $\langle\alpha^*\rangle\varphi$ **eventualities**.

Definition 2.1 A **model** $\mathcal{M}$ of HPDL consists of the following components:

- A nonempty set $|\mathcal{M}|$ of **states**. We call $|\mathcal{M}|$ the **domain** of $\mathcal{M}$.
- A **transition relation** $\xrightarrow{a}_{\mathcal{M}} \subseteq |\mathcal{M}| \times |\mathcal{M}|$ for every action a.
- A set $\mathcal{M}p \subseteq |\mathcal{M}|$ for every atomic proposition p. If p is a nominal, we additionally require $\mathcal{M}p$ to be a singleton (i.e., $|\mathcal{M}p| = 1$).

We fix some notations for binary relations. Let $\to \subseteq X \times X$. Then

$$\to^0 := \{(x,x) \mid x \in X\} \quad \to^{n+1} := \to \circ \to^n \quad \to^* := \bigcup_{n\geq 0} \to^n \quad \to^+ := \to \circ \to^*$$

The transition relations for complex programs and the **denotation** $\varphi^{\mathcal{M}}$ of a formula φ in $\mathcal{M}$ are defined by mutual induction on the structure of formulas and programs as follows:

$$\xrightarrow{\alpha+\beta}_{\mathcal{M}} = \xrightarrow{\alpha}_{\mathcal{M}} \cup \xrightarrow{\beta}_{\mathcal{M}} \qquad \xrightarrow{\alpha^*}_{\mathcal{M}} = \xrightarrow{\alpha}_{\mathcal{M}}{}^*$$

$$\xrightarrow{\alpha\beta}_{\mathcal{M}} = \xrightarrow{\alpha}_{\mathcal{M}} \circ \xrightarrow{\beta}_{\mathcal{M}} \qquad \xrightarrow{\varphi}_{\mathcal{M}} = \{(w,w) \mid w \in \varphi^{\mathcal{M}}\}$$

$$
\begin{aligned}
p^{\mathcal{M}} &= \mathcal{M}p \\
(\neg\varphi)^{\mathcal{M}} &= |\mathcal{M}| \setminus \varphi^{\mathcal{M}} \\
(\varphi \vee \psi)^{\mathcal{M}} &= \varphi^{\mathcal{M}} \cup \psi^{\mathcal{M}} \\
(\varphi \wedge \psi)^{\mathcal{M}} &= \varphi^{\mathcal{M}} \cap \psi^{\mathcal{M}} \\
(\langle\alpha\rangle\varphi)^{\mathcal{M}} &= \{\, v \mid \exists w\colon v \xrightarrow{\alpha}_{\mathcal{M}} w \text{ and } w \in \varphi^{\mathcal{M}} \,\} \\
([\alpha]\varphi)^{\mathcal{M}} &= \{\, v \mid \forall w\colon v \xrightarrow{\alpha}_{\mathcal{M}} w \text{ implies } w \in \varphi^{\mathcal{M}} \,\}
\end{aligned}
$$

We say two formulas φ and ψ are **equivalent** and write $\varphi \equiv \psi$ if $\varphi^{\mathcal{M}} = \psi^{\mathcal{M}}$ for every model $\mathcal{M}$. Some important equivalences are as follows:

$$
\begin{aligned}
\langle\psi\rangle\varphi &\equiv \psi \wedge \varphi & \qquad [\psi]\varphi &\equiv {\sim}\psi \vee \varphi \\
\langle\alpha+\beta\rangle\varphi &\equiv \langle\alpha\rangle\varphi \vee \langle\beta\rangle\varphi & [\alpha+\beta]\varphi &\equiv [\alpha]\varphi \wedge [\beta]\varphi \\
\langle\alpha\beta\rangle\varphi &\equiv \langle\alpha\rangle\langle\beta\rangle\varphi & [\alpha\beta]\varphi &\equiv [\alpha][\beta]\varphi \\
\langle\alpha^*\rangle\varphi &\equiv \varphi \vee \langle\alpha\rangle\langle\alpha^*\rangle\varphi & [\alpha^*]\varphi &\equiv \varphi \wedge [\alpha][\alpha^*]\varphi
\end{aligned}
$$

Note that in the equivalence for $[\psi]\varphi$ we write ${\sim}\varphi$ for the negation normal form of $\neg\varphi$. We need this notation since, given a formula φ, $\neg\varphi$ is in general not negation normal and hence not a formula according to our grammar.

We say a state $w \in |\mathcal{M}|$ **satisfies** φ and write $\mathcal{M}, w \vDash \varphi$ if $w \in \varphi^{\mathcal{M}}$. A model $\mathcal{M}$ **satisfies** (or is a **model of**) a formula φ if $\varphi^{\mathcal{M}}$ is nonempty (or, equivalently, if $\mathcal{M}, w \vDash \varphi$ for some state $w \in |\mathcal{M}|$). A formula is **satisfiable** if it has a model.

A **clause** C is a finite set of formulas. In contrast to common practice, we interpret clauses conjunctively: $\mathcal{M}, w \vDash C \;:\Leftrightarrow\; \forall \varphi \in C\colon \mathcal{M}, w \vDash \varphi$. We allow ourselves to write clauses in contexts that normally expect formulas. When occurring in such a context, a clause C is interpreted as the formula $\bigwedge_{\varphi\in C} \varphi$.

Formulas of the form p, $\neg p$, $\langle a\rangle\varphi$ and $[a]\varphi$ are called **literals**. Other formulas are called **nonliteral**. A clause C is called **normal** if

1. C contains only literals, and
2. C contains no complementary pair p, $\neg p$.

The **request** of a clause C w.r.t. an action a is defined as $C^{\Box}_a := \{\, \varphi \mid [a]\varphi \in C \,\}$. Intuitively, whenever a state satisfies C, all of its a-successors must satisfy $C^{\Box}_a$.

We call a model $\mathcal{M}$ a **clause model** if the domain of $\mathcal{M}$ is a set of clauses. Clause models can be seen as models where every state is annotated with a set of formulas. In general, the states of a clause model will not satisfy all of their annotations.

Definition 2.2 A clause model $\mathcal{M}$ is a **demo** if for every $C \in |\mathcal{M}|$ we have $\mathcal{M}, C \vDash C$.

Constructs closely related to demos have been studied in the context of temporal logics by Ben-Ari et al. [5] and by Emerson and Halpern [15] under the name "Hintikka structures".

Definition 2.3 A set A of formulas is **syntactically closed** if A is closed under the following rules:

$$\frac{\varphi \vee \psi}{\varphi,\ \psi} \qquad \frac{\langle a\rangle\varphi}{\varphi} \qquad \frac{\langle\psi\rangle\varphi}{\psi,\ \varphi} \qquad \frac{\langle\alpha+\beta\rangle\varphi}{\langle\alpha\rangle\varphi,\ \langle\beta\rangle\varphi} \qquad \frac{\langle\alpha\beta\rangle\varphi}{\langle\alpha\rangle\langle\beta\rangle\varphi} \qquad \frac{\langle\alpha^*\rangle\varphi}{\varphi,\ \langle\alpha\rangle\langle\alpha^*\rangle\varphi}$$

$$\frac{\varphi \wedge \psi}{\varphi,\ \psi} \qquad \frac{[a]\varphi}{\varphi} \qquad \frac{[\psi]\varphi}{\sim\psi,\ \varphi} \qquad \frac{[\alpha+\beta]\varphi}{[\alpha]\varphi,\ [\beta]\varphi} \qquad \frac{[\alpha\beta]\varphi}{[\alpha][\beta]\varphi} \qquad \frac{[\alpha^*]\varphi}{\varphi,\ [\alpha][\alpha^*]\varphi}$$

The least syntactically closed set containing a formula φ is called the **syntactic closure** of φ.

The syntactic closure is a minor variant of the Fischer-Ladner closure [19, 23]. Unlike the Fischer-Ladner closure, the syntactic closure is not closed under subformulas (e.g., p is not contained in the syntactic closure of $[p]q$). However, we do have that φ is in the syntactic closure of $[\alpha]\varphi$ for all α and φ, which yields closure under subformulas in the absence of tests. Also, we do not generally close under negation, restricting ourselves to the single negation necessary to deal with formulas $[\psi]\varphi$. Still, the essential properties of the Fischer-Ladner closure also hold in our case, in particular:

Proposition 2.4 ([23]) The syntactic closure of a formula φ is linear in the size of φ (i.e., the number of symbols in the string representation of φ).

3 Basic Decision Procedure

We introduce our approach on the logic K*. The logic K* is a sublogic of HPDL that is obtained by assuming the set of nominals to be empty and restricting the syntax of programs to actions a and iterated actions a^*.

Note that in K* we have only two types of diamond formulas: literals of the form $\langle a\rangle\varphi$ and nonliteral formulas $\langle a^*\rangle\varphi$ (and analogously for box formulas). We use the notations $\langle a^+\rangle\varphi := \langle a\rangle\langle a^*\rangle\varphi$ and $[a^+]\varphi := [a][a^*]\varphi$.

3.1 Support

In contrast to some of the related work [46, 5, 15, 32], where the formal development is based on the notion of Hintikka sets [24, 5, 15, 6, 32], we will work with normal clauses. To deal with nonliteral formulas, we will employ a mechanism

that we call support. Support extends a clause C to an, in a certain sense, maximal Hintikka set H such that $C = \{\, \varphi \in H \mid \varphi \text{ literal} \,\}$. We define support based on the following derivation rules.

$$\frac{\varphi_i}{\varphi_1 \vee \varphi_2}\ i \in \{1,2\} \qquad \frac{\varphi_1 \qquad \varphi_2}{\varphi_1 \wedge \varphi_2} \qquad \frac{\varphi}{\langle a^* \rangle \varphi} \qquad \frac{\langle a^+ \rangle \varphi}{\langle a^* \rangle \varphi} \qquad \frac{\varphi \qquad [a^+]\varphi}{[a^*]\varphi}$$

Applied to a clause C, the rules derive consequences of C.

Definition 3.1 A clause C **supports** a formula φ (notation $C \rhd \varphi$) if φ is derivable from C with the above rules.

For instance, let $C = \{p, [a^+](p \vee q)\}$. Then $C \rhd p \vee q$ by the first rule, and hence $C \rhd [a^*](p \vee q)$ by the last rule. We extend support to clauses as follows: $C \rhd D \ :\Leftrightarrow\ \forall \varphi \in D\colon\ C \rhd \varphi$.

Note that there are no rules deriving literals. Therefore, for every literal φ we have $C \rhd \varphi$ if an only if $\varphi \in C$. In the following, two properties of support will be essential:

Proposition 3.2 Let $\mathcal{M}, w \vDash C$ and $C \rhd \varphi$. Then $\mathcal{M}, w \vDash \varphi$.

Proposition 3.3 If $\mathcal{M}, w \vDash \varphi$, then there is a normal clause C such that $C \rhd \varphi$ and $\mathcal{M}, w \vDash C$.

Both properties are easily shown by induction on φ.

3.2 Demo Sets

We now define a compact representation of demos as sets of normal clauses, called demo sets.

Note that when applied to formulas of K*, the syntactic closure rules from §2 produce formulas of K*. We now fix an arbitrary finite, nonempty, syntactically closed set $\mathcal{F}$ that we call the **formula universe**. We will use $\mathcal{F}$ as a parameter: some of our definitions, including that of demo sets and the decision procedure, will be restricted to formulas from $\mathcal{F}$. This does not restrict the generality of our results since for every formula φ we can compute a formula universe containing φ whose cardinality is linear in the size of φ (Proposition 2.4).

Note also that support is compatible with the formula universe: Whenever $\varphi \in \mathcal{F}$ and $\frac{A}{\varphi}$ is an instance of a derivation rule for support, we have $A \subseteq \mathcal{F}$.

Let S be a set of clauses. We define:

- $C \xrightarrow{a}_S D \ :\Leftrightarrow\ C, D \in S$ and $D \rhd C^{\Box}_a$
- $C \xrightarrow{a^*}_S D \ :\Leftrightarrow\ C \xrightarrow{a}_S{}^{*} D$

Definition 3.4 A **demo set** is a nonempty set $S \subseteq 2^{\mathcal{F}}$ of normal clauses that satisfies the following property:

> For every $C \in S$ and every $\langle\alpha\rangle\varphi \in \mathcal{F}$ such that $C \rhd \langle\alpha\rangle\varphi$ there is some clause D such that $C \xrightarrow{\alpha}_S D$ and $D \rhd \varphi$.

We call this property **transition completeness**.

Checking whether transition completeness in the form it is defined holds for a set S may involve checking diamond formulas from $\mathcal{F}$ that are not present in S (but are supported by a clause in S). The following alternative characterization allows us to check transition completeness by looking only at diamond formulas that occur in S.

Lemma 3.5 A clause set S is transition-complete if and only if for every $C \in S$ we have:

1. If $\langle a\rangle\varphi \in C$ then there is some D such that $C \xrightarrow{a}_S D$ and $D \rhd \varphi$.
2. If $\langle a^+\rangle\varphi \in C$ then there are some D, E such that $C \xrightarrow{a}_S D \xrightarrow{a^*}_S E$ and $E \rhd \varphi$.

Proof The equivalence follows by induction on $\langle\alpha\rangle\varphi$ in the definition of transition completeness together with the following observations:

1. $C \rhd \langle a\rangle\varphi$ if and only if $\langle a\rangle\varphi \in C$.
2. $C \rhd \langle a^*\rangle\varphi$ if and only if $C \rhd \varphi$ or $\langle a^+\rangle\varphi \in C$.

■

We now show that every demo set yields a demo. Every nonempty clause set $S \subseteq 2^{\mathcal{F}}$ induces a clause model $\mathcal{M}_S$ as follows:

$$\begin{aligned} |\mathcal{M}_S| &= S \\ \xrightarrow{a}_{\mathcal{M}_S} &= \xrightarrow{a}_S \\ \mathcal{M}_S p &= \{\, C \in S \mid p \in C \,\} \end{aligned}$$

We show that $\mathcal{M}_S$ is a demo whenever S is a demo set.

Lemma 3.6 Let S be a set of normal clauses, $\{C, D\} \subseteq S$, $C \rhd [\alpha]\varphi$, and $C \xrightarrow{\alpha}_S D$. Then $D \rhd \varphi$.

Proof If $\alpha = a$, the claim is immediate by the definition of $\xrightarrow{a}_S$. If $\alpha = a^*$, there is some $n \geq 0$ such that $C \xrightarrow{a}{}^{n}_S D$. The claim follows by induction on n. ■

Lemma 3.7 Let S be a demo set. Then $\mathcal{M}_S$ is a demo.

Proof We show that $\mathcal{M}_S, C \models \varphi$ for every $C \in S$ and every φ such that $C \rhd \varphi$ by induction on φ. The propositional cases are straightforward. The cases for $\varphi = \langle\alpha\rangle\psi$ and $\varphi = [\alpha]\psi$ follow by transition completeness and Lemma 3.6, respectively, together with the observation that $\xrightarrow{\alpha}_{\mathcal{M}_S} = \xrightarrow{\alpha}_S$ for every α. ∎

By Lemma 3.7 and Proposition 3.2 we conclude that a formula $\varphi \in \mathcal{F}$ is satisfiable whenever there is a demo set $S \subseteq 2^{\mathcal{F}}$ that contains a clause supporting φ.

3.3 Disjunctive Normal Forms

Our decision procedure will construct demos for normal clauses. Hence we need an algorithm that for a general clause yields an equivalent disjunction of normal clauses.

Definition 3.8 A **disjunctive normal form** (**DNF**) of a clause C is a finite set $\{C_1, \ldots, C_n\}$ of normal clauses such that:

1. $C \equiv C_1 \vee \cdots \vee C_n$
2. For every normal clause D:
   $D \rhd C \iff C_i \subseteq D$ for some $i \in \{1, \ldots, n\}$

Example 3.9

- The set $\{\{p_1, p_2, q\}, \{p_1, p_2, \langle a^+\rangle q\}\}$ is a DNF of $\{p_1 \wedge p_2, \langle a^*\rangle q\}$.
- The set $\{\{\langle a^+\rangle p, [a^+]\langle a^+\rangle p\}\}$ is a DNF of $C = \{\langle a^*\rangle p, [a^*]\langle a^+\rangle p\}$. To see (1), observe that $\langle a^*\rangle p \equiv p \vee \langle a^+\rangle p$, $[a^*]\langle a^+\rangle p \equiv \langle a^+\rangle p \wedge [a^+]\langle a^+\rangle p$, and hence $C \equiv (p \vee \langle a^+\rangle p) \wedge \langle a^+\rangle p \wedge [a^+]\langle a^+\rangle p \equiv \langle a^+\rangle p \wedge [a^+]\langle a^+\rangle p$. For (2), note that every normal clause supporting C must include $\{\langle a^+\rangle p, [a^+]\langle a^+\rangle p\}$ in order to support $[a^*]\langle a^+\rangle p$. On the other hand, the literal $\langle a^+\rangle p$ suffices to support $\langle a^*\rangle p$.
  A second DNF of C is $\{\{p, \langle a^+\rangle p, [a^+]\langle a^+\rangle p\}, \{\langle a^+\rangle p, [a^+]\langle a^+\rangle p\}\}$. □

Proposition 3.10 If S is a DNF of C and $D \in S$, then $D \rhd C$.

Proof The claim is a straightforward consequence of condition (2) for DNFs. ∎

By the definition of DNFs, given a clause C and a DNF S of C, C is satisfiable if and only if so is a clause in S. We will now show that for every clause $C \in \mathcal{F}$ one can compute a DNF S of C such that $S \subseteq 2^{\mathcal{F}}$. Thus, do decide arbitrary formulas it suffices to have a decision procedure for normal clauses.

3.4 Computing DNFs

The decision procedure treats DNF computation as a black box in that it simply assumes the existence of an algorithm for computing a DNF for every clause. There are many ways to compute DNFs. We present a conceptually simple solution that admits simple and reasonably efficient implementations.

The basic idea is to base the computation of DNFs directly on the inductive definition of support. Observe that the rules defining support can be turned into tableau rules by reading them backwards:

$$\frac{\varphi_1 \vee \varphi_2}{\varphi_1 \mid \varphi_2} \qquad \frac{\varphi_1 \wedge \varphi_2}{\varphi_1,\ \varphi_2} \qquad \frac{\langle a^* \rangle \varphi}{\varphi \mid \langle a^+ \rangle \varphi} \qquad \frac{[a^*]\varphi}{\varphi,\ [a^+]\varphi}$$

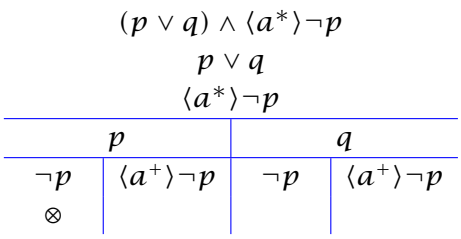
To compute a DNF of a clause C, we first develop C into a **complete tableau**. Consider, for instance, the clause $C = \{(p \vee q) \wedge \langle a^* \rangle \neg p\}$. Figure 1 shows a complete tableau for C. The second and third line are obtained by applying the rule for conjunctions to the first line. Applying the respective rules to the second and third line splits the tableau into four **branches**. The leftmost branch $(C \cup \{p \vee q, \langle a^* \rangle \neg p, p, \neg p\})$ contains a complementary pair of literals $(p, \neg p)$. We call branches with complementary literals **closed**. Branches containing no complementary literals are called **open**. To indicate that the leftmost branch is closed, we mark the branch with the symbol $\otimes$.

The literals of each open branch of a complete tableau form a normal clause. All clauses obtained this way from a complete tableau for a clause C can be shown to form a DNF of C (as demonstrated in Fig. 1). Since the tableau rules are compatible with the syntactic closure conditions, every DNF obtained with the rules from a clause $C \subseteq \mathcal{F}$ is guaranteed to contain only formulas from $\mathcal{F}$.

It remains to argue why every clause has a finite complete tableau, or, in other words, why the tableau construction terminates. This follows since the conclusions of every rule are either literals (and there are no rules for literals) or proper subformulas of its premise.

3.5 Clause Graphs

Our procedure decides the satisfiability of a normal clause by searching for a demo that contains the clause. It stepwise expands a clause model starting from the input clause by adding clauses and transitions until the underlying clause set is transition-complete. To simplify checking for transition completeness, we represent transitions explicitly by labeled edges that we call links. In other words, we work on graphs that have clauses as nodes and links as edges. Our procedure will add new clauses and links until there is exactly one outgoing link for every



Resulting DNF: $\{\{p, \langle a^+ \rangle \neg p\},\ \{q, \neg p\},\ \{q, \langle a^+ \rangle \neg p\}\}$

Figure 1: A complete tableau for $\{(p \vee q) \wedge \langle a^* \rangle \neg p\}$

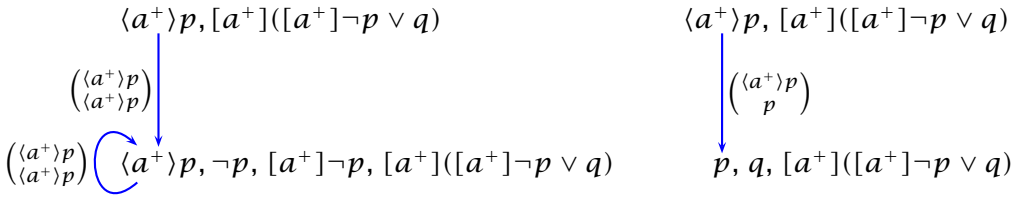


Figure 2: Two clause graphs

diamond formula in every clause. This suffices to guarantee transition completeness for ordinary diamonds. For eventualities, we must additionally ensure the existence of fulfilling paths validating condition (2) of Lemma 3.5. As long as the graph contains no more than one link per diamond formula (a condition we call functionality), this reduces to checking the graph for the absence of a certain kind of loops (the "bad cycles" from [4]).

Graphs can be represented as shown in the example in Fig. 2 (the left graph stems from a failed attempt to find a demo containing the clause $\{\langle a^+ \rangle p, [a^+]([a^+]\neg p \vee q)\}$ while the right graph yields such a demo). A link from a clause C to a clause D labeled with the pair $\binom{\langle a \rangle \varphi}{\psi}$ corresponds to an a-transition that establishes the satisfaction of $\langle a \rangle \varphi$ in C. The formula ψ in the annotation is used to track the fulfillment of eventualities. So, an annotation of the form $\binom{\langle a^+ \rangle \varphi}{\varphi}$ indicates that D fulfills $\langle a^+ \rangle \varphi$ in C by satisfying φ (rather than $\langle a^+ \rangle \varphi$). Note that the two graphs in Fig. 2 contain only normal clauses as nodes (e.g., there is no node $\{\langle a^* \rangle p, [a^*]([a^+]\neg p \vee q)\}$). In this respect they differ from the data structures in [47, 1, 20, 54]. This difference is essential for the correctness of our approach (see §3.8.2).

We define links as tuples $C\binom{\langle a \rangle \varphi}{\psi}D$ satisfying certain additional conditions. Given a clause C of a graph $\mathcal{G}$ and a diamond formula $\langle a \rangle \varphi \in C$ for which $\mathcal{G}$ contains no outgoing link, our procedure will compute some clause D and

formula ψ and extend $\mathcal{G}$ by D and $C\binom{\langle a\rangle\varphi}{\psi}D$. How is D selected?

Clearly, whenever $\langle a\rangle\varphi \in C$ and C is satisfied by a model $\mathcal{M}$, $\mathcal{M}$ must also satisfy $C_a^\Box;\varphi$. Hence, every DNF $\mathcal{D}$ of $C_a^\Box;\varphi$, must contain a normal clause satisfied by $\mathcal{M}$. An intuitive idea would be to select D from the set of candidates provided by some DNF of $C_a^\Box;\varphi$. This, however, turns out to be insufficient for correctness (see the discussion in §3.8.1). Specifically, we need to refine the computation of D for diamond formulas of the form $\langle a^+\rangle\varphi$. Rather than selecting D from a DNF of $C_a^\Box;\langle a^*\rangle\varphi$, we will first decompose $\langle a^*\rangle\varphi$ into φ and $\langle a^+\rangle\varphi$ (which is justified since $\langle a^*\rangle\varphi \equiv \varphi \vee \langle a^+\rangle\varphi$), and then select D from the union of a DNF of $C_a^\Box;\varphi$ and a DNF of $C_a^\Box;\langle a^+\rangle\varphi$.

Formally, we realize this via an auxiliary notion that we call diamond decomposition.

Definition 3.11 The **diamond decomposition** $\mathcal{D}\varphi$ of a formula φ is defined as follows:

1. $\mathcal{D}(\langle a^*\rangle\varphi) = \{\varphi, \langle a^+\rangle\varphi\}$
2. $\mathcal{D}\varphi = \{\varphi\}$ if φ is not of the form $\langle a^*\rangle\psi$

Note that above and in the following, we use higher-order-style notation for function application, which does not require putting parentheses around arguments. In particular, we do not distinguish between $\mathcal{D}\varphi$ and $\mathcal{D}(\varphi)$. Parentheses are only used to improve readability (like with $\mathcal{D}(\langle a^*\rangle\varphi)$ above) or to resolve ambiguity.

Also note that whenever $\psi \in \mathcal{D}\varphi$, we have $\{\psi\} \rhd \varphi$. Moreover, we have $\mathcal{D}\varphi \subseteq \mathcal{F}$ whenever $\varphi \in \mathcal{F}$. We write $X;x$ for $X \cup \{x\}$.

Definition 3.12 A **link** is a tuple $C\binom{\langle a\rangle\varphi}{\psi}D$ such that:

1. C and D are normal clauses,
2. $\langle a\rangle\varphi \in C$,
3. $\psi \in \mathcal{D}\varphi$,
4. D is an element of a DNF of $C_a^\Box;\psi$.

Note that by the definition of DNFs, for every link $C\binom{\langle a\rangle\varphi}{\psi}D$ we have $D \rhd C_a^\Box;\psi$. We call links of the form $C\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi}D$ **delegating** and links of the form $C\binom{\langle a^+\rangle\varphi}{\varphi}D$ **fulfilling**. Note that every link $C\binom{\langle a^+\rangle\varphi}{\psi}D$ is either delegating or fulfilling.

Definition 3.13 A **(clause) graph** $\mathcal{G}$ is a set of normal clauses and links such that $\{C, D\} \subseteq \mathcal{G}$ whenever $C\xi D \in \mathcal{G}$.

Figure 2 shows a graphical representation of two graphs: one consisting of the clauses $C_1 = \{\langle a^+\rangle p, [a^+]([a^+]\neg p \vee q)\}$, $C_2 = \{\langle a^+\rangle p, \neg p, [a^+]\neg p, [a^+]([a^+]\neg p \vee q)\}$ and the links $C_1\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_2$, $C_2\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_2$, and one consisting of $C_1, C_3 = \{p, q, [a^+]([a^+]\neg p \vee q)\}$, and the link $C_1\binom{\langle a^+\rangle p}{p}C_3$.

3.6 Demo Graphs

A demo graph is a syntactic representation of a demo as a clause graph. We will define demo graphs so that their clauses form a demo set. The links of a demo graph will allow us to reduce the conditions in Lemma 3.5 to more primitive conditions that are easier to check algorithmically. With Proposition 3.10 we obtain:

Proposition 3.14 If $\{C, D\} \subseteq S$ and $C\binom{\langle a\rangle\varphi}{\psi}D$ is a link, then $C \xrightarrow{a}_S D$ and $D \rhd \psi$.

Thus, a link $C\binom{\langle a\rangle\varphi}{\psi}D$ ensures that D is an a-successor of C in any clause set containing C and D. Since D also supports φ, D satisfies the first condition of Lemma 3.5 for $\langle a\rangle\varphi \in C$. More importantly, finite sequences of links can be seen as certificates for the fulfillment of eventualities, i.e., the second condition of Lemma 3.5. We define a **path for** $\langle a^+\rangle\varphi$ to be a nonempty sequence

$$(C_1\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi}C_2)(C_2\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi}C_3)\ldots(C_{n-2}\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi}C_{n-1})(C_{n-1}\binom{\langle a^+\rangle\varphi}{\psi}C_n)$$

of links. A path of the above form is called a **run for** $\langle a^+\rangle\varphi$ **in** C if $C_1 = C$ and $\psi = \varphi$, and a **loop for** $\langle a^+\rangle\varphi$ if $C_1 = C_n$ and $\psi = \langle a^+\rangle\varphi$.

With Proposition 3.14, it easily follows that a run for some $\langle a^+\rangle\varphi \in C$ constitutes a certificate for the fulfillment of $\langle a^+\rangle\varphi \in C$ according to Lemma 3.5.

A graph $\mathcal{G}$ is **functional** if $C\binom{\varphi}{\psi}D \in \mathcal{G}$ and $C\binom{\varphi}{\psi'}D' \in \mathcal{G}$ implies $D = D'$ and $\psi = \psi'$. A graph $\mathcal{G}$ **realizes** a diamond formula $\langle a\rangle\varphi \in C$ if $C\binom{\langle a\rangle\varphi}{\psi}D \in \mathcal{G}$ for some ψ and D. Note that the graphs in Fig. 2 are functional and realize every diamond formula.

Definition 3.15 A graph $\mathcal{G}$ is a **demo graph** if $\mathcal{G}$ is finite, functional, realizes every diamond formula, and contains no loops.

Proposition 3.16 Let $\mathcal{G}$ be a demo graph and $\langle a^+\rangle\varphi \in C \in \mathcal{G}$. Then $\mathcal{G}$ contains a unique run for $\langle a^+\rangle\varphi$ in C.

Proof The existence of a run for every $\langle a^+\rangle\varphi \in C \in \mathcal{G}$ follows from $\mathcal{G}$ being finite, realizing every diamond formula and containing no loops. The uniqueness of runs follows from the functionality of $\mathcal{G}$ (which is given by the assumption that $\mathcal{G}$ is a demo graph). ∎

Proposition 3.17 The clauses of a demo graph form a demo set.

Proof Let $\mathcal{G}$ be a demo graph and S the set of its clauses. To show that S is a demo set it suffices to establish the two conditions of Lemma 3.5. Condition (1) follows with Proposition 3.14 since $\mathcal{G}$ realizes every diamond formula. Condition (2) follows with Proposition 3.14 and Proposition 3.16. ∎

Input: a normal clause $C_0 \subseteq \mathcal{F}$

Variable: $\mathcal{G} := \{C_0\}$

Invariant: $\mathcal{G}$ functional and loop-free

while $\mathcal{G}$ does not realize every $\langle a\rangle\varphi \in C \in \mathcal{G}$ **do**

1. **pick** $\langle a\rangle\varphi \in C \in \mathcal{G}$ such that $\mathcal{G}$ does not realize $\langle a\rangle\varphi$ in C
2. **choose** $\psi \in \mathcal{D}\varphi$
3. **pick** DNF S of $C^{\Box}_{a}\,;\psi$
4. **if** $S \neq \emptyset$ **then choose** $D \in S$ **else backtrack**
5. $\mathcal{G} := \mathcal{G}\,;D\,;C\binom{\langle a\rangle\varphi}{\psi}D$
6. **if** $\mathcal{G}$ contains a loop **then backtrack**

Return: – **satisfiable** if while-loop terminates regularly
– **unsatisfiable** if while-loop terminates through backtracking

Note: **pick** is a "don't care" decision (no backtracking needed) and **choose** is a "don't know" decision (backtracking needed).

Figure 3: Decision procedure for K*

3.7 Decision Procedure

Figure 3 defines a decision procedure for K*. The procedure is defined for normal input clauses, the extension to arbitrary clauses being straightforward (see §3.3). Given an initial clause $C_0 \subseteq \mathcal{F}$, the procedure tries to extend C_0 to a demo graph by adding clauses and links until the resulting graph $\mathcal{G}$ realizes all of its diamond formulas. The functionality and loop-freeness of $\mathcal{G}$ are maintained as invariants of the construction. In case the while loop terminates successfully, the invariants together with the negated loop condition imply that $\mathcal{G}$ is a demo graph containing C_0. Hence, the procedure returns "satisfiable". The failure to realize a diamond formula is resolved by backtracking to the last "don't know"[1] choice point. If there is no choice point to backtrack to, the procedure terminates returning "unsatisfiable".

The procedure terminates on all inputs under the assumption that the DNF computed for a clause $C \in \mathcal{F}$ contains only formulas from $\mathcal{F}$. This assumption can be fulfilled as shown in §3.4. We then obtain:

1. $\{\, C \mid C \in \mathcal{G} \,\} \subseteq 2^{\mathcal{F}}$
2. $|\{\, C\xi D \mid C\xi D \in \mathcal{G} \,\}| \leq |\mathcal{F}| \cdot 2^{2\cdot|\mathcal{F}|}$

Thus, the size of $\mathcal{G}$ is exponentially bounded in $|\mathcal{F}|$. From this, termination of

[1] We use the terms "don't care" and "don't know" as they are defined in [37], §5.

the procedure follows in a straightforward way since the "don't know" choice points in steps 2 and 4 are finitely branching.

The correctness of the procedure in case of success follows by Proposition 3.17 and Lemma 3.7. It remains to argue that the procedure succeeds for every satisfiable clause. Our argument employs a measure function for clauses that we call distance.

Let $\mathcal{M}$ be a model, C a clause, φ a formula and a an action. The **a-distance from C to φ in $\mathcal{M}$** is defined as follows:

$$\delta^a_{\mathcal{M}} C\varphi := \min\{\, n \in \mathbb{N} \mid \exists v, w : \mathcal{M}, v \vDash C \text{ and } v \xrightarrow{a}{}^{n}_{\mathcal{M}} w \text{ and } \mathcal{M}, w \vDash \varphi \,\}$$

where $\min \emptyset = \infty$ and $n < \infty$ for all $n \in \mathbb{N}$. This definition of $\delta^a_{\mathcal{M}}$ is inspired by an analogous construction in Baader's [4] correctness proof for (a syntactic variant of) test-free PDL. As an immediate consequence of the definition we obtain:

Proposition 3.18 $\delta^a_{\mathcal{M}} C\varphi < \infty$ if and only if $\mathcal{M}$ satisfies $C\,;\langle a^*\rangle\varphi$.

Note also that $\delta^a_{\mathcal{M}} C\varphi \leq \delta^a_{\mathcal{M}} D\varphi$ whenever $C \subseteq D$.

A model $\mathcal{M}$ **satisfies a link** $C\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi} D$ if the following conditions are satisfied:

1. $\delta^a_{\mathcal{M}} C\varphi > 0 \Longrightarrow \delta^a_{\mathcal{M}} C\varphi > \delta^a_{\mathcal{M}} D\varphi$
2. $\delta^a_{\mathcal{M}} D\varphi > 0$

A model **satisfies a graph** $\mathcal{G}$ if it satisfies all clauses and all delegating links of $\mathcal{G}$.

Proposition 3.19 Satisfiable graphs contain no loops.

Proof Let $\mathcal{G}$ be a graph and $\mathcal{M}$ a satisfying model. Suppose, for contradiction, $\mathcal{G}$ contains a loop $(C_1\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi} C_2) \ldots (C_n\binom{\langle a^+\rangle\varphi}{\langle a^+\rangle\varphi} C_1)$ $(n \geq 1)$. Since every clause of the loop occurs as the target of a link satisfied by $\mathcal{M}$, we have $\delta^a_{\mathcal{M}} C_i\varphi > 0$ for every $i \in [1, n]$. Consequently, $\delta^a_{\mathcal{M}} C_1\varphi > \delta^a_{\mathcal{M}} C_2\varphi > \cdots > \delta^a_{\mathcal{M}} C_n\varphi > \delta^a_{\mathcal{M}} C_1\varphi$. Contradiction. ∎

Clearly, every graph constructed by the procedure is functional since the procedure never adds links for diamond formulas that are already realized. Hence, by Proposition 3.19, the procedure succeeds on a satisfiable clause C_0 if it can extend C_0 to a satisfiable graph that realizes all of its diamond formulas. Since the procedure terminates, this is the case if one can always choose ψ and D (in steps 2 and 4) so that the resulting extension of $\mathcal{G}$ is satisfiable. This is possible by the following lemma.

Lemma 3.20 Let $\mathcal{G}$ be a graph satisfied by a model $\mathcal{M}$ and let $\langle a\rangle\varphi \in C \in \mathcal{G}$. Then there is some $\psi \in \mathcal{D}\varphi$ such that for every DNF S of $C^{\Box}_a\,;\psi$ there is a clause $D \in S$ such that $\mathcal{M}$ satisfies $\mathcal{G}\,;D\,;C\binom{\langle a\rangle\varphi}{\psi} D$.

Proof Unless φ has the form $\langle a^* \rangle \varphi'$, the claim reduces to showing that $\mathcal{M}$ satisfies some clause in every DNF of $C_a^\Box ; \varphi$. This easily follows with condition (1) for DNFs since $\mathcal{M}$ satisfies C and hence $C_a^\Box ; \varphi$.

So, suppose φ has the form $\langle a^* \rangle \varphi'$. Then $\mathcal{M}$ satisfies $C_a^\Box ; \langle a^* \rangle \varphi'$, i.e., $\mathcal{M}$ satisfies $C_a^\Box ; \varphi'$ or $\mathcal{M}$ satisfies $C_a^\Box ; \langle a^+ \rangle \varphi'$.

Suppose $\mathcal{M}$ satisfies $C_a^\Box ; \varphi'$. Then for every DNF S of $C_a^\Box ; \varphi'$ there is a clause $D \in S$ such that $\mathcal{M}$ satisfies D (condition (1) for DNFs). Hence, $\mathcal{M}$ satisfies $G ; D ; C \binom{\langle a^+ \rangle \varphi'}{\varphi'} D$. The claim follows since $\varphi' \in \mathcal{D}(\langle a^* \rangle \varphi')$.

Finally, suppose $\mathcal{M}$ does not satisfy $C_a^\Box ; \varphi'$. Then $\mathcal{M}$ satisfies $C_a^\Box ; \langle a^+ \rangle \varphi'$. Let S be a DNF of $C_a^\Box ; \langle a^+ \rangle \varphi'$ and let D be a clause in S such that $\delta^a_{\mathcal{M}} D \varphi'$ is minimal. By condition (1) for DNFs, we have $\delta^a_{\mathcal{M}} D \varphi' = \delta^a_{\mathcal{M}} (C_a^\Box ; \langle a^+ \rangle \varphi') \varphi'$, and hence $\mathcal{M}$ satisfies D (by Proposition 3.18).

We complete the proof by showing that $\mathcal{M}$ satisfies $C \binom{\langle a^+ \rangle \varphi'}{\langle a^+ \rangle \varphi'} D$. Since $\mathcal{M}$ does not satisfy $C_a^\Box ; \varphi'$, we have $\delta^a_{\mathcal{M}} D \varphi' \geq \delta^a_{\mathcal{M}} C_a^\Box \varphi' > 0$.

Suppose $\delta^a_{\mathcal{M}} C \varphi' > 0$. We have to show that $\delta^a_{\mathcal{M}} C \varphi' > \delta^a_{\mathcal{M}} D \varphi'$. Note that $\delta^a_{\mathcal{M}} C \varphi' > 1$ since otherwise $\mathcal{M}$ would satisfy $C_a^\Box ; \varphi'$, contradicting our assumption. Also, $\delta^a_{\mathcal{M}} C \varphi' < \infty$ (by Proposition 3.18 since $\langle a^+ \rangle \varphi' \in C$). Hence, there are some u, v, w such that $\mathcal{M}, u \vDash C$, $u \xrightarrow{a}_{\mathcal{M}} v \xrightarrow{a}{}^{\delta^a_{\mathcal{M}} C \varphi' - 1}_{\mathcal{M}} w$, and $\mathcal{M}, w \vDash \varphi'$. Clearly, $\mathcal{M}, v \vDash C_a^\Box ; \langle a^+ \rangle \varphi'$, and hence there is a clause $E \in S$ such that $\mathcal{M}, v \vDash E$. Then $\delta^a_{\mathcal{M}} E \varphi' \leq \delta^a_{\mathcal{M}} C \varphi' - 1$. Since D is chosen from S such that $\delta^a_{\mathcal{M}} D \varphi'$ is minimal, we have $\delta^a_{\mathcal{M}} C \varphi' > \delta^a_{\mathcal{M}} E \varphi' \geq \delta^a_{\mathcal{M}} D \varphi'$. ∎

By the above observations we conclude:

Theorem 3.21 The procedure in Fig. 3 terminates for every input clause C and returns "satisfiable" if and only if C is satisfiable.

3.8 Remarks

3.8.1 Diamond Decomposition

Since $\langle a^* \rangle \varphi \equiv \varphi \vee \langle a^+ \rangle \varphi$, it is easy to see that if S is a DNF of $C ; \varphi$ and S' is a DNF of $C ; \langle a^+ \rangle \varphi$, then $S \cup S'$ is a DNF of $C ; \langle a^* \rangle \varphi$. This raises the question why, given some $\langle a^+ \rangle \varphi \in C$, we choose $\psi \in \mathcal{D}(\langle a^* \rangle \varphi)$ in step 2 of the procedure and then compute a DNF of $C_a^\Box ; \psi$ rather than omitting step 2 and simply computing a DNF of $C_a^\Box ; \langle a^* \rangle \varphi$. Of course, we would then have problems selecting a target formula for the link in step 5, but this can be solved by adapting the definition of links following [34].

The real problem is that the resulting procedure may fail on satisfiable clauses depending on the choice of the DNF in step 3. For instance, consider the satisfiable clause $C = \{\langle a^+ \rangle p, [a^+] \langle a^+ \rangle p\}$. The modified procedure running with C as input may then pick $\{C\}$ as a DNF of $C_a^\Box ; \langle a^* \rangle p$ and hence end up with the graph

$\mathcal{G} = \{C, C\binom{\langle a^+\rangle p}{\langle a^+\rangle p} C\}$, which contains a loop. Since $\mathcal{G}$ is the only graph that can be constructed with the picked DNF, the procedure fails to construct a demo graph and returns "unsatisfiable".

To make the procedure succeed on every satisfiable clause, we need to ensure that for every $\langle a\rangle\varphi \in C \in \mathcal{G}$, we can add a link $C\binom{\langle a\rangle\varphi}{\psi} D$ (for some D) for every $\psi \in \mathcal{D}\varphi$ that is consistent with $C^{\Box}_a$. Intuitively, if φ is of the form $\langle\alpha\rangle\varphi'$, it must be possible to add a link for every possible action sequence denoted by α. This requirement will be made precise when we discuss full PDL.

While the required property can be achieved by restricting DNFs to satisfy additional conditions (as done in [34]), the conditions would unnecessarily restrict us in our methods for computing DNFs. In particular, the tableau-based computation method suggested in §3.4 is not compatible with such restrictions. Besides giving us full freedom in computing DNFs, the solution chosen in this paper better separates propositional reasoning from the modal reasoning steps necessary to deal with eventualities.

3.8.2 Non-normal Clauses

Both Pratt [47] and Goré and Widmann [20] base their procedures on graphs that contain non-normal clauses. Such clauses can be used to represent intermediate steps in the computation of a DNF and thus potentially increase information reuse.

Our approach does not employ such auxiliary clauses. Adding them naively leads to problems. The reason is that we search for demo graphs that are required to be functional. When is a graph with non-normal clauses functional? A natural definition is that the functionality condition for normal clauses remains as is while non-normal clauses are restricted to at most one successor (i.e., one way to continue the DNF computation). Indeed, one can show that without eventualities, this setup is an adequate basis for a decision procedure. In particular, we have that a (not necessarily normal) clause is satisfiable if and only if it is contained in a functional demo graph with non-normal clauses.

With eventualities, however, such graphs cannot represent a demo for every satisfiable clause. To see this, consider the graph $\mathcal{G}$ in Fig. 4. The unlabeled links represent a DNF computation for the clause $\{\langle a^*\rangle p, \langle a^*\rangle\neg p, \varphi\}$ while the labeled links realize diamond formulas. Note that all diamond formulas in normal clauses are realized. Moreover, all clauses of $\mathcal{G}$ except $\{p, \neg p, \varphi\}$ are satisfiable. Still, $\mathcal{G}$ contains no functional subgraph that fulfills every eventuality (i.e., contains a run for every eventuality). Every subgraph of $\mathcal{G}$ fulfilling all of its eventualities would have to contain both outgoing links from the topmost clause, which contradicts functionality.

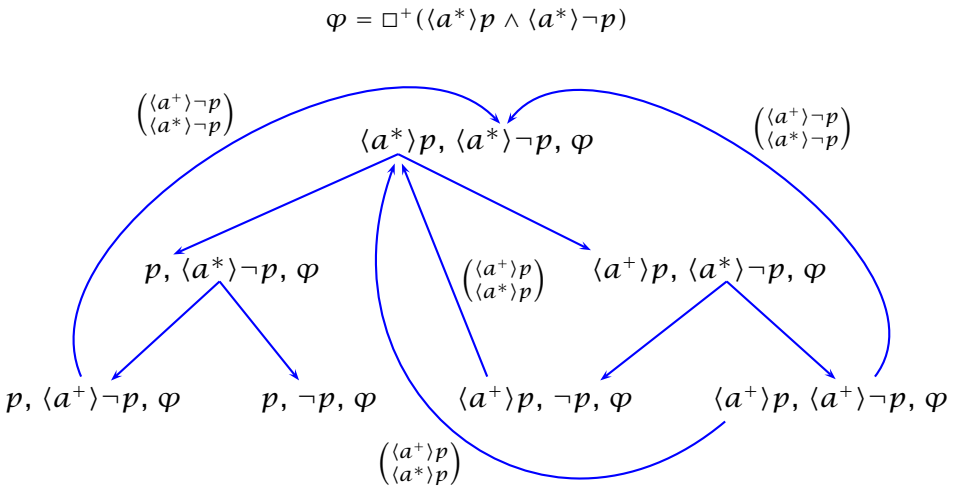


Figure 4: A graph with non-normal clauses

We conclude that DNF computation is crucial to our approach. It cannot be replaced by propositional reasoning on auxiliary, non-normal clauses as used in [47, 20] without giving up either the functionality requirement or the uniqueness of clauses in a clause graph.

4 Regular Programs

We now extend our approach to test-free PDL. While the set of nominals is still assumed to be empty, we extend the syntax of programs to allow program union ($\alpha + \beta$), concatenation ($\alpha\beta$), and iteration (α^*). In other words, we allow as programs arbitrary regular expressions over actions.

The most challenging part of extending our approach from the simple program syntax of K* to regular programs is finding suitable notions of support and DNF. We base both notions on a DNF-like decomposition of regular programs, which we call a program DNF.

4.1 Program Equivalence

Being regular expressions, programs describe regular languages over actions. Regular languages provide a complementary view on the semantics of programs. Every action string $a_1 \dots a_n$ in the regular language $\mathcal{L}\alpha$ denoted by a program α can be interpreted as the composition of the transition relations $\xrightarrow{a_i}_{\mathcal{M}}$. The interpretation of α is then the union of the interpretations of all strings in $\mathcal{L}\alpha$. This view will be essential in arguing the correctness of our approach.

We now provide a brief summary of the relevant concepts from the theory of regular languages. A **string** is a finite sequence of actions $a_1 \dots a_n$ where $n \geq 0$. We denote the empty string by ε. The letters σ and τ range over strings. A **language** L is a set of strings. We define $L \cdot L' := \{ \sigma\tau \mid \sigma \in L, \tau \in L' \}$. Given a language L we define:

$$L^0 := \{\varepsilon\} \qquad L^{n+1} := L \cdot L^n \qquad L^* := \bigcup_{n \in \mathbb{N}} L^n$$

To every program α we inductively assign a **language** $\mathcal{L}\alpha$:

$$\mathcal{L}a := \{a\} \quad \mathcal{L}(\alpha + \beta) := \mathcal{L}\alpha \cup \mathcal{L}\beta \quad \mathcal{L}(\alpha\beta) := \mathcal{L}\alpha \cdot \mathcal{L}\beta \quad \mathcal{L}(\alpha^*) := (\mathcal{L}\alpha)^*$$

We write $\alpha \equiv \beta$ if $\mathcal{L}\alpha = \mathcal{L}\beta$.

As with K*, we want a DNF of a clause C to give us a set of normal clauses $\{C_1, \dots, C_n\}$ such that $C \equiv C_1 \vee \dots \vee C_n$. For PDL, this raises the question of how to compute the literals corresponding to diamond formulas $\langle\alpha\rangle\varphi$ and box formulas $[\alpha]\varphi$ for arbitrary regular programs α. We answer the question by introducing a disjunctive decomposition of programs. Intuitively, given an arbitrary program α, we compute a set $\{\alpha_1, \dots, \alpha_n\}$ of simpler programs such that $\alpha \equiv \alpha_1 + \dots + \alpha_n$. From this it easily follows that $\langle\alpha\rangle\varphi \equiv \langle\alpha_1\rangle\varphi \vee \dots \vee \langle\alpha_n\rangle\varphi$ for every formula φ (and dually for boxes).

The approach sketched is not compatible with the syntactic closure conditions. As with K*, we fix a finite, nonempty, syntactically closed formula universe $\mathcal{F}$ (this is possible since the syntactic closure rules from §2 applied to formulas of test-free PDL produce formulas of test-free PDL). In general we cannot guarantee that $\langle\alpha\rangle\varphi \in \mathcal{F}$ implies $\langle\beta\rangle\varphi \in \mathcal{F}$ for every β in a decomposition of α. To stay in $\mathcal{F}$, we generalize our approach and work with finite sequences of programs $\eta = \beta_1{:}\dots{:}\beta_m$ ($m \geq 0$). Intuitively, η is equivalent to the program $\beta_1 \dots \beta_m$ obtained by composing all of the programs in η. Hence, we will allow ourselves to write sequences in places where one would normally expect a program, interpreting a sequence $\beta_1{:}\dots{:}\beta_m$ as the corresponding program $\beta_1 \dots \beta_m$.

Let $\eta = \alpha_1{:}\dots{:}\alpha_n$ for some $n \geq 0$. We define $\mathcal{L}\eta$ as follows:

- $\mathcal{L}\eta = \{\varepsilon\}$ if $n = 0$,
- $\mathcal{L}\eta = \mathcal{L}\alpha_1 \cdot \dots \cdot \mathcal{L}\alpha_n$ if $n > 0$.

Thus, we have $\mathcal{L}(\alpha_1{:}\dots{:}\alpha_n) = \mathcal{L}(\alpha_n \dots \alpha_n)$.

We denote the empty program sequence with λ. Given a program α and sequences $\eta = \alpha_1{:}\dots{:}\alpha_n$, $\theta = \beta_1{:}\dots{:}\beta_m$, we write $\alpha{:}\eta$ ($\eta{:}\theta$) for the sequence $\alpha{:}\alpha_1{:}\dots{:}\alpha_n$ ($\alpha_1{:}\dots{:}\alpha_n{:}\beta_1{:}\dots{:}\beta_m$). We call a sequence η **normal** if $\eta = \lambda$ or $\eta = a{:}\alpha_1{:}\dots{:}\alpha_n$ for some action a and programs $\alpha_1, \dots, \alpha_n$ ($n \geq 0$).

Definition 4.1 A **DNF of a program** α is a finite set $\{\eta_1, \dots, \eta_n\}$ of normal program sequences such that $\alpha \equiv \eta_1 + \dots + \eta_n$ (or, equivalently, $\mathcal{L}\alpha = \mathcal{L}\eta_1 \cup \dots \cup$

$\mathcal{L}\eta_n$). A **DNF of a program sequence** η is a set $\mathcal{D}$ of program sequences such that $\eta/\mathcal{D}$ is derivable by the following rules:

$$\frac{}{\lambda/\{\lambda\}} \qquad \frac{\mathcal{D} \text{ DNF of } \alpha}{\alpha/\mathcal{D}} \qquad \frac{\alpha/\mathcal{D} \qquad \lambda \notin \mathcal{D}}{\alpha{:}\eta/\{\,\theta{:}\eta \mid \theta \in \mathcal{D}\,\}} \qquad \frac{\alpha/\mathcal{D} \qquad \eta/\mathcal{E} \qquad \lambda \in \mathcal{D}}{\alpha{:}\eta/\{\,\theta{:}\eta \mid \theta \in \mathcal{D},\ \theta \neq \lambda\,\} \cup \mathcal{E}}$$

As a straightforward consequence of the definition we obtain:

Proposition 4.2 Let $\{\eta_1, \ldots, \eta_n\}$ be a DNF of a program sequence η. Then:

1. Every sequence in $\{\eta_1, \ldots, \eta_n\}$ is normal.
2. $\eta \equiv \eta_1 + \cdots + \eta_n$.

Proof We show both claims by induction on the derivation of $\eta/\{\eta_1, \ldots, \eta_n\}$. We distinguish four cases:

- $n = 1$ and $\eta = \eta_1 = \lambda$. Then both claims are immediate.
- $\eta = \alpha$ and $\{\eta_1, \ldots, \eta_n\}$ is a DNF of α. Then both claims are immediate by definition of program DNFs.
- $\eta = \alpha{:}\eta'$, $\alpha/\mathcal{D}$, $\lambda \notin \mathcal{D}$, and $\{\eta_1, \ldots, \eta_n\} = \{\,\theta{:}\eta' \mid \theta \in \mathcal{D}\,\}$ for some $\mathcal{D}$. By the inductive hypothesis, every sequence in $\mathcal{D}$ is normal and $\alpha \equiv \Sigma\mathcal{D}$. Since $\lambda \notin \mathcal{D}$ and the elements of $\mathcal{D}$ are normal, they all have the form $a{:}\theta$ (for some a and θ). Consequently, every η_i has the form $a{:}\theta{:}\eta'$. In particular, every η_i is normal. Claim (2) follows from $\alpha \equiv \Sigma\mathcal{D}$ and the equivalence $(\alpha_1 + \cdots + \alpha_n)\beta \equiv \alpha_1\beta + \cdots + \alpha_n\beta$ (which holds for every $\alpha_1, \ldots, \alpha_n, \beta$) since a program sequence $\beta_1{:}\ldots{:}\beta_m$ is interpreted the same as the composition $\beta_1 \ldots \beta_m$.
- $\eta = \alpha{:}\eta'$, $\alpha/\mathcal{D}$, $\eta'/\mathcal{E}$, $\lambda \in \mathcal{D}$, and $\{\eta_1, \ldots, \eta_n\} = \{\,\theta{:}\eta' \mid \theta \in \mathcal{D}\,\} \cup \mathcal{E}$ for some $\mathcal{D}$, $\mathcal{E}$. By the inductive hypothesis, the sequences in $\mathcal{D}$ and $\mathcal{E}$ are normal, $\alpha \equiv \Sigma\mathcal{D}$, and $\eta' \equiv \Sigma\mathcal{E}$. Since the sequences in $\mathcal{D}$ are normal, so are the sequences in $\{\,\theta{:}\eta' \mid \theta \in \mathcal{D},\ \theta \neq \lambda\,\}$ by the same argument as in the preceding case. Claim (1) follows since also the sequences in $\mathcal{E}$ are normal. Claim (2) follows from $\alpha \equiv \Sigma\mathcal{D}$, $\eta' \equiv \Sigma\mathcal{E}$, and the observation that whenever $\alpha \equiv \alpha_1 + \cdots + \alpha_m + \varepsilon$ and $\beta \equiv \beta_1 + \cdots + \beta_n$, we have $\alpha\beta \equiv \alpha_1\beta + \cdots + \alpha_m\beta + \beta \equiv \alpha_1\beta + \cdots + \alpha_m\beta + \beta_1 + \cdots + \beta_n$. ∎

Let $\eta = \alpha_1{:}\ldots{:}\alpha_n$. We use the notation $\langle\eta\rangle\varphi := \langle\alpha_1\rangle \ldots \langle\alpha_n\rangle\varphi$. The notation $[\eta]\varphi$ is defined analogously. We say a sequence η **is contained in a set of formulas** if so is the formula $\langle\eta\rangle\varphi$ or $[\eta]\varphi$ for some φ. We will now show that for every program sequence contained in $\mathcal{F}$ one can compute a DNF all of whose sequences are contained in $\mathcal{F}$.

4.2 Computing Program DNFs

We now present an algorithm that, given a program sequence η, computes a DNF of η. A key feature of the algorithm is that the computed DNFs are compatible with the syntactic closure conditions for formulas.

Definition 4.3 The **decomposition closure $\mathcal{A}\eta$ of a program sequence η** is the least set that contains η and is closed under the following rules:

$$\frac{\alpha_1 + \alpha_2{:}\eta}{\alpha_i{:}\eta}\ i \in \{1,2\} \qquad \frac{\alpha_1\alpha_2{:}\eta}{\alpha_1{:}\alpha_2{:}\eta} \qquad \frac{\alpha^*{:}\eta}{\eta} \qquad \frac{\alpha^*{:}\eta}{\alpha{:}\alpha^*{:}\eta}$$

The decomposition closure is compatible with the syntactic closure of formulas in the following sense:

Proposition 4.4 Let $\theta \in \mathcal{A}\eta$. Then for every formula φ, $\langle\theta\rangle\varphi$ is contained in the syntactic closure of $\langle\eta\rangle\varphi$ and $[\theta]\varphi$ is contained in the syntactic closure of $[\eta]\varphi$.

Proof It suffices to check that whenever $\frac{\eta}{\theta}$ is an instance of a derivation rule, $\langle\theta\rangle\varphi$ ($[\theta]\varphi$) is contained in the syntactic closure of $\langle\eta\rangle\varphi$ ($[\eta]\varphi$). We show the statement for $\langle\eta\rangle\varphi$ ($[\eta]\varphi$ follows analogously) by distinguishing four cases corresponding to the four derivation rules:

- $\frac{\alpha_1+\alpha_2{:}\eta}{\alpha_i{:}\eta}$ where $i \in \{1,2\}$. Then $\langle\alpha_1 + \alpha_2{:}\eta\rangle\varphi = \langle\alpha_1 + \alpha_2\rangle\langle\eta\rangle\varphi$, $\langle\alpha_1{:}\eta\rangle\varphi = \langle\alpha_i\rangle\langle\eta\rangle\varphi$, and the claim follows since both $\langle\alpha_1\rangle\langle\eta\rangle\varphi$ and $\langle\alpha_2\rangle\langle\eta\rangle\varphi$ are contained in the syntactic closure of $\langle\alpha_1 + \alpha_2\rangle\langle\eta\rangle\varphi$ (see Definition 2.3).
- $\frac{\alpha_1\alpha_2{:}\eta}{\alpha_1{:}\alpha_2{:}\eta}$. Then $\langle\alpha_1\alpha_2{:}\eta\rangle\varphi = \langle\alpha_1\alpha_2\rangle\langle\eta\rangle\varphi$, $\langle\alpha_1{:}\alpha_2{:}\eta\rangle\varphi = \langle\alpha_1\rangle\langle\alpha_2\rangle\langle\eta\rangle\varphi$, and the claim follows since $\langle\alpha_1\rangle\langle\alpha_2\rangle\langle\eta\rangle\varphi$ is contained in the syntactic closure of $\langle\alpha_1\alpha_2\rangle\langle\eta\rangle\varphi$.
- $\frac{\alpha^*{:}\eta}{\eta}$. Then $\langle\alpha^*{:}\eta\rangle\varphi = \langle\alpha^*\rangle\langle\eta\rangle\varphi$ and the claim follows since $\langle\eta\rangle\varphi$ is contained in the syntactic closure of $\langle\alpha^*\rangle\langle\eta\rangle\varphi$.
- $\frac{\alpha^*{:}\eta}{\alpha{:}\alpha^*{:}\eta}$. Then $\langle\alpha^*{:}\eta\rangle\varphi = \langle\alpha^*\rangle\langle\eta\rangle\varphi$, $\langle\alpha{:}\alpha^*{:}\eta\rangle\varphi = \langle\alpha\rangle\langle\alpha^*\rangle\langle\eta\rangle\varphi$, and the claim follows since $\langle\alpha\rangle\langle\alpha^*\rangle\langle\eta\rangle\varphi$ is contained in the syntactic closure of $\langle\alpha^*\rangle\langle\eta\rangle\varphi$.

■

Thus, for every η in $\mathcal{F}$ and every $\theta \in \mathcal{A}\eta$, θ is contained in $\mathcal{F}$. In particular, it follows that the decomposition closure of every program sequence is finite.

Given a program sequence η, we obtain a DNF of η by taking the normal sequences in the decomposition closure of η. In the rest of the section we will show that this approach is correct. We define:

$$\mathcal{A}_{\mathrm{n}}\eta := \{\, \theta \in \mathcal{A}\eta \mid \theta \text{ normal} \,\}$$

We show that $\mathcal{A}_n\eta$ is a DNF of η in two steps. First, we show that $\bigcup\{\,\mathcal{L}\theta \mid \theta \in \mathcal{A}_n\eta\,\} = \mathcal{L}\eta$ and thus, $\mathcal{A}_n\eta$ is a DNF of η if η is a single program. In the second step we generalize the result to arbitrary sequences.

So, our first task is to show $\bigcup\{\,\mathcal{L}\theta \mid \theta \in \mathcal{A}_n\eta\,\} = \mathcal{L}\eta$. The inclusion from left to right is immediate with the following lemma:

Lemma 4.5 If $\theta \in \mathcal{A}\eta$ then $\mathcal{L}\theta \subseteq \mathcal{L}\eta$.

Proof By induction on the derivation of $\theta \in \mathcal{A}\eta$. The claim is immediate if $\theta = \eta$. Otherwise, we distinguish four cases.

Let $\alpha_1 + \alpha_2{:}\eta' \in \mathcal{A}\eta$ and $\theta = \alpha_i{:}\eta'$ for $i \in \{1,2\}$. By the inductive hypothesis, we then have $\mathcal{L}(\alpha_1 + \alpha_2{:}\eta') \subseteq \mathcal{L}\eta$. The claim follows since $\mathcal{L}(\alpha_i{:}\eta') \subseteq \mathcal{L}(\alpha_1 + \alpha_2{:}\eta')$ (as $\mathcal{L}(\alpha_i{:}\eta') \subseteq \mathcal{L}(\alpha_1{:}\eta') \cup \mathcal{L}(\alpha_2{:}\eta') = \mathcal{L}(\alpha_1{:}\eta' + \alpha_2{:}\eta') = \mathcal{L}(\alpha_1 + \alpha_2{:}\eta')$).

Let $\alpha_1\alpha_2{:}\eta' \in \mathcal{A}\eta$ and $\theta = \alpha_1{:}\alpha_2{:}\eta'$. By the inductive hypothesis, we then obtain $\mathcal{L}(\alpha_1\alpha_2{:}\eta') \subseteq \mathcal{L}\eta$, and the claim follows since $\mathcal{L}(\alpha_1{:}\alpha_2{:}\eta') = \mathcal{L}(\alpha_1\alpha_2{:}\eta')$.

Let $\alpha^*{:}\eta' \in \mathcal{A}\eta$ and $\theta = \eta'$. By the inductive hypothesis, we have $\mathcal{L}(\alpha^*{:}\eta') \subseteq \mathcal{L}\eta$, and the claim follows since $\varepsilon \in \mathcal{L}(\alpha^*)$ and hence $\mathcal{L}(\eta') \subseteq \mathcal{L}(\alpha^*{:}\eta')$.

Let $\alpha^*{:}\eta' \in \mathcal{A}\eta$ and $\theta = \alpha{:}\alpha^*{:}\eta'$. By the inductive hypothesis, we have $\mathcal{L}(\alpha^*{:}\eta') \subseteq \mathcal{L}\eta$, and the claim follows since $\mathcal{L}(\alpha\alpha^*) \subseteq \mathcal{L}(\alpha^*)$ and hence $\mathcal{L}(\alpha{:}\alpha^*{:}\eta') = \mathcal{L}(\alpha\alpha^*) \cdot \mathcal{L}\eta' \subseteq \mathcal{L}(\alpha^*) \cdot \mathcal{L}\eta' = \mathcal{L}(\alpha^*{:}\eta')$. ∎

For the inclusion from right to left, observe that whenever $\theta \in \mathcal{A}\eta$ we have $\mathcal{A}\theta \subseteq \mathcal{A}\eta$ and $\mathcal{A}_n\theta \subseteq \mathcal{A}_n\eta$. Moreover:

Lemma 4.6 If $\theta \in \mathcal{A}\eta$ then $\theta{:}\eta' \in \mathcal{A}(\eta{:}\eta')$ for every sequence η'.

Proof By induction on the derivation of $\theta \in \mathcal{A}\eta$. If $\theta = \eta$, then $\theta{:}\eta' = \eta{:}\eta'$ and hence, trivially, $\theta{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. Otherwise, we distinguish four cases.

Let $\theta = \alpha_i{:}\theta'$ for some $i \in \{1,2\}$ and $\alpha_1 + \alpha_2{:}\theta' \in \mathcal{A}\eta$. By the inductive hypothesis, we then have $\alpha_1 + \alpha_2{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. Hence $\alpha_i{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$.

Let $\theta = \alpha_1{:}\alpha_2{:}\theta'$ where $\alpha_1\alpha_2{:}\theta' \in \mathcal{A}\eta$. By the inductive hypothesis, we then have $\alpha_1\alpha_2{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. Hence $\alpha_1{:}\alpha_2{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$.

Let $\theta = \theta'$ for some $\alpha^*{:}\theta' \in \mathcal{A}\eta$. By the inductive hypothesis, we have $\alpha^*{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. Hence $\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$.

Finally, let $\theta = \alpha{:}\alpha^*{:}\theta'$ where $\alpha^*{:}\theta' \in \mathcal{A}\eta$. By the inductive hypothesis, we have $\alpha^*{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. Hence $\alpha{:}\alpha^*{:}\theta'{:}\eta' \in \mathcal{A}(\eta{:}\eta')$. ∎

We are now ready to show the remaining inclusion.

Lemma 4.7 $\mathcal{L}\eta \subseteq \bigcup\{\,\mathcal{L}\theta \mid \theta \in \mathcal{A}_n\eta\,\}$

Proof Let $\eta = \alpha_1{:}\ldots{:}\alpha_n$. If $n = 0$, then $\mathcal{A}_n\eta = \{\eta\}$ and the claim is trivial. Otherwise, we show $\forall\sigma \in \mathcal{L}\eta\colon\ \sigma \in \mathcal{L}\theta$ for some $\theta \in \mathcal{A}_n\eta$ by lexicographic

induction on n and α_1. Let $\sigma \in \mathcal{L}\eta$. Then $\sigma = \sigma_1 \ldots \sigma_n$ such that $\sigma_i \in \mathcal{L}\alpha_i$ for every $i \in [1, n]$. We distinguish four cases depending on the shape of α_1.

Let $\alpha_1 = a$. Then the claim is trivial since $\mathcal{A}_\mathrm{n}(a{:}\alpha_2{:}\ldots{:}\alpha_n) = \{a{:}\alpha_2{:}\ldots{:}\alpha_n\}$.

Let $\alpha_1 = \beta + \gamma$. Then $\sigma \in \mathcal{L}(\beta{:}\alpha_2{:}\ldots{:}\alpha_n) \cup \mathcal{L}(\gamma{:}\alpha_2{:}\ldots{:}\alpha_n)$, and the claim follows by the inductive hypothesis since $\mathcal{A}_\mathrm{n}(\beta{:}\alpha_2{:}\ldots{:}\alpha_n), \mathcal{A}_\mathrm{n}(\gamma{:}\alpha_2{:}\ldots{:}\alpha_n) \subseteq \mathcal{A}_\mathrm{n}\eta$.

Let $\alpha_1 = \beta\gamma$. Then $\sigma_1 = \sigma_\beta\sigma_\gamma$ such that $\sigma_\beta \in \mathcal{L}\beta$ and $\sigma_\gamma \in \mathcal{L}\gamma$. Moreover, $\beta{:}\gamma{:}\alpha_2{:}\ldots{:}\alpha_n \in \mathcal{A}\eta$. We distinguish two subcases.

- Let $\sigma_\beta = \varepsilon$. By the inductive hypothesis, we have $\varepsilon \in \mathcal{L}\theta$ for some $\theta \in \mathcal{A}_\mathrm{n}\beta$. Consequently, since θ is normal, it is empty. Hence, $\gamma{:}\alpha_2{:}\ldots{:}\alpha_n \in \mathcal{A}\eta$ (Lemma 4.6). By the inductive hypothesis, we have $\sigma = \sigma_\gamma\sigma_2 \ldots \sigma_n \in \mathcal{L}\eta'$ for some $\eta' \in \mathcal{A}_\mathrm{n}(\gamma{:}\alpha_2{:}\ldots{:}\alpha_n)$. The claim follows.
- Let $\sigma_\beta = a\tau$. By the inductive hypothesis, we have $a\tau \in \mathcal{L}\theta$ for some $\theta \in \mathcal{A}_\mathrm{n}\beta$. Consequently, since θ is normal, it has the form $a{:}\beta_1{:}\ldots{:}\beta_m$. Then $\sigma = \sigma_\beta\sigma_\gamma\sigma_2 \ldots \sigma_n \in \mathcal{L}(a{:}\beta_1{:}\ldots{:}\beta_m{:}\gamma{:}\alpha_2{:}\ldots{:}\alpha_n)$. The claim follows with Lemma 4.6.

Finally, let $\alpha_1 = \beta^*$. We distinguish two subcases.

- Let $\sigma_1 = \varepsilon$. By the inductive hypothesis, $\sigma = \sigma_2 \ldots \sigma_n \in \mathcal{L}\theta$ for some $\theta \in \mathcal{A}_\mathrm{n}(\alpha_2{:}\ldots{:}\alpha_n)$. The claim follows since $\alpha_2{:}\ldots{:}\alpha_n \in \mathcal{A}\eta$.
- Let $\sigma_1 = a\tau$. W.l.o.g., $\tau = \tau_1\tau_2$ such that $a\tau_1 \in \mathcal{L}\beta$ and $\tau_2 \in \mathcal{L}\beta^*$. By the inductive hypothesis, we have $a\tau_1 \in \mathcal{L}\theta$ for some $\theta \in \mathcal{A}_\mathrm{n}\beta$. Consequently, since θ is normal, it has the form $a{:}\beta_1{:}\ldots{:}\beta_m$. Then $\sigma = a\tau_1\tau_2\sigma_2 \ldots \sigma_n \in \mathcal{L}(a{:}\beta_1{:}\ldots{:}\beta_m{:}\eta)$. The claim follows with Lemma 4.6 since $\beta{:}\eta \in \mathcal{A}\eta$. ∎

It remains to argue that $\mathcal{A}_\mathrm{n}\eta$ constitutes a DNF for every η. This follows by induction on the length of η with the following proposition:

Proposition 4.8

1. $\mathcal{A}\lambda = \{\lambda\}$.
2. If $\lambda \notin \mathcal{A}\alpha$ then $\mathcal{A}(\alpha{:}\eta) = \{\, \theta{:}\eta \mid \theta \in \mathcal{A}\alpha \,\}$.
3. If $\lambda \in \mathcal{A}\alpha$ then $\mathcal{A}(\alpha{:}\eta) = \{\, \theta{:}\eta \mid \theta \in \mathcal{A}\alpha,\ \theta \neq \lambda \,\} \cup \mathcal{A}\eta$.

Proof Claim (1) is immediate by the definition of $\mathcal{A}$. For (2) and (3), the inclusion from right to left follows with Lemma 4.6. For the other inclusion, we show that every $\theta \in \mathcal{A}(\alpha{:}\eta)$ is contained in $\{\, \theta{:}\eta \mid \theta \in \mathcal{A}\alpha \,\}$ ($\{\, \theta{:}\eta \mid \theta \in \mathcal{A}\alpha,\ \theta \neq \lambda \,\} \cup \mathcal{A}\eta$) by induction on the derivation of $\theta \in \mathcal{A}(\alpha{:}\eta)$. ∎

The presented method for computing program DNFs is related to Brzozowski's [10] derivative-based approach to constructing finite automata from regular expressions and its extension by Antimirov [2]. We conclude the

section by comparing the three approaches. While Brzozowski's construction yields deterministic automata, Antimirov's partial derivatives yield nondeterministic automata. Brzozowski establishes that the number of derivatives of every regular expression is finite modulo associativity, commutativity and idempotence of program union, from which he obtains finiteness of the resulting automata. This finiteness result is strengthened by Antimirov, who shows that the number of syntactically distinct partial derivatives of a regular expression is bounded by the length of the expression. Our method is particularly close to Antimirov's partial derivative computation in that it yields nondeterministic automata (a program DNF is allowed to contain more than one sequence that begins with the same action) and in that we obtain a strong finiteness result for our construction. In our case, however, the finiteness result directly follows from the finiteness of the syntactic closure [19]. Thus, our approach establishes a connection between Fischer and Ladner's finiteness result in [19] and Antimirov's bound in [2]. Since our approach extends to tests, it also generalizes some of Antimirov's results.

Despite the similarities, our approach has a number of important technical differences to Brzozowski and Antimirov. Their constructions iterate over all actions, computing a derivative (or a set of partial derivatives) for each action. Our approach, on the other hand, immediately computes normal programs, which can be seen as pairs of an action and a corresponding partial derivative. Actions whose (partial) derivatives denote the empty language are not considered in the first place, which may yield better performance on regular expressions with many different actions. At the same time, our approach is conceptually simpler than the algorithms of Brzozowski and Antimirov. In particular, our algorithm requires no auxiliary functions.

4.3 Support and Clausal DNFs

Definition 4.9 We write $C \rhd \varphi$ and say C **supports** φ if φ is derivable from the formulas in C with the following rules:

$$\frac{\varphi_i}{\varphi_1 \vee \varphi_2}\ i \in \{1,2\} \qquad \frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2}$$

$$\frac{\langle\eta\rangle\varphi}{\langle\alpha\rangle\varphi}\ \eta \text{ contained in a DNF of } \alpha \qquad \frac{[\eta_1]\varphi \ \ldots \ [\eta_n]\varphi}{[\alpha]\varphi}\ \{\eta_1,\ldots,\eta_n\} \text{ DNF of } \alpha$$

Note that, just as for K* (Definition 3.1), the premises of every rule are either literals or proper subformulas of the conclusion of the rule.

We need to show that our new definition of support has the right semantic properties, i.e., satisfies Propositions 3.2 and 3.3. To show both propositions we

need to establish a semantic equivalence between the premises and the conclusion of each of the above derivation rules. For conjunctions and disjunctions this is straightforward. For diamond and box formulas we need to lift the equivalence of program sequences given by the program DNF to the semantic equivalence of formulas.

Intuitively, action strings are a restricted class of programs. Hence, we can interpret them as relations on states in the same way we do it for programs. We capture this intuition formally by defining the relations $\xrightarrow{\sigma}_{\mathcal{M}}$ as follows:

$$v \xrightarrow{\varepsilon}_{\mathcal{M}} w \iff v = w$$
$$v \xrightarrow{a\sigma}_{\mathcal{M}} w \iff \exists u\colon\ v \xrightarrow{a}_{\mathcal{M}} u \text{ and } u \xrightarrow{\sigma}_{\mathcal{M}} w$$

Proposition 4.10 $v \xrightarrow{\sigma\tau}_{\mathcal{M}} w \iff \exists u\colon\ v \xrightarrow{\sigma}_{\mathcal{M}} u \text{ and } u \xrightarrow{\tau}_{\mathcal{M}} w$

Proof Straightforward induction on σ. ∎

Consistently with our interpretation of program sequences, given some $\eta = \alpha_1{:}\ldots{:}\alpha_n$, we define $\xrightarrow{\eta}_{\mathcal{M}}$ such that $\xrightarrow{\eta}_{\mathcal{M}} = \{\,(w,w) \mid w \in |\mathcal{M}|\,\}$ if $n = 0$ and $\xrightarrow{\eta}_{\mathcal{M}} = \xrightarrow{\alpha_1\ldots\alpha_n}_{\mathcal{M}}$ otherwise. The correspondence between the transition relation of a program sequence η and the interpretation of strings in $\mathcal{L}\eta$ is captured by the following proposition.

Proposition 4.11

1. $v \xrightarrow{\eta}_{\mathcal{M}} w \iff \exists \sigma \in \mathcal{L}\eta\colon\ v \xrightarrow{\sigma}_{\mathcal{M}} w$
2. $\mathcal{M}, v \vDash \langle\eta\rangle\varphi \iff \exists \sigma \in \mathcal{L}\eta\, \exists w\colon\ v \xrightarrow{\sigma}_{\mathcal{M}} w \text{ and } \mathcal{M}, w \vDash \varphi$
3. $\mathcal{M}, v \vDash [\eta]\varphi \iff \forall \sigma \in \mathcal{L}\eta\, \forall w\colon\ v \xrightarrow{\sigma}_{\mathcal{M}} w \text{ implies } \mathcal{M}, w \vDash \varphi$

Proof Claims (2) and (3) follow from (1) by the definition of satisfaction. For (1), let $\eta = \alpha_1{:}\ldots{:}\alpha_n$. Both directions of the equivalence are immediate if $n = 0$. Otherwise, the claims are shown by lexicographic induction on n and α_1, using a case distinction on the shape of α_1. For the direction from left to right, we assume $v \xrightarrow{\eta}_{\mathcal{M}} w$ and show the existence of $\sigma \in \mathcal{L}\eta$ such that $v \xrightarrow{\sigma}_{\mathcal{M}} w$. For the other direction, we assume $\sigma \in \mathcal{L}\eta$ and $v \xrightarrow{\sigma}_{\mathcal{M}} w$ and show $v \xrightarrow{\eta}_{\mathcal{M}} w$. The details are straightforward. ∎

With Proposition 4.11 we can finally show the desired equivalences for diamond and box formulas:

Proposition 4.12 Let $\mathcal{D}$ be a DNF of η. Then:

1. $\mathcal{M}, w \vDash \langle\eta\rangle\varphi \iff \exists \theta \in \mathcal{D}\colon\ \mathcal{M}, w \vDash \langle\theta\rangle\varphi$
2. $\mathcal{M}, w \vDash [\eta]\varphi \iff \forall \theta \in \mathcal{D}\colon\ \mathcal{M}, w \vDash [\theta]\varphi$

Proof The claims follow with Propositions 4.11 and 4.10. ∎

Proposition 3.2 now follows from Proposition 4.12 and the semantic equivalences for conjunctions and disjunctions by induction on the derivation of support. Proposition 3.3 is shown similarly.

The definition of DNFs for clauses remains the same as in §3.3 (Definition 3.8; now w.r.t. the new definition of support). Thus, Proposition 3.10 remains valid. Note that one could restrict the rules for $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ to programs α that are not actions without changing the induced support relation. The restricted rules can be turned into tableau rules for computing DNFs following the approach in §3.4.

Remark 4.13 One may wonder why we do not base support on the following set of rules that are directly induced by the equivalences in §2.

$$\frac{\varphi_i}{\varphi_1 \vee \varphi_2}\ i \in \{1,2\} \qquad \frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2} \qquad \frac{\langle\alpha_i\rangle}{\langle\alpha_1 + \alpha_2\rangle\varphi}\ i \in \{1,2\} \qquad \frac{[\alpha_1]\varphi \quad [\alpha_2]\varphi}{[\alpha_1 + \alpha_2]\varphi}$$

$$\frac{\langle\alpha_1\rangle\langle\alpha_2\rangle\varphi}{\langle\alpha_1\alpha_2\rangle\varphi} \qquad \frac{[\alpha_1][\alpha_2]\varphi}{[\alpha_1\alpha_2]\varphi} \qquad \frac{\varphi}{\langle\alpha^*\rangle\varphi} \qquad \frac{\langle\alpha\rangle\langle\alpha^*\rangle\varphi}{\langle\alpha^*\rangle\varphi} \qquad \frac{\varphi \quad [\alpha][\alpha^*]\varphi}{[\alpha^*]\varphi}$$

The problem with the rules is that the induced notion of support is too weak to ensure that every satisfiable formula is supported by a satisfiable normal clause (Proposition 3.3). For instance, consider $\varphi = [a^{**}]p$. It is easy to see that φ is not derivable by the above rules from any normal clause and hence has no support. One consequence of this is that we can no longer guarantee that every clause has a DNF. Consider, for instance, $C = \{\varphi\}$. Since φ is not supported by any normal clause, neither is C. Hence, by Proposition 3.10, the only possible DNF of C is $\emptyset$. This, however, contradicts condition (1) for DNFs since C is satisfiable. □

4.4 Demo Sets

We now show how to adapt the definition of demo sets and the relevant proofs from §3.2 to account for regular programs and the new notion of support. Let $\xrightarrow{a}_S$ be defined as before. We extend the definition to complex programs as follows:

$$\xrightarrow{\alpha+\beta}_S \;=\; \xrightarrow{\alpha}_S \cup \xrightarrow{\beta}_S \qquad \xrightarrow{\alpha\beta}_S \;=\; \xrightarrow{\alpha}_S \circ \xrightarrow{\beta}_S \qquad \xrightarrow{\alpha^*}_S \;=\; \xrightarrow{\alpha}_S{}^*$$

The notions of transition completeness and demo sets (Definition 3.4) adapt without changes to their formulation. Also, the definition of $\mathcal{M}_S$ remains unchanged. Note that since

$\xrightarrow{a}_{\mathcal{M}_S} = \xrightarrow{a}_S$ for every action a, we obtain $\xrightarrow{\alpha}_{\mathcal{M}_S} = \xrightarrow{\alpha}_S$ for every program α. We define the relations $\xrightarrow{\sigma}_S$ and $\xrightarrow{\eta}_S$ from $\xrightarrow{\alpha}_S$ analogously to how $\xrightarrow{\sigma}_{\mathcal{M}}$ and $\xrightarrow{\eta}_{\mathcal{M}}$ are defined from $\xrightarrow{\alpha}_{\mathcal{M}}$ and obtain $\xrightarrow{\sigma}_S = \xrightarrow{\sigma}_{\mathcal{M}_S}$ and $\xrightarrow{\eta}_S = \xrightarrow{\eta}_{\mathcal{M}_S}$.

To obtain Lemma 3.6, we need an additional step:

Lemma 4.14 Let S be a set of clauses, $\{C, D\} \subseteq S$, $C \rhd [\eta]\varphi$, and $C \xrightarrow{\sigma}_S D$ for some $\sigma \in \mathcal{L}\eta$. Then $D \rhd \varphi$.

Proof We proceed by lexicographic induction on the length of σ and the derivation of $C \rhd [\eta]\varphi$. We do a case distinction on η. If $\eta = \lambda$ the claim is immediate. Otherwise, η has the form $\alpha_1{:}\eta'$. We do a case distinction on the shape of α_1. If α_1 is an action, then $[\eta]\varphi \in C$, σ has the form $a\sigma'$, and the claim follows by the inductive hypothesis for σ'. If α_1 is not an action, then there is some DNF $\mathcal{D}$ of α_1 and some $\theta \in \mathcal{D}$ such that $\sigma \in \mathcal{L}(\theta{:}\eta')$ and $C \rhd [\theta][\eta']\varphi$. The claim follows by the inductive hypothesis for σ and $[\theta][\eta']\varphi$. ∎

From this and Proposition 4.11 (1), we obtain Lemma 3.6 in its original formulation. Also, the formulation and the proof of Lemma 3.7 remain unchanged.

Since $\xrightarrow{\sigma}_S = \xrightarrow{\sigma}_{\mathcal{M}_S}$ and $\xrightarrow{\eta}_S = \xrightarrow{\eta}_{\mathcal{M}_S}$, Proposition 4.11 (1) holds for $\xrightarrow{\eta}_S$ and $\xrightarrow{\sigma}_S$. With this, Lemma 3.5 adapts as follows:

Lemma 4.15 A clause set S is transition-complete if and only if for every $\langle\eta\rangle\varphi \in C \in S$ there is some $D \in S$ such that $C \xrightarrow{\eta}_S D$ and $D \rhd \varphi$.

Proof For the direction from left to right, we show the following generalization. If S is transition-complete, $C \in S$, and $C \rhd \langle\eta\rangle\varphi$, then there is some $D \in S$ such that $C \xrightarrow{\eta}_S D$ and $D \rhd \varphi$. The strengthened claim easily follows by induction on the length of η.

For the other direction, we show that for every $C \in S$ and every formula $\langle\alpha\rangle\varphi$ such that $C \rhd \langle\alpha\rangle\varphi$ there is some D such that $C \xrightarrow{\alpha}_S D$ and $D \rhd \varphi$ by case analysis on the shape of α. If α is an action, we have $\langle\alpha\rangle\varphi \in C$ and the claim is immediate by the assumption. Otherwise, we have $C \rhd \langle\theta\rangle\varphi$ for some normal sequence θ. If θ is empty, the claim follows with Proposition 4.11 (1) for $\xrightarrow{\eta}_S$ since $C \xrightarrow{\varepsilon}_S C$ and $C \rhd \varphi$. Otherwise, we have $\langle\theta\rangle\varphi \in C$ and the claim follows with Proposition 4.11 (1) for $\xrightarrow{\eta}_S$ from the assumption. ∎

4.5 Demo Graphs

What happens to clause graphs and demo graphs? To answer this, we first need to provide a new definition of links.

Definition 4.16 A **link** is a tuple $C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D$ such that:

1. C and D are normal clauses,
2. φ is not a diamond formula,
3. $\langle a\rangle\langle\eta\rangle\varphi \in C$,
4. θ is an element of a DNF of η,
5. D is an element of a DNF of $C^{\Box}_{a};\langle\theta\rangle\varphi$.

Example 4.17

- Let $C = \{\langle a\rangle(p_1 \vee p_2), [a]q, r\}$. The tuples $C\binom{\langle a\rangle(p_1\vee p_2)}{p_1\vee p_2}\{p_1, q\}$ and $C\binom{\langle a\rangle(p_1\vee p_2)}{p_1\vee p_2}\{p_2, q\}$ are links according to the above definition, where $\varphi = p_1 \vee p_2$ and $\eta = \theta = \lambda$.
- The tuples $\{\langle a^+\rangle p\}\binom{\langle a^+\rangle p}{p}\{p\}$ and $\{\langle a^+\rangle p\}\binom{\langle a^+\rangle p}{\langle a^+\rangle p}\{\langle a^+\rangle p\}$ are links. We have $\varphi = p$, $\eta = a^*$, and $\theta \in \{\lambda, a{:}a^*\}$ ($\{\lambda, a{:}a^*\}$ is a DNF of a^*).
- Let $C = \{\langle a\rangle\langle(b{+}c)d^*\rangle\langle e\rangle p\}$. Then the tuples $C\binom{\langle a\rangle\langle(b+c)d^*\rangle\langle e\rangle p}{\langle b\rangle\langle d^*\rangle\langle e\rangle p}\{\langle b\rangle\langle d^*\rangle\langle e\rangle p\}$ and $C\binom{\langle a\rangle\langle(b+c)d^*\rangle\langle e\rangle p}{\langle c\rangle\langle d^*\rangle\langle e\rangle p}\{\langle c\rangle\langle d^*\rangle\langle e\rangle p\}$ are links. We have $\varphi = p$, $\eta = (b + c)d^*{:}e$, and $\theta \in \{b{:}d^*{:}e, c{:}d^*{:}e\}$ ($\{b{:}d^*{:}e, c{:}d^*{:}e\}$ is a DNF of $(b + c)d^*{:}e$). □

We call a link **delegating** if it has the form $C\binom{\langle a\rangle\varphi}{\langle b\rangle\psi}D$ and **fulfilling** if it has the form $C\binom{\langle a\rangle\varphi}{\psi}D$ where ψ is not a diamond formula. Clause graphs are defined as in §3.5 relatively to the new definition of links.

Clearly, Proposition 3.14 holds for the new definition of links. Functionality and realization are defined as before. A **path for $\langle a\rangle\varphi$ in $\mathcal{G}$** is now a nonempty sequence

$$(C_1\binom{\langle a_1\rangle\varphi_1}{\langle a_2\rangle\varphi_2}C_2)(C_2\binom{\langle a_2\rangle\varphi_2}{\langle a_3\rangle\varphi_3}C_3)\ldots(C_{n-2}\binom{\langle a_{n-2}\rangle\varphi_{n-2}}{\langle a_{n-1}\rangle\varphi_{n-1}}C_{n-1})(C_{n-1}\binom{\langle a_{n-1}\rangle\varphi_{n-1}}{\psi}C_n)$$

of links in $\mathcal{G}$ such that $\langle a_1\rangle\varphi_1 = \langle a\rangle\varphi$. A path of the above form is called a **run for $\langle a\rangle\varphi$ in C** if $C_1 = C$ and ψ is not a diamond formula (i.e., if the last link of the path is fulfilling). The path is called a **loop for $\langle a\rangle\varphi$** if $C_1 = C_n$ and $\psi = \langle a_1\rangle\varphi_1 = \langle a\rangle\varphi$.

As before (Definition 3.15), a clause graph $\mathcal{G}$ is a demo graph if $\mathcal{G}$ is finite, functional, realizes every diamond formula, and contains no loops. Proposition 3.16 is adapted as follows.

Proposition 4.18 Let $\mathcal{G}$ be a demo graph and $\langle a\rangle\varphi \in C \in \mathcal{G}$. Then $\mathcal{G}$ contains a unique run for $\langle a\rangle\varphi$ in C.

Proof For existence of runs, suppose $\mathcal{G}$ contains no run for $\langle a\rangle\varphi$ in C. To obtain a contradiction, it suffices to show that $\mathcal{G}$ has a path for $\langle a\rangle\varphi$ that originates in C and has length n for every $n \in \mathbb{N}$. Since both $\mathcal{G}$ and $\mathcal{F}$ are finite, and since

$\mathcal{G}$ contains no run for $\langle a\rangle\varphi$ in C, $\mathcal{G}$ then has a path that contains a fragment $(C_1\binom{\langle a_1\rangle\varphi_1}{\langle a_2\rangle\varphi_2}C_2)(C_{m-1}\binom{\langle a_{m-1}\rangle\varphi_{m-1}}{\langle a_m\rangle\varphi_m}C_m)$ such that $C_1 = C_m$ and $\langle a_1\rangle\varphi_1 = \langle a_m\rangle\varphi_m$, contradicting the assumption that $\mathcal{G}$ has no loops.

The existence of arbitrarily long paths for $\langle a\rangle\varphi$ from C is argued inductively. The existence of a path of length 1 is immediate by the assumption that $\langle a\rangle\varphi$ realizes every diamond formula. Now suppose $\mathcal{G}$ has a path for $\langle a\rangle\varphi$ of length n originating in C, and let $C_{n-1}\binom{\langle b\rangle\langle\eta\rangle\psi}{\langle\theta\rangle\psi}C_n$ be the last link of the path, where ψ is not a diamond formula. Since $\mathcal{G}$ contains no run for $\langle a\rangle\varphi$ in C, $\langle\theta\rangle\psi$ must be a diamond formula, and hence θ is nonempty. Moreover, since θ is contained in a DNF of η, θ is normal. Hence, θ has the form $c{:}\theta'$, i.e., $\langle\theta\rangle\psi = \langle c\rangle\langle\theta'\rangle\psi$. In particular, $\langle\theta\rangle\psi$ is a literal, and thus contained in C_n (since C_n comes from a DNF of $C_{n-1}{}^{\Box}_{b}\,;\langle\theta\rangle\psi$). Since $\mathcal{G}$ realizes every diamond formula, there is a link of the form $C_n\binom{\langle c\rangle\langle\theta'\rangle\psi}{\psi'}D$, and hence a path for $\langle a\rangle\varphi$ in C of length $n+1$.

Uniqueness once again follows from the assumed functionality of $\mathcal{G}$. Suppose $\langle a\rangle\varphi$ in C has two distinct runs, i.e., two runs that differ in at least one link. For the first distinct pair of links $C_1\binom{\varphi_1}{\psi_1}D_1$ and $C_2\binom{\varphi_2}{\psi_2}D_2$ in the two runs we must have $C_1 = C_2$ and $\varphi_1 = \varphi_2$ (since the preceding fragments of the two runs are identical). Hence, we have $\varphi_1 \neq \varphi_2$ or $D_1 \neq D_2$, contradicting functionality. ∎

The clauses of a demo graph form a demo set (Proposition 3.17), which is now shown via Lemma 4.15 in place of Lemma 3.5. The auxiliary condition of Lemma 4.15 is established by induction on the length of runs in the demo graph with the following lemma:

Lemma 4.19 If $\mathcal{D}$ is a DNF of η and $\theta \in \mathcal{D}$, then $\langle\theta\rangle\varphi \rhd \langle\eta\rangle\varphi$ for all φ.

Proof By induction on the length of η. If $\eta = \lambda$, then $\theta = \lambda$. Hence $\langle\theta\rangle\varphi = \langle\eta\rangle\varphi = \varphi$ and the claim is immediate.

If $\eta = \alpha{:}\eta'$ for some α and (possibly empty) η', then $\langle\eta\rangle\varphi = \langle\alpha\rangle\langle\eta'\rangle\varphi$ and $\mathcal{D} = \{\,\eta_1{:}\eta' \mid \eta_1 \in \mathcal{E}_1,\ \eta_1 \neq \lambda\,\} \cup \mathcal{E}_2$ where $\mathcal{E}_1$ is a DNF of α and $\mathcal{E}_2$ is a DNF of η' if $\lambda \in \mathcal{E}_1$, and otherwise $\mathcal{E}_2 = \emptyset$. We distinguish two cases. If $\theta \in \{\,\eta_1{:}\eta' \mid \eta_1 \in \mathcal{E}_1,\ \eta_1 \neq \lambda\,\}$, then $\langle\theta\rangle\varphi = \langle\eta_1\rangle\langle\eta'\rangle\varphi$ where $\eta_1 \in \mathcal{E}_1$, and the claim follows by the definition of support.

If $\theta \in \mathcal{E}_2$, then $\lambda \in \mathcal{E}_1$, and hence, by the definition of support, it suffices to show $\langle\theta\rangle\varphi \rhd \langle\eta'\rangle\varphi$ (since $\langle\eta'\rangle\varphi = \langle\lambda\rangle\langle\eta'\rangle\varphi$). The claim $\langle\theta\rangle\varphi \rhd \langle\eta'\rangle\varphi$ is immediate by the inductive hypothesis since $\mathcal{E}_2$ is a DNF of η'. ∎

4.6 Decision Procedure

The decision procedure for PDL is defined in Fig. 5. The overall working principle of the procedure remains the same as for K*. The main change compared

Input: a normal clause $C_0 \subseteq \mathcal{F}$

Variable: $\mathcal{G} := \{C_0\}$

Invariant: $\mathcal{G}$ functional and loop-free

while $\mathcal{G}$ does not realize every $\langle a\rangle\varphi \in C \in \mathcal{G}$ **do**

1. **pick** $\langle a\rangle\langle\eta\rangle\varphi \in C \in \mathcal{G}$ such that
   - φ is not a diamond formula
   - $\mathcal{G}$ does not realize $\langle a\rangle\langle\eta\rangle\varphi$ in C
2. **pick** DNF $\mathcal{D}$ of η
3. **choose** $\theta \in \mathcal{D}$
4. **pick** DNF S of $C_a^{\Box}; \langle\theta\rangle\varphi$
5. **if** $S \neq \emptyset$ **then choose** $D \in S$ **else backtrack**
6. $\mathcal{G} := \mathcal{G}; D; C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi} D$
7. **if** $\mathcal{G}$ contains a loop **then backtrack**

Return: – **satisfiable** if while-loop terminates regularly
– **unsatisfiable** if while-loop terminates through backtracking

Figure 5: Decision procedure for test-free PDL

to the procedure for K* is the use of a program DNF instead of the diamond decomposition in the selection of the target formula for a link.

The termination of the procedure follows by the same argument as for K* assuming that DNFs for programs and clauses do not leave the formula universe.

As before, a successful termination of the while loop implies that $\mathcal{G}$ is a demo graph containing the initial clause C_0, which in turn implies that C_0 is satisfiable. To argue that the procedure succeeds on all satisfiable inputs, we need to generalize our semantic argument for K*. We begin with the notion of distance, which we generalize as follows.

Let $\mathcal{M}$ be a model, C a clause, η a program sequence, and φ a formula. We define the **η-distance from C to φ in $\mathcal{M}$** as follows:

$$\delta^{\eta}_{\mathcal{M}} C\varphi := \min\{\,|\sigma| \mid \sigma \in \mathcal{L}\eta \text{ and } \exists v, w\colon\ \mathcal{M}, v \vDash C \text{ and } v \xrightarrow{\sigma}_{\mathcal{M}} w \text{ and } \mathcal{M}, w \vDash \varphi\,\}$$

where we write $|\sigma|$ for the length of σ. Proposition 3.18 generalizes as follows.

Proposition 4.20 $\delta^{\eta}_{\mathcal{M}} C\varphi < \infty$ if and only if $\mathcal{M}$ satisfies $C; \langle\eta\rangle\varphi$.

We no longer need to distinguish between delegating and fulfilling links for link satisfaction. We say a model $\mathcal{M}$ **satisfies a link** $C\binom{\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D$ (where φ is not a diamond formula) if $\delta^{\eta}_{\mathcal{M}}C\varphi > \delta^{\theta}_{\mathcal{M}}D\varphi$. Note that every fulfilling link $C\binom{\langle a\rangle\varphi}{\varphi}D$ between two clauses C, D that are satisfied by a model $\mathcal{M}$ is also satisfied by $\mathcal{M}$ since in every such case $\delta^{a}_{\mathcal{M}}C\varphi = 1 > 0 = \delta^{\lambda}_{\mathcal{M}}D\varphi$. A model **satisfies a graph** if it satisfies all of its clauses and links.

Proposition 3.19 can be shown to hold in its original form with minor adaptations to the proof. Lemma 3.20 is reformulated as follows.

Lemma 4.21 Let $\mathcal{G}$ be a graph satisfied by a model $\mathcal{M}$ and let $\langle a\rangle\langle\eta\rangle\varphi \in C \in \mathcal{G}$ where φ is not a diamond formula. Then for every DNF $\mathcal{D}$ of η there is some $\theta \in \mathcal{D}$ such that for every DNF S of $C^{\Box}_{a}\,;\langle\theta\rangle\varphi$ there is some $D \in S$ such that $\mathcal{M}$ satisfies $\mathcal{G}\,; D\,; C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D$.

The lemma is shown by a straightforward adaptation of the argument for Lemma 3.20. We conclude that the procedure in Fig. 5 decides the satisfiability of normal clauses for test-free PDL.

5 Tests

What happens when we add tests? As it turns out, tests require no fundamental changes to our approach. However, many arguments and algorithms need to be generalized. The main technical difficulty in adding tests concerns program DNFs. Since programs are no longer ordinary regular expressions, the regular string semantics underlying our definitions and algorithms for test-free PDL needs to be generalized. The changes have an impact on the definition of support and clausal DNFs. Moreover, since in the presence of tests the definitions of formulas and programs become mutually recursive, we need to generalize our proof of demo satisfiability. Rather than repeating the entire development in §4 for the case with tests, we only detail the changes to the setup of §4 that are necessary to deal with tests.

5.1 Program Equivalence

Since regular languages do not account for tests, we define a more general semantics. This semantics is an adaptation of the language-theoretic model of Kleene algebras with tests [39], initially introduced independently of Kleene algebras with tests by Kaplan [36].

A **guarded string** is a finite sequence $C_0a_1C_1 \ldots a_nC_n$ where $n \geq 0$, $a_1, \ldots, a_n$ are actions, and $C_0, C_1, \ldots, C_n$ are clauses. The clauses of a guarded string represent constraints on states that are expressed by tests. The letters σ and τ

now range over guarded strings and ω ranges over partial guarded strings that may be empty, have the form $a_1C_1 \ldots a_nC_n$ (when following a guarded string) or the form $C_1a_1 \ldots C_na_n$ (when preceding a guarded string). The **length $|\sigma|$ of a guarded string** $\sigma = C_0a_1C_1 \ldots a_nC_n$ is n. A language is now a set of guarded strings. Let L, L' be languages and C be a clause. We define the following notation:

$$\begin{aligned} \lceil C \rceil &:= \{ D \mid D \text{ a clause}, C \subseteq D \} \\ L \cdot L' &:= \{ \omega C \omega' \mid \omega C \in L,\ C\omega' \in L' \} \\ L^0 &:= \lceil \emptyset \rceil \end{aligned}$$

The notations L^{n+1} and L^* are defined from L^0 and $L \cdot L'$ as before. To every program α we assign a **language $\mathcal{L}\alpha$** as follows:

$$\begin{aligned} \mathcal{L}a &:= \{ CaD \mid C, D \text{ clauses} \} & \mathcal{L}(\alpha + \beta) &:= \mathcal{L}\alpha \cup \mathcal{L}\beta \\ \mathcal{L}\varphi &:= \lceil \{\varphi\} \rceil & \mathcal{L}(\alpha\beta) &:= \mathcal{L}\alpha \cdot \mathcal{L}\beta \\ & & \mathcal{L}(\alpha^*) &:= (\mathcal{L}\alpha)^* \end{aligned}$$

As before, we write $\alpha \equiv \beta$ if $\mathcal{L}\alpha = \mathcal{L}\beta$.

Example 5.1

- $\mathcal{L}(pq) = \mathcal{L}p \cdot \mathcal{L}q = \lceil \{p\} \rceil \cap \lceil \{q\} \rceil = \lceil \{p, q\} \rceil = \{ C \mid \{p, q\} \subseteq C \}$
  Note in particular the equality $\mathcal{L}p \cdot \mathcal{L}q = \lceil \{p\} \rceil \cap \lceil \{q\} \rceil$. It stems from the fact that for languages consisting entirely of strings of length 0 language concatenation reduces to intersection.
- $\mathcal{L}((pa)^*\neg p) = \{ C_0aC_1 \ldots aC_n \mid n \geq 0,\ p \in C_0 \cap \cdots \cap C_{n-1},\ \neg p \in C_n \}$ □

In the absence of tests, we defined a DNF of a program α as a finite set of normal program sequences $\{\eta_1, \ldots, \eta_n\}$ such that $\alpha \equiv \eta_1 + \cdots + \eta_n$. With tests, this definition is no longer adequate since programs with tests cannot generally be represented as sets of normal programs (e.g., there is no set of normal programs $\{\eta_1, \ldots, \eta_n\}$ such that $p \equiv \eta_1 + \cdots + \eta_n$). We solve the problem by generalizing the notion of a program DNF.

We define a **guarded sequence** as a pair $C\eta$ where C is a clause and η a program sequence. We call a guarded sequence $C\eta$ **normal** if η is normal. A guarded sequence $C\eta$ is **contained in a set of formulas** A if at least one of the following conditions holds:

- There is some φ such that $\langle\eta\rangle\varphi \in A$ and $C \subseteq A$.
- There is some φ such that $[\eta]\varphi \in A$ and $\{ \sim\psi \mid \psi \in C \} \subseteq A$.

We define $\mathcal{L}(C\eta) := \lceil C \rceil \cdot \mathcal{L}\eta$. Note that for every program sequence $\alpha_1{:}\ldots{:}\alpha_m$ and formulas $\varphi_1, \ldots, \varphi_n$ we have $\mathcal{L}(\{\varphi_1, \ldots, \varphi_n\}\alpha_1{:}\ldots{:}\alpha_m) =$

$\mathcal{L}(\varphi_1{:}\ldots{:}\varphi_n{:}\alpha_1{:}\ldots{:}\alpha_m)$. Thus, when written in a place where one would normally expect a program sequence or a program, a guarded sequence can be interpreted accordingly.

The definition of DNFs for programs and program sequences is generalized as follows.

Definition 5.2 A **DNF of a program** α is a finite set $\{C_1\eta_1, \ldots, C_n\eta_n\}$ of normal guarded sequences such that:

1. $\alpha \equiv C_1\eta_1 + \cdots + C_n\eta_n$.
2. Every formula from $C_1 \cup \cdots \cup C_n$ is contained in α.

A **DNF of a program sequence** η is a set $\mathcal{D}$ of guarded program sequences such that $\eta/\mathcal{D}$ is derivable by the following rules:

$$\frac{}{\lambda/\{\emptyset\lambda\}} \qquad \frac{\mathcal{D} \text{ DNF of } \alpha}{\alpha/\mathcal{D}}$$

$$\frac{\alpha/\mathcal{D} \qquad \eta/\mathcal{E}}{\alpha{:}\eta/\{\, C(\theta{:}\eta) \mid C\theta \in \mathcal{D},\ \theta \neq \lambda \,\} \cup \{\, (C \cup D)\theta \mid C\lambda \in \mathcal{D},\ D\theta \in \mathcal{E} \,\}}$$

As in the test-free case, we easily obtain that whenever $\{C_1\eta_1, \ldots, C_n\eta_n\}$ is a DNF of η, all of the guarded sequences in $\{C_1\eta_1, \ldots, C_n\eta_n\}$ are normal and $\eta \equiv C_1\eta_1 + \cdots + C_n\eta_n$.

Example 5.3 Let $\eta = pa^*{:}qb^*$ and let $\mathcal{D} = \{\{p\}a{:}a^*{:}qb^*, \{p,q\}\lambda, \{p,q\}b{:}b^*\}$. We can show that $\mathcal{D}$ is a DNF of η as follows. First, we observe that $\{\{p\}\lambda, \{p\}a{:}a^*\}$ is a DNF of pa^* and $\{\{q\}\lambda, \{q\}b{:}b^*\}$ is a DNF of qb^*. Hence $pa^*/\{\{p\}\lambda, \{p\}a{:}a^*\}$ and $qb^*/\{\{q\}\lambda, \{q\}b{:}b^*\}$ by the second derivation rule. Consequently

$$\eta = pa^*{:}qb^*/\{\{p\}a{:}a^*{:}qb^*\} \cup \{\{p,q\}\lambda, \{p,q\}b{:}b^*\} = \mathcal{D}$$

by the third derivation rule. Also note that for every set $\mathcal{E}$ such that $\eta/\mathcal{E}$ we have

$$\eta{:}\lambda/\{\, C\theta{:}\lambda \mid C\theta \in \mathcal{E}, \theta \neq \lambda \,\} \cup \{\, (C \cup \emptyset)\lambda \mid C\lambda \in \mathcal{E} \,\} = \mathcal{E}$$

since $\lambda/\{\emptyset\lambda\}$. □

We will now show that for every program sequence we can compute a DNF in a way that is compatible with the syntactic closure.

5.2 Computing Program DNFs

Our approach for computing program DNFs in the presence of tests directly generalizes our approach in the test-free case. The decomposition closure is adapted as follows.

Definition 5.4 The **decomposition closure $\mathcal{A}\eta$ of a program sequence η** is the least set of guarded sequences that contains $\emptyset\eta$ and is closed under the following rules:

$$\frac{C(\varphi{:}\eta)}{(C;\varphi)\eta} \qquad \frac{C(\alpha_1+\alpha_2{:}\eta)}{C(\alpha_i{:}\eta)}\ i\in\{1,2\} \qquad \frac{C(\alpha_1\alpha_2{:}\eta)}{C(\alpha_1{:}\alpha_2{:}\eta)} \qquad \frac{C(\alpha^*{:}\eta)}{C\eta} \qquad \frac{C(\alpha^*{:}\eta)}{C(\alpha{:}\alpha^*{:}\eta)}$$

By the following generalization of Proposition 4.4, decomposition closure is compatible with the syntactic closure and property (2) of program DNFs.

Proposition 5.5 Let $C\theta \in \mathcal{A}\eta$ and let φ be a formula. Then:

1. Every formula in C is contained (as a subformula) in η.
2. Every formula in $C;\langle\theta\rangle\varphi$ is contained in the syntactic closure of $\langle\eta\rangle\varphi$.
3. Every formula in $\{\sim\psi \mid \psi\in C\};[\theta]\varphi$ is contained in the syntactic closure of $[\eta]\varphi$.

The proposition is shown similarly to Proposition 4.4.

As in the test-free case, one can show that the normal guarded sequences in the decomposition closure of a program sequence η constitute a DNF of η. The proof is a straightforward generalization of the argument in §4.2.

5.3 Support and Clausal DNFs

Definition 5.6 We write $C \rhd \varphi$ and say C **supports** φ if φ is derivable from the formulas in C with the rules for conjunctions and disjunctions from Definition 4.9 together with the following two rules:

$$\frac{\psi_1 \ \ldots\ \psi_n\ \ \langle\eta\rangle\varphi}{\langle\alpha\rangle\varphi}\ \{\psi_1,\ldots,\psi_n\}\eta \text{ contained in a DNF of } \alpha$$

$$\frac{\psi_1 \ \ldots\ \psi_n}{[\alpha]\varphi}\ \{C_1\eta_1,\ldots,C_n\eta_n\} \text{ DNF of } \alpha,\ \psi_i\in\{\sim\psi \mid \psi\in C_i\};[\eta_i]\varphi$$

Once again, we need to show the validity of Propositions 3.2 and 3.3. This can be done by suitably extending the corresponding argument in §4.3. We begin by

adapting the definition of $\xrightarrow{\sigma}_{\mathcal{M}}$ to guarded strings:

$$v \xrightarrow{C}_{\mathcal{M}} w \iff v = w \text{ and } \mathcal{M}, v \models C$$
$$v \xrightarrow{Ca\sigma}_{\mathcal{M}} w \iff \mathcal{M}, v \models C \text{ and } \exists u\colon v \xrightarrow{a}_{\mathcal{M}} u \text{ and } u \xrightarrow{\sigma}_{\mathcal{M}} w$$

By the definition, the clauses of a guarded string σ act as constraints on the states of a model $\mathcal{M}$ connected by $\xrightarrow{\sigma}_{\mathcal{M}}$. In particular, we observe:

Proposition 5.7

1. $v \xrightarrow{C\omega D}_{\mathcal{M}} w \iff v \xrightarrow{\emptyset\omega D}_{\mathcal{M}} w$ and $\mathcal{M}, v \models C$
2. $v \xrightarrow{C\omega D}_{\mathcal{M}} w \iff v \xrightarrow{C\omega\emptyset}_{\mathcal{M}} w$ and $\mathcal{M}, w \models D$

Proposition 4.10 adapts as follows.

Proposition 5.8 If $v \xrightarrow{\omega C\omega'}_{\mathcal{M}} w \iff \exists u\colon v \xrightarrow{\omega C}_{\mathcal{M}} u$ and $u \xrightarrow{C\omega'}_{\mathcal{M}} w$

The formulation of Proposition 4.11 remains unchanged. Its proof once again reduces to showing claim (1) since (2) and (3) follow from (1) and the definition of satisfaction for diamonds and boxes. The two directions of (1) are shown by induction on α using Proposition 5.7, Proposition 5.8, and the observation that for every program α and every $\omega C\omega' \in \mathcal{L}\alpha$, we have $\omega D\omega' \in \mathcal{L}\alpha$ whenever $C \subseteq D$.

As a consequence of Proposition 4.11, we obtain:

Proposition 5.9 Let $\mathcal{D}$ be a DNF of η. Then:

1. $\mathcal{M}, w \models \langle\eta\rangle\varphi \iff \exists C\theta \in \mathcal{D}\colon \mathcal{M}, w \models C$ and $\mathcal{M}, w \models \langle\theta\rangle\varphi$
2. $\mathcal{M}, w \models [\eta]\varphi \iff \forall C\theta \in \mathcal{D}\colon \mathcal{M}, w \models {\sim}C$ or $\mathcal{M}, w \models [\theta]\varphi$

where we write $\mathcal{M}, w \models {\sim}C$ as a shortcut for $\exists \psi \in C\colon \mathcal{M}, w \models {\sim}\psi$.

Proof We show (1) (the proof of (2) is analogous). For the direction from left to right, suppose $\mathcal{M}, w \models \langle\eta\rangle\varphi$. By Proposition 4.11 (2), there is some $\sigma = C_0a_1C_1 \dots a_nC_n \in \mathcal{L}\eta$ and some v such that $w \xrightarrow{\sigma}_{\mathcal{M}} v$ and $\mathcal{M}, v \models \varphi$. Since $\mathcal{D}$ is a DNF of η, we have $\eta \equiv \Sigma\mathcal{D}$, and hence $\mathcal{D}$ contains some normal guarded sequence $C\theta$ such that $\sigma \in \mathcal{L}(C\theta)$. Since θ is of the form λ or $a{:}\theta$, we must have $C_0 \in \lceil C \rceil$ and $\tau = \emptyset a_1C_1 \dots a_nC_n \in \mathcal{L}\theta$. On the other hand, by Proposition 5.7 (1), we have $\mathcal{M}, w \models C_0$ and $w \xrightarrow{\tau}_{\mathcal{M}} v$. Taken together, $w \xrightarrow{\tau}_{\mathcal{M}} v$, $\mathcal{M}, v \models \varphi$ and $\tau \in \mathcal{L}\theta$ imply $\mathcal{M}, w \models \langle\theta\rangle\varphi$ by Proposition 4.11 (2). Moreover, since $\mathcal{M}, w \models C_0$ and $C_0 \in \lceil C \rceil$ (i.e., $C \subseteq C_0$), we have $\mathcal{M}, w \models C$. The claim follows.

For the direction from right to left, suppose $\mathcal{M}, w \models C$ and $\mathcal{M}, w \models \langle\theta\rangle\varphi$ where $C\theta \in \mathcal{D}$. By Proposition 4.11 (2), there is some $\tau = C_0a_1C_1 \dots a_nC_n \in \mathcal{L}\theta$ and some v such that $w \xrightarrow{\tau}_{\mathcal{M}} v$ and $\mathcal{M}, v \models \varphi$. Since $\mathcal{M}, w \models C$, we then have $w \xrightarrow{\sigma}_{\mathcal{M}} v$ where $\sigma = (C \cup C_0)a_1C_1 \dots a_nC_n$. Clearly, $\sigma \in \lceil C \rceil \cdot \mathcal{L}\theta = \mathcal{L}(C\theta)$, and hence $\sigma \in \mathcal{L}\eta$. The claim follows by Proposition 4.11 (2). ∎

Proposition 3.2 follows by induction on the derivation of support. For Proposition 3.3, we additionally observe that, by property (2) of program DNFs, every premise of a derivation rule for support is literal or strictly smaller than the conclusion.

The definition of clausal DNFs remains the same as in §3.3 (Definition 3.8), and hence Proposition 3.10 remains valid. If we restrict the rules for $\langle\eta\rangle\varphi$ and $[\eta]\varphi$ to non-normal η (which does not change the support relation), the corresponding tableau rules can be used to compute clausal DNFs following the approach in §3.4 (termination of the tableau construction follows from property (2) of program DNFs).

5.4 Demo Sets

We adapt our setup to account for tests following [32]. To define demo sets, we extend the definition of $\xrightarrow{\alpha}_S$ to tests such that $\xrightarrow{\varphi}_S = \{(C,C) \mid C \in S, C \rhd \varphi\}$. Once again, we fix a finite, nonempty, syntactically closed formula universe $\mathcal{F}$ that now contains formulas of PDL. With this, transition completeness and demo sets are defined as before (Definition 3.4).

Unlike before, we no longer have $\xrightarrow{\alpha}_{\mathcal{M}_S} = \xrightarrow{\alpha}_S$ for every program α. To see this, consider $S = \{\{p\}\}$ and suppose $\varphi = (p \wedge q) \vee (p \wedge \neg q) \in \mathcal{F}$. Then $\{p\} \xrightarrow{\varphi}_{\mathcal{M}_S} \{p\}$ but not $\{p\} \xrightarrow{\varphi}_S \{p\}$ since $\{p\} \not\rhd \varphi$. However, we can still show the following lemma.

Lemma 5.10 Let α be a program and let $\mathcal{M}_S, C \vDash \varphi$ for all tests φ contained in α and all $C \in S$ such that $C \rhd \varphi$. Then $\xrightarrow{\alpha}_S \subseteq \xrightarrow{\alpha}_{\mathcal{M}_S}$.

Proof Suppose $C \xrightarrow{\alpha}_S D$. We show $C \xrightarrow{\alpha}_{\mathcal{M}_S} D$ by induction on α. If $\alpha = a$, the claim is immediate since, by definition, $\xrightarrow{a}_{\mathcal{M}_S} = \xrightarrow{a}_S$. If $\alpha = \varphi$, then $D = C$ and $C \rhd \varphi$. Since φ is contained in α, by assumption we have $\mathcal{M}_S, C \vDash \varphi$, and hence $C \xrightarrow{\varphi}_{\mathcal{M}_S} D$. The inductive cases are straightforward, so we show only $\alpha = \alpha_1^*$. If $\alpha = \alpha_1^*$, we have $C \xrightarrow{\alpha_1}{}^{n}_S D$ for some $n \geq 0$, and hence $C \xrightarrow{\alpha_1}{}^{n}_{\mathcal{M}_S} D$ by the inductive hypothesis. The claim follows. ∎

Lemma 4.14 now has to be formulated with respect to $\xrightarrow{\sigma}_{\mathcal{M}_S}$ rather than $\xrightarrow{\sigma}_S$. Moreover, it requires an additional assumption:

Lemma 5.11 Let S be a set of clauses, $\{C,D\} \subseteq S$, $C \rhd [\eta]\varphi$, and $C \xrightarrow{\sigma}_{\mathcal{M}_S} D$ for some $\sigma \in \mathcal{L}\eta$. Moreover, let $\mathcal{M}_S, C' \vDash \psi$ for every $C' \in S$ and every formula ψ such that $C' \rhd \psi$ and $\sim\psi$ is contained in η. Then $D \rhd \varphi$.

Proof We proceed by lexicographic induction on the length of σ and the derivation of $C \rhd [\eta]\varphi$ and do a case distinction on the shape of η. If $\eta = \lambda$ or $\eta = \alpha_1{:}\eta'$

where α_1 is an action, the argument proceeds as in the proof of Lemma 4.14. If α_1 is not an action, then there is some DNF $\mathcal{D}$ of α_1 and $C'\theta \in \mathcal{D}$ such that $\sigma \in \mathcal{L}(C'(\theta{:}\eta'))$ and C supports some formula in $\{ \sim\psi \mid \psi \in C' \}$ or $[\theta][\eta']\varphi$.

We now show that $C \rhd [\theta][\eta']\varphi$. Suppose, for contradiction, $C \rhd \sim\psi$ for some $\psi \in C'$. By property (2) of program DNFs, ψ is contained in η and hence, by our assumption, $\mathcal{M}_S, C \vDash \sim\psi$. This contradicts $C \xrightarrow{\sigma}_{\mathcal{M}_S} D$ since $\sigma \in \mathcal{L}(C'(\theta{:}\eta'))$.

Once we have $C \rhd [\theta][\eta']\varphi$ the claim follows by the inductive hypothesis for σ and $[\theta][\eta']\varphi$ since $\sigma \in \mathcal{L}(C'(\theta{:}\eta')) \subseteq \mathcal{L}(\theta{:}\eta')$. ∎

We conclude the following adaptation of Lemma 3.6:

Lemma 5.12 Let S be a set of clauses, $\{C, D\} \subseteq S$, $C \rhd [\alpha]\varphi$, and $C \xrightarrow{\alpha}_{\mathcal{M}_S} D$. Moreover, let $\mathcal{M}_S, C' \vDash \psi$ for every $C' \in S$ and every formula ψ such that $C' \rhd \psi$ and $\sim\psi$ is contained in α. Then $D \rhd \varphi$.

Proof By Proposition 4.11 (1), there is some $\sigma \in \mathcal{L}\alpha$ such that $C \xrightarrow{\sigma}_{\mathcal{M}_S} D$. The claim follows by Lemma 5.11 (which applies as programs are program sequences of length 1). ∎

The proof of Lemma 3.7 (whose formulation remains unchanged) now proceeds with Lemmas 5.10 and 5.12 by induction on a modified notion of the size of φ (notation $|\varphi|$), which is defined as usual with the exception that $|\neg p| = |p|$. Since this implies $|\sim\varphi| = |\varphi|$ for all φ, the additional assumption in Lemma 5.12 follows by the inductive hypothesis.

Finally, Lemma 4.15 requires no changes to its formulation and its proof proceeds similarly to before.

5.5 Demo Graphs

The definition of clause graphs (Definition 3.13) remains unchanged except for links, which must be adapted to the new definition of DNFs. We do so by changing properties (4) and (5) in Definition 4.16 as follows:

4. there is a clause C' such that $C'\theta$ is an element of a DNF of η and
5. D is an element of a DNF of $C_a^{\Box} \cup C'\,;\langle\theta\rangle\varphi$.

The definitions of paths, runs, loops, and demo graphs adapt without changes to their formulation. Also, Propositions 4.18 and 3.17 remain valid (the proof of Proposition 4.18 requires no changes and the proof of Proposition 3.17 adapts in a straightforward way).

Input: a normal clause $C_0 \subseteq \mathcal{F}$

Variable: $\mathcal{G} := \{C_0\}$

Invariant: $\mathcal{G}$ functional and loop-free

while $\mathcal{G}$ does not realize every $\langle a\rangle\varphi \in C \in \mathcal{G}$ **do**

1. **pick** $\langle a\rangle\langle\eta\rangle\varphi \in C \in \mathcal{G}$ such that
   - φ is not a diamond formula
   - $\mathcal{G}$ does not realize $\langle a\rangle\langle\eta\rangle\varphi$ in C
2. **pick** DNF $\mathcal{D}$ of η
3. **choose** $C'\theta \in \mathcal{D}$
4. **pick** DNF S of $C_a^{\Box} \cup C'; \langle\theta\rangle\varphi$
5. **if** $S \neq \emptyset$ **then choose** $D \in S$ **else backtrack**
6. $\mathcal{G} := \mathcal{G}; D; C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D$
7. **if** $\mathcal{G}$ contains a loop **then backtrack**

Return: – **satisfiable** if while-loop terminates regularly
– **unsatisfiable** if while-loop terminates through backtracking

Figure 6: Decision procedure for PDL

5.6 Decision Procedure

The decision procedure (Fig. 6) remains the same as in the test-free case (Fig. 5) except for steps 3 and 4 of the while-loop, which are adapted to the new definition of DNFs. The correctness of the procedure is argued as before. The definition of η-distance remains unchanged (except that σ now ranges over guarded strings and the notations $|\sigma|$ and $\xrightarrow{\sigma}_{\mathcal{M}}$ are defined as in §5.1 and §5.3, respectively). Proposition 4.20 follows analogously to before. The definitions of link and graph satisfaction, as well as the proof of Proposition 3.19 are unchanged compared to §4.6 while Lemma 4.21 adapts to reflect the changes in the decision procedure:

Lemma 5.13 Let $\mathcal{G}$ be a graph satisfied by a model $\mathcal{M}$ and let $\langle a\rangle\langle\eta\rangle\varphi \in C \in \mathcal{G}$ where φ is not a diamond formula. Then for every DNF $\mathcal{D}$ of η there is some $C'\theta \in \mathcal{D}$ such that for every DNF S of $C_a^{\Box} \cup C'; \langle\theta\rangle\varphi$ there is some $D \in S$ such that $\mathcal{M}$ satisfies $\mathcal{G}; D; C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D$.

The proof of the lemma adapts without problems.

6 Nominals

Finally, we consider the full language with nominals. Our setup allows us to incorporate nominals with only minor extensions of the decision procedure. When adding new clauses to the graph, the procedure now has to complete them by additional formulas induced by the presence of nominals. We call this technique nominal completion. The goal of nominal completion is to maintain an additional invariant on clause graphs that we call nominal coherence. Nominal coherence ensures the existence of clauses that serve as canonical representatives for nominals in the construction of a demo set from a demo graph.

Our approach differs from existing approaches to dealing with nominals [28, 7, 29, 50, 33, 11, 25] in that once a clause or a link has been added to the graph, it remains unchanged in all extensions of the graph. This simplifies the presentation and allows us to adapt our correctness arguments from the previous sections to HPDL in a straightforward way.

Once again, we proceed incrementally. We point out the changes necessary to the definitions in §5 to account for nominals and argue the correctness of the adapted system.

6.1 Nominal Coherence and Nominal Completion

The notions of demo sets and demo graphs must now account for the special semantics of nominals. We fix $\mathcal{F}$ to be a finite, nonempty, syntactically closed set of formulas of HPDL (see §2). **Demo sets** are now defined as nonempty sets of normal clauses that are transition-complete and, for every nominal $x \in \mathcal{F}$, contain at most one clause C such that $x \in C$. With this additional constraint in place, it is easily seen that every demo set describes a model of all of its clauses. Formally, the proof of this fact (Lemma 3.7) proceeds analogously to the case without nominals (§5.4) except for the following modifications:

Given a demo set S, we observe that $S' = S \cup \{\{x\} \mid x \text{ does not occur in any clause in } S\}$ is a demo set that contains exactly one clause for every nominal. Moreover, since every model of S' is a model of S, to show S satisfiable it suffices to construct a model of S'. Hence, we restrict Lemma 3.7 to demo sets that contain exactly one clause per nominal. The proof of Lemma 3.7 requires two additional base cases for nominals, which are straightforward if the demo set contains exactly one clause per nominal.

The definitions of clause graphs (Definition 3.13), functionality and realization are unchanged. For demo graphs (Definition 3.15), we introduce an additional constraint that we call nominal coherence.

Definition 6.1 A graph $\mathcal{G}$ is **nominally coherent** if for every nominal $x \in C \in \mathcal{G}$

there is a greatest clause containing x, i.e., a clause $D \in \mathcal{G}$ such that $\forall E \in \mathcal{G}\colon\ x \in E \implies E \subseteq D$.

Given a nominal x, we see the greatest clause containing x, whose existence is asserted by nominal coherence, as a canonical representative of the state corresponding to x.

Definition 6.2 The **nominal completion $C^{\mathcal{G}}$ of a clause C w.r.t. a graph $\mathcal{G}$** is defined as follows:

$$C^{\mathcal{G}} := C \cup \bigcup \{\, D \in \mathcal{G} \mid C \cap D \text{ contains a nominal} \,\}$$

If $\mathcal{G}$ is nominally coherent, nominal completion w.r.t. $\mathcal{G}$ maps every clause in $\mathcal{G}$ that contains a nominal x to the unique greatest clause for x in $\mathcal{G}$.

Proposition 6.3 Let $\mathcal{G}$ be a clause graph, $\mathcal{M}$ be a model of all the clauses in $\mathcal{G}$, and $w \in |\mathcal{M}|$. Then for every clause C we have $\mathcal{M}, w \vDash C \iff \mathcal{M}, w \vDash C^{\mathcal{G}}$.

Proof The direction from right to left is immediate. The other direction follows by the semantics of nominals. ∎

Corollary 6.4 Let $\mathcal{G}$ be a clause graph and $\mathcal{M}$ be a model of all the clauses in $\mathcal{G}$. If C is a normal clause satisfied by $\mathcal{M}$, then so is $C^{\mathcal{G}}$.

Proposition 6.5 A graph $\mathcal{G}$ is nominally coherent if and only if $C^{\mathcal{G}} \in \mathcal{G}$ for every $C \in \mathcal{G}$.

Proof For the direction from left to right, let $x \in C \in \mathcal{G}$ and let D be the greatest clause in $\mathcal{G}$ containing x, whose existence is guaranteed by nominal coherence. By the maximality of D and the definition of nominal completion, we have $D = C^{\mathcal{G}}$. The claim follows.

For the other direction, we proceed by contradiction. Suppose $\mathcal{G}$ is not nominally coherent but $C^{\mathcal{G}} \in \mathcal{G}$ holds for every $C \in \mathcal{G}$. Since $\mathcal{G}$ is not nominally coherent, there is a nominal x and two distinct clauses $C, D \in \mathcal{G}$ such that $x \in C \cap D$ but C and D are both maximal in $\mathcal{G}$. Then $C \subsetneq C \cup D \subseteq C^{\mathcal{G}}$. But since $C^{\mathcal{G}} \in \mathcal{G}$, neither C nor G can be maximal in $\mathcal{G}$. Contradiction. ∎

Proposition 6.6 If $\mathcal{G}$ is nominally coherent, then $(C^{\mathcal{G}})^{\mathcal{G}} = C^{\mathcal{G}}$ for every clause C.

Proof The inclusion $C^{\mathcal{G}} \subseteq (C^{\mathcal{G}})^{\mathcal{G}}$ is immediate. For the other inclusion, the claim follows by a straightforward case analysis. ∎

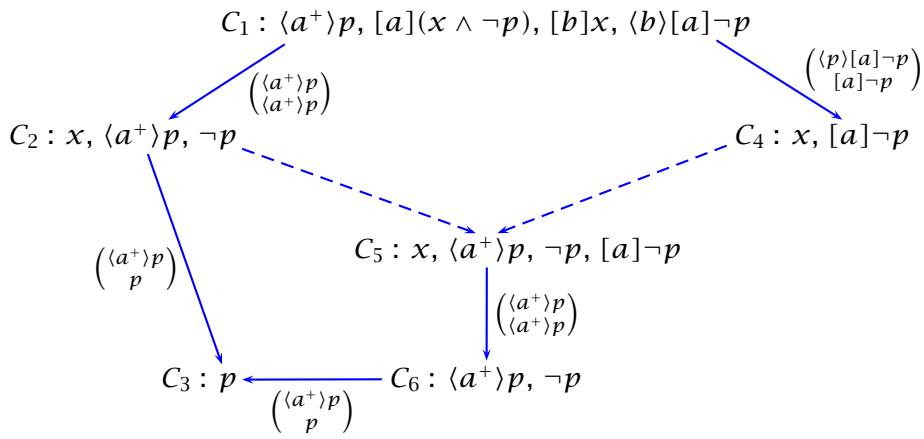


Figure 7: A demo graph

The **core** of a clause graph $\mathcal{G}$ is the set $\operatorname{Core}\mathcal{G} := \{C \in \mathcal{G} \mid C^{\mathcal{G}} = C\}$. In other words, the core consists of all clauses that cannot be extended by nominal completion. If $\mathcal{G}$ is nominally coherent, the clauses in $\operatorname{Core}\mathcal{G}$ that contain nominals are precisely the greatest clauses whose existence is guaranteed by nominal coherence. By Propositions 6.5 and 6.6, if $\mathcal{G}$ is nominally coherent, then for every clause $C \in \mathcal{G}$ we have $C^{\mathcal{G}} \in \operatorname{Core}\mathcal{G}$. So, every clause in $\mathcal{G}$ is subsumed by a larger clause in $\operatorname{Core}\mathcal{G}$, and hence every model that satisfies $\operatorname{Core}\mathcal{G}$ will satisfy all clauses in $\mathcal{G}$. In the case without nominals, we defined demo graphs so that the set of all their clauses forms a demo set. Now we define demo graphs so that their core forms a demo set. To ensure that $\operatorname{Core}\mathcal{G}$ is a demo set, we only need to consider links departing from clauses in $\operatorname{Core}\mathcal{G}$. Provided $\mathcal{G}$ is nominally coherent, every such link $C\xi D$ can now be interpreted as a transition from $C(= C^{\mathcal{G}})$ to $D^{\mathcal{G}}$.

To account for this, we define a **path for $\langle a\rangle\varphi$ in $\mathcal{G}$** as a nonempty sequence

$$(C_1\tbinom{\langle a_1\rangle\varphi_1}{\langle a_2\rangle\varphi_2}C_2)(C_2^{\mathcal{G}}\tbinom{\langle a_2\rangle\varphi_2}{\langle a_3\rangle\varphi_3}C_3)\ldots(C_{n-2}^{\mathcal{G}}\tbinom{\langle a_{n-2}\rangle\varphi_{n-2}}{\langle a_{n-1}\rangle\varphi_{n-1}}C_{n-1})(C_{n-1}^{\mathcal{G}}\tbinom{\langle a_{n-1}\rangle\varphi_{n-1}}{\psi}C_n)$$

of links in $\mathcal{G}$ such that $\langle a_1\rangle\varphi_1 = \langle a\rangle\varphi$. A path of the above form is called a **run for $\langle a\rangle\varphi$ in C** if $C_1 = C$ and ψ is not a diamond formula. The path is called a **loop for $\langle a\rangle\varphi$** if $C_1 = C_n^{\mathcal{G}}$ and $\psi = \langle a_1\rangle\varphi_1 = \langle a\rangle\varphi$. Note that while we define paths, runs and loops for arbitrary graphs, the notions become truly meaningful only for nominally coherent graphs.

For instance, consider the graph $\mathcal{G}$ in Fig. 7. The graph consists of six clauses (C_1 – C_6) and five links. Clauses C_2, C_4 and C_5 share a nominal: x. Moreover, we have $C_2^{\mathcal{G}} = C_4^{\mathcal{G}} = C_5^{\mathcal{G}} = C_5$ (we use a dashed arrow from

C_2 and C_4 to C_5 to indicate that $C_2^{\mathcal{G}} = C_4^{\mathcal{G}} = C_5$). Therefore, $\mathcal{G}$ is nominally coherent. Both diamond formulas in C_1 have runs in $\mathcal{G}$: the run for $\langle p\rangle[a]\neg p$ consists of the single link $C_1\binom{\langle b\rangle[a]\neg p}{[a]\neg p}C_4$ while the run for $\langle a^+\rangle p$ is $(C_1\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_2)(C_5\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_6)(C_6\binom{\langle a^+\rangle p}{p}C_3)$. Note that $(C_1\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_2)(C_2\binom{\langle a^+\rangle p}{p}C_3)$ is not a path since its second link departs from C_2 rather than $C_2^{\mathcal{G}} = C_5$. Finally, note that $\operatorname{Core}\mathcal{G} = \{C_1, C_3, C_5, C_6\}$.

Lemma 6.7 Let $\mathcal{G}$ be a clause graph and C, D be two clauses in $\operatorname{Core}\mathcal{G}$. If $C \cap D$ contains a nominal, then $C = D$.

Proof If $x \in C \cap D$ for some x, then $D \subseteq C^{\mathcal{G}} = C$ and $C \subseteq D^{\mathcal{G}} = D$. ∎

Definition 6.8 A clause graph $\mathcal{G}$ is a **demo graph** if $\mathcal{G}$ is finite, functional, nominally coherent, realizes every diamond formula in $\operatorname{Core}\mathcal{G}$, and contains no loops.

Note that the graph $\mathcal{G}$ in Fig. 7 is a demo graph. So is the graph $\mathcal{G}' = \mathcal{G} \setminus \{C_2\binom{\langle a^+\rangle p}{p}C_3\}$: since $C_2 \notin \operatorname{Core}\mathcal{G} = \operatorname{Core}\mathcal{G}'$, $\mathcal{G}'$ does not need to realize $\langle a^+\rangle p$ in C_2.

Proposition 4.18 is adapted as follows.

Proposition 6.9 Let $\mathcal{G}$ be a demo graph and $\langle a\rangle\varphi \in C$ for some clause $C \in \operatorname{Core}\mathcal{G}$. Then $\mathcal{G}$ contains a unique run for $\langle a\rangle\varphi$ in C.

Proof The existence of runs follows from the absence of loops and the finiteness of $\mathcal{G}$ if one can show that every path in $\mathcal{G}$ that is not a run can be extended by a link. So, let $C\binom{\langle a\rangle\varphi}{\langle b\rangle\psi}D$ be the last link of a path in $\mathcal{G}$. Since $\mathcal{G}$ realizes every diamond formula in its core and $\langle b\rangle\psi \in D$, it suffices to show $D^{\mathcal{G}} \in \operatorname{Core}\mathcal{G}$, which follows with Propositions 6.5 and 6.6. The uniqueness of runs follows from the functionality of $\mathcal{G}$. ∎

Proposition 3.17 decomposes into the following two propositions. Taken together, the propositions ensure that every demo graph $\mathcal{G}$ gives rise to a model that satisfies all of the clauses in $\mathcal{G}$.

Proposition 6.10 Let $\mathcal{G}$ be a nominally coherent graph. Then for every clause $C \in \mathcal{G}$ there is some $D \in \operatorname{Core}\mathcal{G}$ such that $C \subseteq D$.

Proof The claim follows with Propositions 6.5 and 6.6. ∎

Proposition 6.11 The core of a demo graph is a demo set.

Proof Transition completeness follows with Proposition 6.9 by the same reasoning as in the test-free case (see §5.5). The uniqueness of clauses for every nominal follows with Lemma 6.7. ∎

Input: a normal clause $C_0 \subseteq \mathcal{F}$

Variable: $\mathcal{G} := \{C_0\}$

Invariant: $\mathcal{G}$ functional, loop-free, and nominally coherent

while $\mathcal{G}$ does not realize every $\langle a\rangle\varphi \in C \in \mathrm{Core}\,\mathcal{G}$ **do**

1. **pick** $\langle a\rangle\langle\eta\rangle\varphi \in C \in \mathrm{Core}\,\mathcal{G}$ such that
   - φ is not a diamond formula
   - $\mathcal{G}$ does not realize $\langle a\rangle\langle\eta\rangle\varphi$ in C
2. **pick** DNF $\mathcal{D}$ of η
3. **choose** $C'\theta \in \mathcal{D}$
4. **pick** DNF S of $C_a^{\Box} \cup C'\,;\langle\theta\rangle\varphi$
5. **if** $S \neq \emptyset$ **then choose** $D \in S$ **else backtrack**
6. **if** $D^{\mathcal{G}}$ not normal **then backtrack**
7. $\mathcal{G} := \mathcal{G}\,;D\,;C\binom{\langle a\rangle\langle\eta\rangle\varphi}{\langle\theta\rangle\varphi}D\,;D^{\mathcal{G}}$
8. **if** $\mathcal{G}$ contains a loop **then backtrack**

Return: – **satisfiable** if while-loop terminates regularly
– **unsatisfiable** if while-loop terminates through backtracking

Figure 8: Decision procedure for HPDL

6.2 Decision Procedure

The decision procedure for HPDL is defined in Fig. 8. The changes compared to the procedure for PDL are limited to step 6 (which is new) and step 7. Moreover, the graph $\mathcal{G}$ satisfies nominal coherence as an additional invariant. Whenever the procedure extends $\mathcal{G}$ by a link $C\xi D$, it ensures that nominal coherence is maintained by adding the clause $D^{\mathcal{G}}$ (step 7). Since $\mathcal{G}$ contains only normal clauses, prior to adding $D^{\mathcal{G}}$, the procedure checks if $D^{\mathcal{G}}$ is normal (step 6). Note that since $D^{\mathcal{G}}$ is a union of normal clauses, it is normal if and only if it does not contain a complementary pair of literals. So, the conditional in step 6 effectively checks for the absence of pairs $p, \neg p$. The preservation of nominal coherence is then ensured by the following lemma:

Lemma 6.12 Let $\mathcal{G}$ be a nominally coherent graph and let C be a normal clause such that $C^{\mathcal{G}}$ is normal. Then $\mathcal{G}\,;C\,;C^{\mathcal{G}}$ is nominally coherent.

Proof Let $\mathcal{G}' = \mathcal{G}\,;C\,;C^{\mathcal{G}}$. By Proposition 6.5, it suffices to show $(C^{\mathcal{G}})^{\mathcal{G}'} \in \mathcal{G}'$. This

follows from the nominal coherence of $\mathcal{G}$ since $(C^{\mathcal{G}})^{\mathcal{G}'} = (C^{\mathcal{G}})^{\mathcal{G}} = C^{\mathcal{G}}$ (Proposition 6.6). ∎

The graph $\mathcal{G}$ in Fig. 7 results from a possible run of the decision procedure on C_1. In the first iteration of the while-loop, the procedure picks the formula $\langle a^+\rangle p \in C_1$ and the DNF $\mathcal{D} = \{\lambda, a{:}a^*\}$ of a^*, then chooses $\emptyset(a{:}a^*) \in \mathcal{D}$, picks the DNF $\{C_2\}$ of $C_1{}^{\Box}_{a} \cup \emptyset; \langle a^+\rangle p = \{x \wedge \neg p, \langle a^+\rangle p\}$, and chooses $C_2 \in \{C_2\}$. Since $C_2^{\mathcal{G}} = C_2$ is normal, the procedure extends $\mathcal{G}$ by C_2 and $C_1\binom{\langle a^+\rangle p}{\langle a^+\rangle p}C_2$. In the second iteration of the loop, the procedure realizes $\langle a^+\rangle p \in C_2$ by extending $\mathcal{G}$ by C_3 and the link from C_2 to C_3. Thereafter, the procedure realizes $\langle p\rangle[a]\neg p \in C_1$ by extending $\mathcal{G}$ by C_4, the link from C_1 to C_4, and $C_5 = C_4^{\mathcal{G}} = C_2 \cup C_4$. The procedure concludes by realizing $\langle a^+\rangle p \in C_5$ (introducing C_6 and the link from C_5 to C_6) and, finally, $\langle a^+\rangle p \in C_6$ (introducing the link from C_6 to C_3). Since $\mathcal{G}$ realizes every diamond formula in $\mathrm{Core}\,\mathcal{G}$, the procedure terminates returning "satisfiable".

The correctness of the procedure is argued similarly to before. Termination is guaranteed since the size of $\mathcal{G}$ is bounded in the cardinality of $\mathcal{F}$. The invariants together with the negated loop condition imply that $\mathcal{G}$ is a demo graph, and is hence satisfiable by Proposition 6.11 and Lemma 3.7.

It remains to argue that the procedure succeeds for every satisfiable clause. For this we follow our chain of reasoning from before. The definitions of $\delta^{\eta}_{\mathcal{M}}$ and link satisfaction remain unchanged. Proposition 6.3 yields the following corollary.

Corollary 6.13 If $\mathcal{M}$ satisfies all clauses of $\mathcal{G}$, then for every clause C, formula φ and program sequence η we have $\delta^{\eta}_{\mathcal{M}} C\varphi = \delta^{\eta}_{\mathcal{M}} C^{\mathcal{G}}\varphi$.

With this corollary, the proof of Proposition 3.19 adapts in a straightforward way. The formulation and proof of Lemma 5.13 remain unchanged. In addition, we now have to justify that for every clause D chosen according to Lemma 5.13, $D^{\mathcal{G}}$ is normal and satisfied by $\mathcal{M}$. Both claims follow by Corollary 6.4.

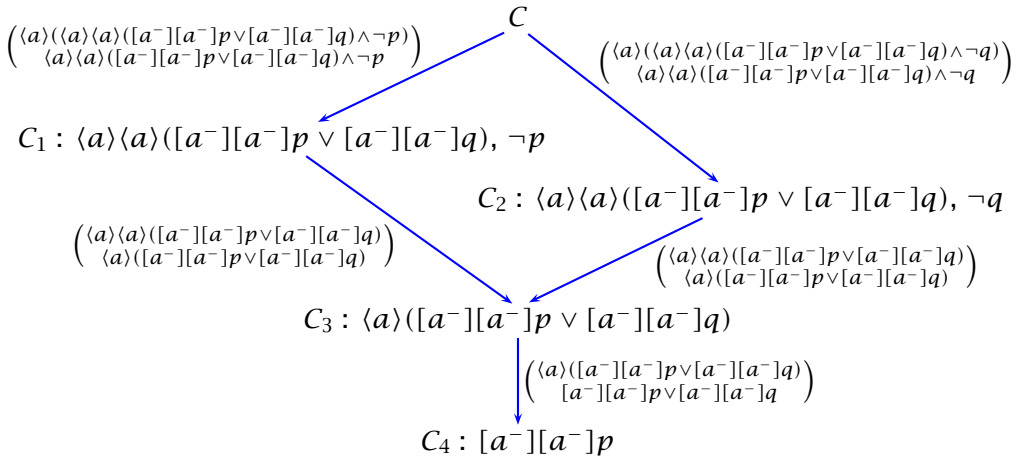
7 Beyond HPDL

An interesting avenue for future research is extending the present approach to more expressive logics. A possible first step is looking at extensions of (H)PDL that are obtained by enriching the language of programs by additional constructs like converse, program intersection, or (atomic) program complementation. Unfortunately, extending our framework to any of these logics is a non-trivial task. In the case of PDL with program intersection (IPDL), the difficulty is already suggested by the fact that IPDL is 2EXPTIME-complete [41]. In its present

form, our framework is geared towards NEXPTIME procedures. To obtain procedures of higher complexity, one would either have to significantly change the demo search strategy or to modify the notion of a clause so as to allow for a larger state space. For a more concrete problem with IPDL, consider the clause $C = \{\langle a\rangle p, \langle b\rangle p, [a \cap b]\neg p\}$ (note that there is no obvious way to decompose $[a \cap b]\neg p$ into a Boolean combination of literals). Since $C^{\Box}_a = C^{\Box}_b$ (regardless of how we extend the notion of request to program intersection), we will only produce one successor of C for both $\langle a\rangle p$ and $\langle b\rangle p$. A model of C, however, needs distinct successors for the two diamond formulas. Looking from another perspective, we could say that our approach relies on stability of satisfaction under filtration in the style of [42, 19] (see also [6]), which does not hold in the presence of program intersection.

The same applies to program complementation. Consider, for instance, the clause $C = \{\langle a\rangle p, \langle \bar{a}\rangle p\}$, where $\bar{a}$ denotes the complement of a. Then $\mathcal{M}$ where $|\mathcal{M}| = \{u, v, w\}$, $\xrightarrow{a}_{\mathcal{M}} = \{(u, v)\}$, $\xrightarrow{b}_{\mathcal{M}} = \{(u, w)\}$ and $\mathcal{M}p = \{v, w\}$ is a model of C (assuming v and w are distinct) but the filtration of $\mathcal{M}$ (with respect to $\{p\}$) is not a model of C.

Unlike program intersection and complementation, the extension of PDL by converse preserves stability of satisfaction under filtration (see, e.g., [32]). Still, converse poses a problem for our nondeterministic demo search strategy. Consider the clause $C = \{\langle a\rangle(\langle a\rangle\langle a\rangle([a^-][a^-]p \vee [a^-][a^-]q) \wedge \neg p), \langle a\rangle(\langle a\rangle\langle a\rangle([a^-][a^-]p \vee [a^-][a^-]q) \wedge \neg q)\}$ where a^- denotes the converse of a. While C is satisfiable, it is not obvious how to find a demo for C in our framework. Expansion of C yields the graph



where the only "don't know" choice occurs in the computation of C_4, in which the disjunct $[a^-][a^-]p$ is chosen over $[a^-][a^-]q$. Since C_4 will be an a-successor of C_3 in the clause model induced by the graph, C_3 needs to be extended by $[a^-]p$

(either by replacing C_3 by a larger clause or adding an extended copy of C_3). Then, however, both C_1 and C_2 need to be extended by p, which will render C_1 unsatisfiable. Symmetrically, choosing $C_4 = \{[a^-][a^-]p\}$ will lead to a conflict in C_2. To solve the problem, it seems necessary to have a mechanism which allows to introduce two separate copies of C_3, one for C_1 and one for C_2. This way, one can select $\{[a^-][a^-]q\}$ as a descendant of C_1 and $\{[a^-][a^-]p\}$ as a descendant of C_2 without violating functionality. Possibly, this can be achieved by adapting some of the ideas in [21, 54] to the nondeterministic setting.

A possibly more promising direction is extending our approach to temporal logics. We expect that the present framework can be adapted to both linear temporal logic [45, 43] and computation tree logic [12, 14] using ideas from [9]. In particular, such an extension would require devising a mechanism for tracking the fulfillment of several, possibly conflicting eventualities at the same time. Also, the correspondence between a demo graph and a model would become less immediate. Instead of building the model directly from the clauses of the graph, one would have to rely on a model construction in the style of [15, 14].

To deal with nominals in the presence of universal eventualities as they occur in CTL (i.e., formulas of the form $AF\varphi$ or $A(\varphi U\psi)$), more serious modifications of the approach seem to be necessary. In particular, one can no longer rely on model constructions in [15, 14] since they can introduce multiple copies of a state, which is not generally compatible with the semantics of nominals.

Possible extensions to even more expressive temporal logics like (hybrid) CTL* [16] or μ-calculus [38] seem far more challenging and require further investigation. Inspiration for this can be drawn from both automata-theoretic approaches [17, 8] and existing goal-directed calculi [48, 30] for nominal-free versions of the logics.

8 Concluding Remarks

We presented a modular, extensible approach to deciding modal logics with eventualities ranging from K* to HPDL. Our approach for the first time yields goal-directed decision procedures for logics combining nominals and eventualities. In particular, we for the first time obtain a goal-directed procedure for HPDL. Our approach is designed in such a way that the machinery needed to deal with nominals is completely decoupled from the treatment of eventualities.

Similarly to existing approaches to hybrid logics [28, 7, 29, 50, 33, 11, 25], we base our procedure on nondeterministic search. Unlike existing approaches to logics with nominals, all of which employ prefixes, our approach is prefix-free. Moreover, it relies on a particularly simple "blocking strategy" for termination: before adding a clause C to a graph G, an implementation of our procedure

would have to check if C is already contained in $\mathcal{G}$ so as to make sure that $\mathcal{G}$ remains a set. Unlike with existing approaches, which explicitly restrict their respective calculi to achieve termination, the set containment check is implicit in the formulation of our procedure.

To account for the semantics of nominals, we introduce an additional representation invariant on clause graphs that we call nominal coherence, as well as a technique called nominal completion that we use to maintain the invariant. Also new to the present approach is the notion of support, which generalizes the ideas behind Hintikka sets as used, e.g., in [46, 32].

New and crucial to our approach is the separation of demo search into a nondeterministic construction of demo graphs from normal clauses and a DNF computation that reduces arbitrary clauses to normal clauses. This separation is essential for the completeness of our search strategy, which differs from the strategies used by pruning-based procedures in that we restrict the search to demos containing only one outgoing edge per diamond formula and clause. We call this property functionality. Functionality greatly simplifies checking the fulfillment of eventualities, reducing it to simple reachability checking. In a prefixed setting, this kind of reachability checking was first employed by Baader [4]. In the automata-theoretic setting, ideas similar to functionality were exploited by Emerson and Sistla [18] to reduce the blowup associated with determinization (see also [17]).

Initially conceived for the basic hybrid logic with eventualities [34], our approach scales to PDL and HPDL. The key adaptation necessary to treat regular programs as they occur in PDL concerns the definitions of support and DNFs. The treatment of nominals, on the other hand, is completely unaffected by the transition from K* to PDL.

While we provide straightforward algorithms for computing clausal DNFs and program DNFs, the overall procedure does not depend on the details of these algorithms. This makes it possible to implement the DNF computation by different, possibly more efficient algorithms than the ones provided in the paper without having to reprove the correctness of the overall procedure.

For all logics considered in the paper, the size of the clause graph constructed by our procedures is exponentially bounded in the input, which yields a NEXPTIME bound on the worst-case complexity of the procedures. This is suboptimal since all of the logics have an EXPTIME-complete decision problem [6, 49, 32]. In fact, in [32] a worst-case optimal procedure for HPDL with converse and difference modalities is presented that directly extends Pratt's procedure in [46]. However, the procedure in [32] is not goal-directed (and hence not practical) since its treatment of nominals relies on brute-force enumeration of nominally coherent subsets (of which there are exponentially many in the size of the input). The

procedures in the present paper, on the other hand, have the potential to display acceptable performance on practical problems. Indeed, in the absence of eventualities the procedures reduce to variants of state-of-the-art tableau-based decision procedures for hybrid logic such as those described in [28, 7, 33]. While generally not worst-case optimal, such procedures are easy to implement and have considerable potential for optimization [27, 53]. They provide a basis for a number of systems (e.g., [52, 51, 26, 22]), some of which have been successfully applied in practical settings. While recent extensions of the pruning-based approach by Goré and Widmann [20, 21] yield efficient and worst-case optimal decision procedures, it is presently not known how to extend their approach to nominals so that worst-case optimality is retained [21, 31]. Thus, devising efficient and worst-case optimal decision procedures for modal logics with nominals and eventualities remains an open problem.

References

[1] Pietro Abate, Rajeev Goré, and Florian Widmann. An on-the-fly tableau-based decision procedure for PDL-satisfiability. In Carlos Areces and Stéphane Demri, editors, *M4M-5*, volume 231 of *Electron. Notes Theor. Comput. Sci.*, pages 191–209. Elsevier, 2009.

[2] Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.

[3] Carlos Areces and Balder ten Cate. Hybrid logics. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 821–868. Elsevier, 2007.

[4] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, DFKI, 1990.

[5] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The temporal logic of branching time. *Acta Inform.*, 20(3):207–226, 1983.

[6] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

[7] Thomas Bolander and Patrick Blackburn. Termination for hybrid tableaus. *J. Log. Comput.*, 17(3):517–554, 2007.

[8] Piero A. Bonatti, Carsten Lutz, Aniello Murano, and Moshe Y. Vardi. The complexity of enriched μ-calculi. In Michele Bugliesi, Bart Preneel, Vladimiro

Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 540–551. Springer, 2006.

[9] Kai Brünnler and Martin Lange. Cut-free systems for temporal logic. *J. Log. Algebr. Program.*, 76(2):216–225, 2008.

[10] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[11] Serenella Cerrito and Marta Cialdea Mayer. An efficient approach to nominal equalities in hybrid logic tableaux. *J. Appl. Non-Class. Log.*, 20(1-2):39–61, 2010.

[12] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

[13] Giuseppe De Giacomo and Fabio Massacci. Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Inf. Comput.*, 162(1–2):117–137, 2000.

[14] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 995–1072. Elsevier, 1990.

[15] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[16] E. Allen Emerson and Joseph Y. Halpern. “Sometimes” and “not never” revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[17] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[18] E. Allen Emerson and A. Prasad Sistla. Deciding full branching time logic. *Inf. Control*, 61(3):175–201, 1984.

[19] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[20] Rajeev Goré and Florian Widmann. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In Renate A. Schmidt, editor, *CADE-22*, volume 5663 of *LNCS*, pages 437–452. Springer, 2009.

[21] Rajeev Goré and Florian Widmann. Optimal tableaux for propositional dynamic logic with converse. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 225–239. Springer, 2010.

[22] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus: A tableau prover for hybrid logic. In Thomas Bolander and Torben Braüner, editors, *M4M-6*, volume 262 of *Electron. Notes Theor. Comput. Sci.*, pages 127–139. Elsevier, 2010.

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

[24] K. Jaakko J. Hintikka. Form and content in quantification theory. Two papers on symbolic logic. *Acta Philos. Fenn.*, 8:7–55, 1955.

[25] Guillaume Hoffmann. Lightweight hybrid tableaux. *J. Appl. Log.*, 8(4):397–408, 2010.

[26] Guillaume Hoffmann and Carlos Areces. HTab: A terminating tableaux system for hybrid logic. In Carlos Areces and Stéphane Demri, editors, *M4M-5*, volume 231 of *Electron. Notes Theor. Comput. Sci.*, pages 3–19. Elsevier, 2009.

[27] Ian Horrocks. Implementation and optimization techniques. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, pages 329–373. Cambridge University Press, 2nd edition, 2007.

[28] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the SHOQ(D) description logic. In Bernhard Nebel, editor, *IJCAI 2001*, pages 199–204. Morgan Kaufmann, 2001.

[29] Ian Horrocks and Ulrike Sattler. A tableau decision procedure for SHOIQ. *J. Autom. Reason.*, 39(3):249–276, 2007.

[30] Natthapong Jungteerapanich. A tableau system for the modal μ-calculus. In Martin Giese and Arild Waaler, editors, *TABLEAUX 2009*, volume 5607 of *LNCS*, pages 220–234. Springer, 2009.

[31] Mark Kaminski. *Incremental Decision Procedures for Modal Logics with Nominals and Eventualities*. PhD thesis, Saarland University, 2012.

[32] Mark Kaminski, Thomas Schneider, and Gert Smolka. Correctness and worst-case optimality of Pratt-style decision procedures for modal and hybrid logics. In Kai Brünnler and George Metcalfe, editors, *TABLEAUX 2011*, volume 6793 of *LNCS*, pages 196–210. Springer, 2011.

[33] Mark Kaminski and Gert Smolka. Terminating tableau systems for hybrid logic with difference and converse. *J. Log. Lang. Inf.*, 18(4):437–464, 2009.

[34] Mark Kaminski and Gert Smolka. Terminating tableaux for hybrid logic with eventualities. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 240–254. Springer, 2010.

[35] Mark Kaminski and Gert Smolka. Clausal tableaux for hybrid PDL. In Hans van Ditmarsch, David Fernández Duque, Valentin Goranko, Wojtek Jamroga, and Manuel Ojeda-Aciego, editors, *M4M-7*, volume 278 of *Electron. Notes Theor. Comput. Sci.*, pages 99–113. Elsevier, 2011.

[36] Donald M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3(4):361–386, 1969.

[37] Robert Kowalski. *Logic for Problem Solving*. North-Holland, 1979.

[38] Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[39] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In Dirk van Dalen and Marc Bezem, editors, *CSL'96*, volume 1258 of *LNCS*, pages 244–259. Springer, 1996.

[40] Saul A. Kripke. Semantical analysis of modal logic I: Normal modal propositional calculi. *Z. Math. Log. Grundl. Math.*, 9:67–96, 1963.

[41] Martin Lange and Carsten Lutz. 2-ExpTime lower bounds for propositional dynamic logic with intersection. *J. Symb. Log.*, 70(4):1072–1086, 2005.

[42] Edward J. Lemmon and Dana Scott. *The 'Lemmon Notes': An Introduction to Modal Logic*. Blackwell, 1977.

[43] Orna Lichtenstein and Amir Pnueli. Propositional temporal logics: Decidability and completeness. *L. J. IGPL*, 8(1):55–85, 2000.

[44] Linh Anh Nguyen and Andrzej Szałas. Checking consistency of an ABox w.r.t. global assumptions in PDL. *Fundam. Inform.*, 102(1):97–113, 2010.

[45] Amir Pnueli. The temporal logic of programs. In *FOCS '77*, pages 46–57. IEEE Computer Society Press, 1977.

[46] Vaughan R. Pratt. Models of program logics. In *Proc. 20th Annual Symp. on Foundations of Computer Science (FOCS'79)*, pages 115–122. IEEE Computer Society Press, 1979.

[47] Vaughan R. Pratt. A near-optimal method for reasoning about action. *J. Comput. Syst. Sci.*, 20(2):231–254, 1980.

[48] Mark Reynolds. A faster tableau for CTL*. In Gabriele Puppis and Tiziano Villa, editors, *GandALF 2013*, volume 119 of *Electron. Proc. Theor. Comput. Sci.*, pages 50–63, 2013.

[49] Ulrike Sattler and Moshe Y. Vardi. The hybrid μ-calculus. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR 2001*, volume 2083 of *LNCS*, pages 76–91. Springer, 2001.

[50] Renate A. Schmidt and Dmitry Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 194–209. Springer, 2008.

[51] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Semant.*, 5(2):51–53, 2007.

[52] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR 2006*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.

[53] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *J. Autom. Reason.*, 39(3):277–316, 2007.

[54] Florian Widmann. *Tableaux-based Decision Procedures for Fixed Point Logics*. PhD thesis, Australian National University, 2010.

[55] Pierre Wolper. Temporal logic can be more expressive. *Inf. Control*, 56(1-2):72–99, 1983.