Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 278 (2011) 99–113

www.elsevier.com/locate/entcs

Clausal Tableaux for Hybrid PDL

Mark Kaminski Gert Smolka

Saarland University
Saarbrücken, Germany

Abstract

We present the first tableau-based decision procedure for PDL with nominals. The procedure is based on a prefix-free clausal tableau system designed as a basis for gracefully degrading reasoners. The clausal system factorizes reasoning into regular, propositional, and modal reasoning. This yields a modular decision procedure and pays off in transparent correctness proofs.

Keywords: hybrid logic, dynamic logic, tableau systems, decision procedures

1 Introduction

PDL (propositional dynamic logic) [6,13,9] is an expressive modal logic invented for reasoning about programs. It extends basic modal logic with expressions called programs. Programs describe relations from states to states and are used to express modalities. Programs are composed with the operators familiar from regular expressions. In addition, they may employ formulas so that conditionals and while loops can be expressed. Fischer and Ladner [6] show the decidability of PDL using a filtration argument. They also prove that the satisfiability problem for PDL is EXPTIME-hard. Pratt [17] shows that PDL satisfiability is in EXPTIME using a tableau method with an and-or graph representation. Goré and Widmann [7,8] address the efficient implementation of Pratt-style decision procedures.

We consider PDL extended with nominals [14,15], a logic we call hybrid PDL or HPDL. Nominals are atomic formulas that hold exactly for one state. Nominals equip PDL with equality and are the characteristic feature of hybrid logic [2]. The satisfiability problem of HPDL is in EXPTIME [15,18].

We are interested in a tableau system for HPDL that can serve as a basis for gracefully degrading decision procedures. We found it impossible to extend one of the existing tableau methods for PDL [17,5,1,7] to nominals. The difficulties are in the correctness proofs. For Pratt-like methods [17,7], the problem stems from the fact that the global and-or graph representation is not compatible with nominal

1571-0661
doi:10.1016/j.entcs.2011.10.009

propagation (see Remark 5.6 in [10] for a discussion and an example; the problem is also noted in [19]).

The difficulties led us to the development of a new tableau method for modal logic. The new method is based on a prefix-free clausal form. In a previous paper [10] we used the method to give a tableau-based decision procedure for the sublogic of HPDL that restricts programs to the forms a and a^* where a is a primitive action. In the present paper we extend the clausal method to full HPDL and obtain the first tableau-based decision procedure for HPDL.

Our method factorizes reasoning into regular reasoning, propositional reasoning and modal reasoning. At each level we realize reasoning with tableau methods. Nominals are handled at the modal level. Given our approach, the integration of nominals is straightforward. The modular structure of our decision procedure pays off in transparent correctness proofs. Each level invites optimizations. The regular level, in particular, asks for further investigation. It may profit from efficient methods for translating regular expressions into deterministic automata.

In contrast to previous approaches, we do not rely on the Fischer-Ladner closure. Instead, we use the notion of a finitary regular DNF that can be obtained at the regular level.

Following Baader [3] and De Giacomo and Massacci [5], we disallow bad loops and thus avoid the a posteriori eventuality checking of Pratt's method [17].

The paper is organized as follows. First we define HPDL and outline the clausal tableau method with examples. Then we address, one after the other, regular, propositional, and modal reasoning. Finally, we prove the correctness of the decision procedure.

2 Hybrid PDL

We define the syntax and semantics of HPDL. We assume that three kinds of *names* are given:

- *nominals* (metavariables x, y, z; denote states)
- *predicates* (metavariables p, q, r; denote sets of states)
- *actions* (metavariables a, b, c; denote relations from states to states).

The interpretations of HPDL are the usual transition systems where states are labelled with predicates and edges are labelled with actions. Formally, an *interpretation* $\mathcal{I}$ is a tuple consisting of the following components:

- A nonempty set $|\mathcal{I}|$ of *states*.
- A state $\mathcal{I}x \in |\mathcal{I}|$ for every nominal x.
- A set $\mathcal{I}p \subseteq |\mathcal{I}|$ for every predicate p.
- A relation $\xrightarrow{a}_{\mathcal{I}} \subseteq |\mathcal{I}| \times |\mathcal{I}|$ for every action a.

Formulas (metavariables s, t, u) and *programs* (α, β, γ) are defined as follows:

$$s \ ::= \ x \mid p \mid \neg s \mid s \wedge s \mid \langle \alpha \rangle s$$

$$\alpha \ ::= \ a \mid s \mid 1 \mid \alpha + \alpha \mid \alpha\alpha \mid \alpha^*$$

The grammar is to be read inclusive, that is, every nominal and every predicate is a formula, and every action and every formula is a program. Given an interpretation, formulas denote sets of states and programs denote relations from states to states. We use the letters X, Y, Z to denote states. The semantic relations $\mathcal{I}, X \models s$ and $X \xrightarrow{\alpha}_{\mathcal{I}} Y$ are defined by mutual induction on the structure of formulas and programs:

$$\begin{aligned}
\mathcal{I}, X \models x &\iff X = \mathcal{I}x \\
\mathcal{I}, X \models p &\iff X \in \mathcal{I}p \\
\mathcal{I}, X \models \neg s &\iff \text{not } \mathcal{I}, X \models s \\
\mathcal{I}, X \models s \wedge t &\iff \mathcal{I}, X \models s \text{ and } \mathcal{I}, X \models t \\
\mathcal{I}, X \models \langle\alpha\rangle s &\iff \exists Y\colon\ X \xrightarrow{\alpha}_{\mathcal{I}} Y \text{ and } \mathcal{I}, Y \models s
\end{aligned}$$

$$\begin{aligned}
X \xrightarrow{a}_{\mathcal{I}} Y &\iff X \xrightarrow{a}_{\mathcal{I}} Y \\
X \xrightarrow{s}_{\mathcal{I}} Y &\iff X = Y \text{ and } \mathcal{I}, X \models s \\
X \xrightarrow{1}_{\mathcal{I}} Y &\iff X = Y \\
X \xrightarrow{\alpha+\beta}_{\mathcal{I}} Y &\iff X \xrightarrow{\alpha}_{\mathcal{I}} Y \text{ or } X \xrightarrow{\beta}_{\mathcal{I}} Y \\
X \xrightarrow{\alpha\beta}_{\mathcal{I}} Y &\iff \exists Z\colon\ X \xrightarrow{\alpha}_{\mathcal{I}} Z \text{ and } Z \xrightarrow{\beta}_{\mathcal{I}} Y \\
X \xrightarrow{\alpha^*}_{\mathcal{I}} Y &\iff X \xrightarrow{\alpha}{}^*_{\mathcal{I}} Y
\end{aligned}$$

$\xrightarrow{\alpha}{}^*_{\mathcal{I}}$ denotes the reflexive transitive closure of $\xrightarrow{\alpha}_{\mathcal{I}}$

Given a set A of formulas, we write $\mathcal{I}, X \models A$ if $\mathcal{I}, X \models s$ for all formulas $s \in A$. An interpretation $\mathcal{I}$ *satisfies* (or is a *model* of) a formula s or a set A of formulas if there is a state $X \in |\mathcal{I}|$ such that $\mathcal{I}, X \models s$ or, respectively, $\mathcal{I}, X \models A$. A formula s (a set A) is *satisfiable* if s (A) has a model.

The *complement* $\sim s$ of a formula s is t if $s = \neg t$ and $\neg s$ otherwise. Note that $\sim\sim s = s$ if s is not a double negation. We use the notations $s \vee t := \neg(\sim s \wedge \sim t)$ and $[\alpha]s := \neg\langle\alpha\rangle\sim s$. Note that $\sim\langle\alpha\rangle p = [\alpha]\neg p$ and $\sim\langle\alpha\rangle\neg p = [\alpha]p$. The *size* $|s|$ and $|\alpha|$ of formulas and programs is defined as the size of the abstract syntax tree. For instance, $|ap| = |\langle p\rangle q| = 3$. Note that $|s \vee t| > |s|, |t|$ and $|[\alpha]s| \geq |\langle\alpha\rangle s| > |\alpha|, |s|$.

3 Outline of the Method

Our tableau method is based on a clausal form, which provides for the separation of regular, propositional, and modal reasoning. We start with a few definitions and three examples.

A *basic formula* is a formula of the form p, x, or $\langle a\alpha\rangle s$. A *literal* is a basic formula or the complement of a basic formula. A *clause* (written C, D, E) is a finite set of literals that contains no complementary pair (i.e., a pair of the form p, $\neg p$). We interpret clauses conjunctively. *Satisfaction of clauses* (i.e., $\mathcal{I}, X \models C$) is

a special case of satisfaction of sets of formulas (i.e., $\mathcal{I}, X \models A$), which was defined in §2. For instance, the clause $\{\langle a\rangle p, [a](\neg p \wedge q)\}$ is unsatisfiable.

A *claim* is a pair C^s consisting of a clause C and a diamond formula s. While we do not require $s \in C$, for every claim C^s that we consider in the following it will be the case that C supports s. The notion of support is defined formally later such that C supporting s (or A) and $\mathcal{I}, X \models C$ implies $\mathcal{I}, X \models s$ ($\mathcal{I}, X \models A$). The *request* of a clause C for an action a is $\mathcal{R}_a C := \{\, [\alpha]s \mid [a\alpha]s \in C \,\}$. As an example, consider the clause $C = \{\langle ab^*\rangle p,\ \langle bb^*\rangle p,\ [a(a+b)^*]\neg p\}$. We have $\mathcal{R}_a C = \{[(a+b)^*]\neg p\}$ and $\mathcal{R}_b C = \emptyset$.

Our tableau method works on clauses and links (to be formally defined later) rather than single formulas. Intuitively, a link is a pair of claims $C^s D^t$ denoting that in order for a model to satisfy the diamond literal s in C, it suffices to satisfy $D \cup \{t\}$. Given a single clause, the method tries to extend it to a tableau branch in which every diamond literal s in every clause C is realized with a link $C^s D^t$ where D is one of the clauses of the branch. Provided the relation induced by the links is terminating, every such branch is a model of all of its clauses. If every branch constructed by the method fails to realize some diamond literal with a link or contains a loop formed by links, we conclude that the input is unsatisfiable. Thus we obtain a decision procedure for the satisfiability of clauses. At the same time, the procedure decides the satisfiability of formulas since in HPDL a formula s is satisfiable if and only if so is the clause $\{\langle a\rangle s\}$ (the choice of a does not matter).

The method is implemented with three reasoners. The regular reasoner decomposes a program α into simpler (according to some measure) programs $\beta_1, \ldots, \beta_n$ such that $\alpha \equiv \beta_1 + \cdots + \beta_n$. For instance, a^* is decomposed into aa^* and 1.

The propositional reasoner determines for every set A of formulas a set of clauses supporting A such that $\mathcal{I}, X \models A$ if and only if $\mathcal{I}, X \models C$ for one of the clauses. Given the formula $\langle a^*\rangle p \wedge [b^*]\neg p$, for instance, the propositional reasoner determines the single clause $\{\langle aa^*\rangle p,\ \neg p,\ [bb^*]\neg p\}$.

The modal reasoner is the top-level reasoner of our tableau method. For every satisfiable clause it constructs a finite model whose states are clauses and where every state C satisfies the clause C. To do so, the modal reasoner starts with the initial clause and derives further clauses until every diamond literal s in every clause C is realized with a link. The modal reasoner calls the regular reasoner to determine the successor formula t and the propositional reasoner to determine the successor clause D. The tableau method terminates since the derived clauses must take their literals from a finite set that can be determined from the initial formulas.

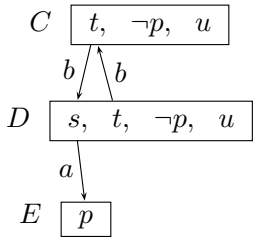
Let us now give three examples illustrating our method in action. At this point, they are there to provide some additional intuition about the method and do not have to be understood in all details. To fully understand the examples one should review them after all of the formal prerequisites have been introduced in § 8.

Example 3.1 Consider the following literals and clauses:

$$\begin{array}{lcllcl} s & := & \langle a(a+b)^*\rangle p & \qquad C & := & \{t, \neg p, u\} \\ t & := & \langle b(a+b)^*\rangle p & \qquad D & := & \{s, t, \neg p, u\} \end{array}$$

$$u := [bb^*](\neg p \wedge t) \qquad E := \{p\}$$

We start the modal reasoner with the satisfiable clause C. There is one claim C^t to be realized. We need a clause that supports the formulas $\langle(a+b)^*\rangle p$ and $[b^*](\neg p \wedge t)$. The regular reasoner and the propositional reasoner determine C^t and D^s as possible successor clauses and successor literals. The modal reasoner rejects C^t since it would introduce the loop C^tC^t. The pair D^s is fine and adds the clause D and the link C^tD^s. The claim C^t is now realized. However, the new clause D has two unrealized claims D^s and D^t. To realize D^s, we need a clause that supports the formula $\langle(a+b)^*\rangle p$. The regular and the propositional reasoner yield the pairs $E^{\langle 1\rangle p}$, $\{s\}^s$, and $\{t\}^t$. We choose $E^{\langle 1\rangle p}$ and add the clause E and the link $D^tE^{\langle 1\rangle p}$. It remains to realize D^t. To do so, we need a clause that supports the formulas $\langle(a+b)^*\rangle p$ and $[b^*](\neg p \wedge t)$. As before, the regular and the propositional reasoner yield C^t and D^s. Both are fine. We choose C^t and add the link D^tC^t. This gives us a model for the initial clause C. A graphical representation of the model looks as follows:



Example 3.2 Consider the following literals:

$$s := \langle a(a+b)^*\rangle \neg p \qquad u := [a(b+a)^*]p$$
$$t := \langle b(a+b)^*\rangle \neg p \qquad v := [b(b+a)^*]p$$

Here is a closed tableau for the unsatisfiable clause $\{s, u\}$:

$$C_1 = \{s, u\}$$

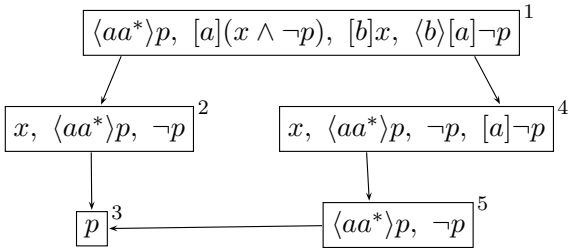
$C_2 = \{s, p, u, v\}$	$C_3 = \{t, p, u, v\}$
$C_1^sC_2^s$	$C_1^sC_3^t$
$C_4 = \{t, p, u, v\}$	$C_5 = \{s, p, u, v\}$
$C_2^sC_4^t$	$C_3^tC_5^s$

The tableau is closed since all possible links for the claims C_4^t and C_5^s introduce loops. For instance, for C_4^t the regular and the propositional reasoner yield the links $C_4^tC_2^s$ and $C_4^tC_4^t$. Note that the clause names C_i do not act as prefixes. They are only used for explanatory purposes.

Example 3.3 Due to the clausal form, the extension of our tableau method to nominals is straightforward. When we add a new clause to a branch, we add to the

new clause all literals that occur in clauses of the branch that have a nominal in common with the new clause. This takes care of nominal propagation. Clauses and links that are already on the branch remain unchanged.

Consider the clause $C = \{\langle aa^*\rangle p,\ [a](x \wedge \neg p),\ [b]x,\ \langle b\rangle[a]\neg p\}$. The initial tableau just consisting of C can be developed into a maximal branch as shown below (graphical representation). The numbers indicate the order in which the clauses are introduced. When clause 4 is introduced, nominal propagation from clause 2 takes place. Note that we obtain a model of all clauses on the branch by taking the clauses 1, 3, 4, and 5 as states and the triples $1a4$, $1b4$, $4a5$, and $5a3$ as transitions.



4 Language-Theoretic Semantics

We define a language-theoretic semantics for programs that treats formulas as atomic objects. This semantics is the base for the regular reasoner and decouples it from the propositional reasoner. It is also essential for the correctness proofs of the modal reasoner. The semantics is an adaption of the language-theoretic model of Kleene algebras with tests [12].

The letters A, B range over finite sets of formulas. A *guarded string* is a finite sequence $Aa_1A_1 \dots a_nA_n$ where $n \geq 0$. The letters σ and τ range over guarded strings. Every program corresponds to a set of guarded strings, which can be seen as runs of the program. For instance, the program $(pa)^*b\neg p$ corresponds to the set of all guarded strings $A_1aA_2 \dots aA_naA_{n+1}bA_{n+2}$ such that $n \geq 1$, $p \in A_1 \cap \dots \cap A_n$, and $\neg p \in A_{n+2}$ (there are no restrictions on A_{n+1}).

The *length* $|\sigma|$ *of a guarded string* $\sigma = Aa_1A_1 \dots a_nA_n$ is n. We use For to denote the set of all formulas. A *language* is a set of guarded strings. For languages L and L' and sets of formulas A we define the following:

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{L}A & := & \{\, B \mid A \subseteq B \subseteq_{\text{fin}} \text{For} \,\} & L^0 & := & \mathcal{L}\emptyset \\ L \cdot L' & := & \{\, \omega A\omega' \mid \omega A \in L,\ A\omega' \in L' \,\} & L^{n+1} & := & L \cdot L^n \\ & & & L^* & := & \bigcup_{n\in\mathbb{N}} L^n \end{array}$$

where ω, ω' range over partial, possibly empty guarded strings. Note that $L^* = \mathcal{L}\emptyset \cup (L - \mathcal{L}\emptyset) \cdot L^*$. We assign to every program α a *language* $\mathcal{L}\alpha$:

$$\mathcal{L}a \ := \ \{\, AaB \mid A, B \subseteq_{\text{fin}} \text{For} \,\} \qquad\qquad \mathcal{L}(\alpha + \beta) \ := \ \mathcal{L}\alpha \cup \mathcal{L}\beta$$

$$\begin{aligned} \mathcal{L}s &:= \mathcal{L}\{s\} & \mathcal{L}(\alpha\beta) &:= \mathcal{L}\alpha \cdot \mathcal{L}\beta \\ \mathcal{L}1 &:= \mathcal{L}\emptyset & \mathcal{L}\alpha^* &:= (\mathcal{L}\alpha)^* \end{aligned}$$

Note that $\mathcal{L}(s^*) = \mathcal{L}1 = \mathcal{L}((s+t)^*)$.

Given an interpretation $\mathcal{I}$, we define the relations $\xrightarrow{\sigma}_{\mathcal{I}} \subseteq |\mathcal{I}| \times |\mathcal{I}|$ by induction on the structure of σ:

$$\begin{aligned} X \xrightarrow{A}_{\mathcal{I}} Y &\iff X = Y \text{ and } \mathcal{I}, X \models A \\ X \xrightarrow{Aa\sigma}_{\mathcal{I}} Y &\iff \mathcal{I}, X \models A \text{ and } \exists Z\colon\ X \xrightarrow{a}_{\mathcal{I}} Z \text{ and } Z \xrightarrow{\sigma}_{\mathcal{I}} Y \end{aligned}$$

Proposition 4.1

(i) $X \xrightarrow{\alpha}_{\mathcal{I}} Y \iff \exists\, \sigma \in \mathcal{L}\alpha\colon\ X \xrightarrow{\sigma}_{\mathcal{I}} Y$

(ii) $\mathcal{I}, X \models \langle\alpha\rangle s \iff \exists\, \sigma \in \mathcal{L}\alpha\ \exists Y\colon\ X \xrightarrow{\sigma}_{\mathcal{I}} Y$ *and* $\mathcal{I}, Y \models s$

(iii) $\mathcal{I}, X \models [\alpha]s \iff \forall\, \sigma \in \mathcal{L}\alpha\ \forall Y\colon\ X \xrightarrow{\sigma}_{\mathcal{I}} Y$ *implies* $\mathcal{I}, Y \models s$

5 Regular DNF

We now describe the regular reasoner. The regular reasoner relies on the language-theoretic semantics and ignores the propositional and modal aspects of the language.

A program is *basic* if it has the form $a\alpha$, and *normal* if it is 1 or basic. Intuitively, a regular DNF of a program α is a decomposition of α into normal programs $\beta_1, \ldots, \beta_n$ such that $\alpha \equiv \beta_1 + \cdots + \beta_n$ (or, more formally, $\mathcal{L}\alpha = \mathcal{L}\beta_1 \cup \cdots \cup \beta_n$). This simple intuition does not account for tests. The program p, for instance, cannot be represented by any set of normal programs. Hence formally we proceed as follows.

We use $\mathcal{F}\alpha$ to denote the set of all formulas that occur in α as subprograms. For instance, $\mathcal{F}(a\neg p + b\langle ap\rangle q) = \{\neg p,\ \langle ap\rangle q\}$. Formulas that occur as programs are called *tests*. Note that $\mathcal{F}\alpha$ does not include tests that occur in tests occurring in α.

Proposition 5.1 *If $s \in \mathcal{F}\alpha$, then, for every t, $|s| < |\neg s| < |\langle\alpha\rangle t| \leq |[\alpha]t|$.*

A *guarded program* is a pair $A\alpha$ where A is a set of formulas and α is a program. A guarded program $A\alpha$ is *normal* if α is normal. The language of a guarded program is $\mathcal{L}(A\alpha) := \mathcal{L}A \cdot \mathcal{L}\alpha$. A *regular DNF* is a function $\mathcal{D}$ that maps every program α to a finite set $\mathcal{D}\alpha$ of normal guarded programs such that:

(i) $\mathcal{L}\alpha = \bigcup_{B\beta \in \mathcal{D}\alpha} \mathcal{L}(B\beta)$

(ii) If $B\beta \in \mathcal{D}\alpha$, then $B \cup \mathcal{F}\beta \subseteq \mathcal{F}\alpha$.

The regular reasoner computes a regular DNF. Kleene's theorem (regular expressions translate into finite automata) [16] suggests that regular DNFs exist. We give a naive algorithm that computes a regular DNF. For space reasons we omit the correctness proof. The algorithm employs the following inference rules for guarded

programs.

$$\frac{Aa}{Aa1} \qquad \frac{As}{(A\,;s)1} \qquad \frac{As\beta}{(A\,;s)\beta} \qquad \frac{A1\beta}{A\beta} \qquad \frac{A(\alpha_1+\alpha_2)}{A\alpha_1\,,\ A\alpha_2} \qquad \frac{A(\alpha_1+\alpha_2)\beta}{A\alpha_1\beta\,,\ A\alpha_2\beta}$$

$$\frac{A(\alpha_1\alpha_2)\beta}{A\alpha_1\alpha_2\beta} \qquad \frac{A\alpha^*}{A1\,,\ A\alpha\alpha^*} \qquad \frac{A\alpha^*\beta}{A\beta\,,\ A\alpha\alpha^*\beta}$$

The notation $A\,;s$ stands for the set $A \cup \{s\}$. Also, we write programs of the form $\alpha(\beta\gamma)$ without parentheses as $\alpha\beta\gamma$. Given a set G of guarded programs, we denote the closure of G under the rules with $\mathcal{R}G$. One can show that $\mathcal{R}G$ describes the same language as G, and that $\mathcal{R}G$ is finite if G is finite. If G is a set of guarded programs, we call a guarded program $A\alpha \in G$ *minimal in* G if there is no $B\alpha \in G$ such that $B \subsetneq A$. We obtain a regular DNF $\mathcal{D}$ by taking for $\mathcal{D}\alpha$ all normal guarded programs in $\mathcal{R}\{\emptyset\alpha\}$ that are minimal in $\mathcal{R}\{\emptyset\alpha\}$.

Example 5.2 Consider the program $(a+b)^*$. We have:

$$\mathcal{R}\{\emptyset(a+b)^*\} = \{\emptyset(a+b)^*,\ \emptyset 1,\ \emptyset(a+b)(a+b)^*,\ \emptyset a(a+b)^*,\ \emptyset b(a+b)^*\}$$
$$\mathcal{D}\{(a+b)^*\} = \{\emptyset 1,\ \emptyset a(a+b)^*,\ \emptyset b(a+b)^*\}$$

Example 5.3 Consider the program $(p+q)^*$ where p, q are predicates. We have $\mathcal{D}\{(p+q)^*\} = \{\emptyset 1\}$. We profit from the optimization that only the minimal guarded programs are taken for the DNF. Otherwise $\mathcal{D}\{(p+q)^*\}$ would contain three further elements: $\{p\}1$, $\{q\}1$, and $\{p,\,q\}1$.

While the above naive algorithm yields a regular DNF for every program α, its efficiency in practice remains to be seen. For programs without tests (i.e., for regular expressions), efficient regular DNFs can be obtained via translation into deterministic finite automata [4]. We expect that similarly efficient regular DNFs also exist for programs with tests.

We fix some computable regular DNF $\mathcal{D}$ for the rest of the paper.

Proposition 5.4

(i) $\mathcal{I}, X \models \langle\alpha\rangle s \iff \exists\ B\beta \in \mathcal{D}\alpha\colon\ \mathcal{I}, X \models B\,;\langle\beta\rangle s$

(ii) $\mathcal{I}, X \models [\alpha]s \iff \forall\ B\beta \in \mathcal{D}\alpha\colon\ (\exists\, t \in B\colon\ \mathcal{I}, X \models \neg t)$ *or* $\mathcal{I}, X \models [\beta]s$

Proof. Follows with Proposition 4.1. □

6 Propositional DNF

The propositional reasoner relies on a support relation from clauses to formulas that abstracts from most modal aspects of the language. We define the *support relation* $C \triangleright s$ by recursion on s.

$$C \triangleright s \iff s \in C \quad \text{if } s \text{ is a literal}$$

$$
\begin{aligned}
C \rhd \neg\neg s &\iff C \rhd s \\
C \rhd s \wedge t &\iff C \rhd s \text{ and } C \rhd t \\
C \rhd s \vee t &\iff C \rhd s \text{ or } C \rhd t \\
C \rhd \langle 1 \rangle s &\iff C \rhd s \\
C \rhd [1] s &\iff C \rhd s \\
C \rhd \langle \alpha \rangle s &\iff \exists B\beta \in \mathcal{D}\alpha\colon\ (\forall t \in B\colon\ C \rhd t) \text{ and } C \rhd \langle \beta \rangle s && \text{if } \alpha \text{ not normal} \\
C \rhd [\alpha] s &\iff \forall B\beta \in \mathcal{D}\alpha\colon\ (\exists t \in B\colon\ C \rhd \neg t) \text{ or } C \rhd [\beta] s && \text{if } \alpha \text{ not normal}
\end{aligned}
$$

The last two equivalences of the definition employ the regular DNF $\mathcal{D}$ fixed above. The recursion terminates since either the size of the formula is reduced (verify with Proposition 5.1) or the recursion is on a formula $\langle\beta\rangle s$ or $[\beta]s$ where β is normal and s is unchanged. We say C *supports* s if $C \rhd s$. We write $C \rhd A$ and say C *supports* A if $C \rhd s$ for every $s \in A$. Note that $C \rhd D \iff D \subseteq C$ (recall that C and D denote clauses).

Proposition 6.1 *If $C \rhd A$ and $C \subseteq D$ and $B \subseteq A$, then $D \rhd B$.*

Proposition 6.2 *If $\mathcal{I}, X \models C$ and $C \rhd A$, then $\mathcal{I}, X \models A$.*

Proof. Follows with Proposition 5.4. □

We define propositional DNFs as functions that, applied to a formula set A, yield a DNF (in the traditional sense) of the conjunction of the formulas in A, represented as a set of clauses. In other words, we require that a propositional DNF $\mathcal{D}$ applied to a set A satisfies the equivalence $\bigwedge_{s\in A} s \equiv \bigvee_{C\in\mathcal{D}A} \bigwedge_{t\in C} t$.

Formally, a *propositional DNF* is a function $\mathcal{D}$ that maps every finite set A of formulas to a finite set of clauses such that:

(i) $\mathcal{I}, X \models A \iff \exists D \in \mathcal{D}A\colon\ \mathcal{I}, X \models D$.

(ii) $C \rhd A \iff \exists D \in \mathcal{D}A\colon\ D \subseteq C$.

Property (ii) for propositional DNFs immediately implies the following proposition.

Proposition 6.3 *If $C \in \mathcal{D}A$, then $C \rhd A$.*

In the following, we will often use Proposition 6.3 implicitly.

For the termination of the modal reasoner the propositional DNF must have some additional finiteness property. We need a few preparatory definitions. The *variants* of a program α are the basic programs β such that $B\beta \in \mathcal{D}\alpha$ for some B. A *base* is a set U of basic formulas such that $\langle\beta\rangle s \in U$ whenever $\langle a\alpha\rangle s \in U$ and β is a variant of α. A base U *supports a formula* s if the following conditions are satisfied:

(i) U contains every basic formula that occurs in s.

(ii) If $\langle\alpha\rangle t$ occurs in s, α is not basic, and β is a variant of α, then $\langle\beta\rangle t \in U$.

A base *supports a set of formulas* A if it supports every formula $s \in A$.

Proposition 6.4 *Every finite set of formulas is supported by a finite base.*

Proof. Follows from property (ii) for the underlying regular DNF. □

A propositional DNF $\mathcal{D}$ is *finitary* if for every finite set of formulas A and every base U supporting A and every clause $C \in \mathcal{D}A$ it holds that U supports C.

Proposition 6.5 *There is a computable finitary propositional DNF.*

Proof. The definition of the support relation can be seen as a tableau-style decomposition procedure for formulas. Using this procedure, one develops A into a complete tableau. The literals of each open branch yield a clause. All clauses obtained this way constitute a DNF of A.

The direction "$\Leftarrow$" of property (i) for propositional DNFs follows with Proposition 6.2. That the DNF is finitary follows from the fact that the decomposition does not introduce new formulas except for diamond formulas obtained with the finitary regular DNF. □

Example 6.6 Take the regular DNF given in §5 and the propositional DNF given in the proof of Proposition 6.5. We have:

$$\begin{aligned}
\mathcal{D}\{\langle b^*\rangle p\} &= \{\{p\},\ \{\langle bb^*\rangle p\}\} \\
\mathcal{D}\{\langle b^*\rangle p,\ [b^*](q \wedge \neg p)\} &= \{\{\langle bb^*\rangle p,\ q,\ \neg p,\ [bb^*](q \wedge \neg p)\}\} \\
\mathcal{D}\{\langle a^*\rangle p,\ [a^*]\neg p\} &= \emptyset \\
\mathcal{D}\{\langle (a+b)^*\rangle p\} &= \{\{p\},\ \{\langle a(a+b)^*\rangle p\},\ \{\langle b(a+b)^*\rangle p\}\}
\end{aligned}$$

For the third example note that $[a^*]\neg p$ is the complement of $\langle a^*\rangle p$.

We fix some computable and finitary propositional DNF $\mathcal{D}$ for the rest of the paper.

7 Diamond Expansion and Nominal Propagation

We now return to the modal reasoner, which was first explained in §3. The modal reasoner builds a tableau where each branch contains clauses and links. The goal consists in constructing a branch where every claim is realized with a link and some further conditions are satisfied. We first make precise how the modal reasoner derives new clauses.

An *expansion* of a claim $C^{\langle a\alpha\rangle s}$ is a claim $D^{\langle\beta\rangle s}$ such that $B\beta \in \mathcal{D}\alpha$ and $D \in \mathcal{D}(B\,;\langle\beta\rangle s \cup \mathcal{R}_a C)$ for some B. The following proposition formulates an important property of expansions (the proposition will not be needed later).

Proposition 7.1 *Let C^s be a claim such that $s \in C$ and let $\mathcal{I}$ satisfy C. Then there exists an expansion D^t of C^s such that $\mathcal{I}$ satisfies D.*

A *link* is a pair C^sD^t of two claims such that $s \in C$ and there is an expansion E^t of C^s such that $E \subseteq D$. A *quasi-branch* is a finite set Γ of clauses and links such that $\{C, D\} \subseteq \Gamma$ whenever $C^sD^t \in \Gamma$. A quasi-branch Γ *realizes a claim* C^s if Γ contains some link C^sD^t. A base *supports a quasi-branch* Γ if it supports every

clause of Γ. An interpretation $\mathcal{I}$ *satisfies a quasi-branch* Γ (or is a *model of* Γ) if $\mathcal{I}$ satisfies every clause in Γ.

We call a clause *nominal* if it contains a nominal. Let Γ be a quasi-branch and A be a set of formulas. We realize *nominal propagation* with the notation

$$A^\Gamma \;:=\; A \cup \{\, s \mid \exists x \in A\; \exists C \in \Gamma\colon\; x \in C \wedge s \in C \,\}$$

Note that A^Γ is the least set of formulas that contains A and all clauses $C \in \Gamma$ that have a nominal in common with A. Thus $(A^\Gamma)^\Gamma = A^\Gamma$. Moreover, $A^\Gamma = A$ if A contains no nominal.

Proposition 7.2 *If an interpretation satisfies* Γ *and* C*, it satisfies* C^Γ*.*

Proposition 7.3 *Let* U *be a base that supports a quasi-branch* Γ*,* C^s *be a claim such that* $s \in C \in \Gamma$*, and* D^t *be an expansion of* C^s*. Then* U *supports* D^Γ*.*

8 Branches and Expansion Rule

A quasi-branch that realizes all its claims does not necessarily have a model. To guarantee the existence of a model, we impose certain conditions on quasi-branches that act as invariants of the modal reasoner. One of the conditions is loop freeness.

Example 8.1 Consider the clause $C = \{\langle aa^*\rangle\neg p,\ p, q,\ [aa^*](p \wedge q)\}$. Note that C is unsatisfiable, and that $\{C,\ C^{\langle aa^*\rangle\neg p} C^{\langle aa^*\rangle\neg p}\}$ is a quasi-branch that realizes every claim. The link of this quasi-branch describes a loop.

A *path in a quasi-branch* Γ is a sequence ${C_1}^{s_1} \dots {C_n}^{s_n}$ of claims such that:

(i) $\forall i \in [1,n]\colon\; C_i^\Gamma = C_i$.

(ii) $\forall i \in [1,n-1]\; \exists D\colon\; C_i^{s_i} D^{s_{i+1}} \in \Gamma$ and $D^\Gamma = C_{i+1}$.

A *loop in a quasi-branch* Γ is a path ${C_1}^{s_1} \dots {C_n}^{s_n}$ in Γ such that $n \geq 2$ and ${C_n}^{s_n} = {C_1}^{s_1}$. A *branch* is a quasi-branch Γ that satisfies the following conditions:

- *Functionality:* If $C^s D^t \in \Gamma$ and $C^s E^u \in \Gamma$, then $D^t = E^u$.
- *Loop-freeness:* There is no loop in Γ.
- *Nominal coherence:* If $C \in \Gamma$, then $C^\Gamma \in \Gamma$.

The *core of a branch* Γ is $\mathcal{C}\Gamma := \{\, C \in \Gamma \mid C^\Gamma = C \,\}$. A branch Γ is *evident* if Γ realizes $C^{\langle\alpha\rangle s}$ for all $\langle\alpha\rangle s \in C \in \mathcal{C}\Gamma$. We will show that every evident branch has a model. The modal reasoner works on branches and applies the following expansion rule:

Expansion Rule
If $\langle\alpha\rangle s \in C \in \mathcal{C}\Gamma$ and Γ does not realize $C^{\langle\alpha\rangle s}$,
then expand Γ to all branches $\Gamma\,; D^\Gamma\,; C^{\langle\alpha\rangle s}(D^\Gamma)^t$
such that D^t is an expansion of $C^{\langle\alpha\rangle s}$ and D^Γ is a clause.

Note that a single clause always yields a branch. So the modal reasoner can start with any clause.

Proposition 8.2 *The modal reasoner terminates on every branch.*

Proof. Since branches are finite by definition, we know by Proposition 6.4 that the initial branch is supported by a finite base. By Proposition 7.3 we know that the expansion rule only adds clauses that are supported by the initial base. The claim follows since a finite base can only support finitely many clauses. □

Given termination, the correctness of the modal reasoner can be established by showing two properties:

(i) *Model Existence:* Every evident branch has a model.

(ii) *Soundness:* Every satisfiable clause can be developed into an evident branch.

9 Model Existence

Proposition 9.1 *Let* Γ *be an evident branch and* $\langle\alpha\rangle s \in C \in \mathcal{C}\Gamma$. *Then there exists a unique path* $C^{\langle\alpha\rangle s} \dots D^{\langle 1\rangle s}$ *in* Γ.

Proof. The path exists since Γ is loop-free and realizes every claim with a clause in $\mathcal{C}\Gamma$. The path is unique since Γ is functional. □

Lemma 9.2 *If* $X \xrightarrow{a}_{\mathcal{I}} Y$ *and* $\mathcal{I}, Y \models B\,;\langle\beta\rangle s$ *and* $B\beta \in \mathcal{D}\alpha$, *then* $\mathcal{I}, X \models \langle a\alpha\rangle s$.

Proof. Follows with Propositions 4.1 and 5.4. □

The model existence proof requires a somewhat involved induction, which we realize with the following lemma.

Lemma 9.3 *Let* Γ *be an evident branch and* $\mathcal{I}$ *be an interpretation such that:*

- $|\mathcal{I}| = \mathcal{C}\Gamma$
- $C \xrightarrow{a}_{\mathcal{I}} D \iff \exists \alpha, s, t, E\colon\ C^{\langle a\alpha\rangle s} E^t \in \Gamma$ *and* $D = E^\Gamma$ *for all actions* a
- $C \in \mathcal{I}p \iff p \in C$ *for all predicates* p
- $\mathcal{I}x = C \iff x \in C$ *for all nominals* x *that occur in* Γ

Let $|\mathcal{F}\alpha| := \max\{\, |s| \mid s \in \mathcal{F}\alpha \,\}$. *Then for all* $n \in \mathbb{N}$:

(i) *For every path* $C^{\langle\alpha\rangle s} \dots D^{\langle 1\rangle s}$ *in* Γ *such that* $|\mathcal{F}\alpha|, |s| < n$:
$\mathcal{I}, C \models \langle\alpha\rangle s$.

(ii) *For all* C, D, σ, α, s *such that* $|\mathcal{F}\alpha|, |s| < n - 1$:
If $C \triangleright [\alpha]s$, $\sigma \in \mathcal{L}\alpha$, *and* $C \xrightarrow{\sigma}_{\mathcal{I}} D$, *then* $D \triangleright s$.

(iii) *For all* C, s *such that* $C \in \mathcal{C}\Gamma$ *and* $|s| = n$:
If $C \triangleright s$, *then* $\mathcal{I}, C \models s$.

Proof. By induction on n. See [11] for details. □

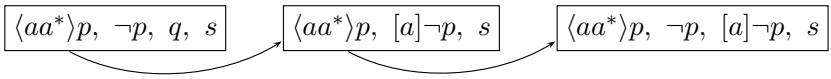
Theorem 9.4 (Model Existence) *Every evident branch has a finite model.*

Proof. Follows with Lemma 9.3 (iii). See [11] for details. □

10 Soundness

We have now arrived at the crucial part of the correctness proof. Ideally, we would like to show that a satisfiable branch with an unrealized claim can always be expanded. However, this is not true.

Example 10.1 Consider the following branch where $s := [aa^*](q \vee [a]\neg p)$:



The branch is satisfiable. Still it is impossible to realize the claim for the third clause since each of the two possible expansions introduces a loop.

Following [10], we solve the problem with the notion of a straight model. A straight model requires that all links on the branch make progress towards the fulfillment of the diamond literal they serve. Every satisfiable initial branch has a straight model, and every unrealized claim on a branch with a straight model $\mathcal{I}$ can be expanded such that $\mathcal{I}$ is a straight model of the expanded branch.

Let $\mathcal{I}$ be an interpretation and A be a set of formulas. The *depth of A and $\langle\alpha\rangle s$ in $\mathcal{I}$* is defined as

$$\delta_{\mathcal{I}} A(\langle\alpha\rangle s) := \min\{\, |\sigma| \mid \sigma \in \mathcal{L}\alpha \text{ and } \exists X, Y \in |\mathcal{I}|\colon\ \mathcal{I}, X \models A \text{ and } X \xrightarrow{\sigma}_{\mathcal{I}} Y \text{ and } \mathcal{I}, Y \models s \,\}$$

where $\min \emptyset = \infty$ and $n < \infty$ for all $n \in \mathbb{N}$.

Proposition 10.2 *$\delta_{\mathcal{I}} As < \infty$ iff $\mathcal{I}$ satisfies $A\,;s$.*

Proof. Follows with Proposition 4.1. □

In particular, we have $\delta_{\mathcal{I}} Cs < \infty$ for every $s \in C \in \Gamma$ if $\mathcal{I}$ is a model of Γ.

A link $C^s D^t$ is *straight* for an interpretation $\mathcal{I}$ if $\delta_{\mathcal{I}} Cs > \delta_{\mathcal{I}} Dt$ whenever $\delta_{\mathcal{I}} Cs > 0$. A *straight model* of a quasi-branch Γ is a model of Γ such that every link $C^s D^t \in \Gamma$ is straight for $\mathcal{I}$.

Proposition 10.3 *Let $\mathcal{I}$ be a model of a quasi-branch Γ. Then $\delta_{\mathcal{I}} As = \delta_{\mathcal{I}} A^{\Gamma} s$.*

Lemma 10.4 (Straightness) *A quasi-branch that has a straight model contains no loops.*

Proof. By contradiction. Let $\mathcal{I}$ be a straight model of a quasi-branch Γ and ${C_1}^{s_1} \ldots {C_n}^{s_n}$ be a loop in Γ. Then $n \geq 2$ and ${C_1}^{s_1} = {C_n}^{s_n}$. It suffices to show that $\delta_{\mathcal{I}} C_i s_i > \delta_{\mathcal{I}} C_{i+1} s_{i+1}$ for all $i \in [1, n-1]$. Let $i \in [1, n-1]$. Then $s_i = \langle a\alpha\rangle t$, ${C_i}^{s_i} D^{s_{i+1}} \in \Gamma$, and $D^{\Gamma} = C_{i+1}$ for some a, α, t, and D. Since every $\sigma \in \mathcal{L}(a\alpha)$ contains the action a, we have $\delta_{\mathcal{I}} C_i s_i > 0$. Since ${C_i}^{s_i} D^{s_{i+1}}$ is straight for $\mathcal{I}$, $\delta_{\mathcal{I}} C_i s_i > \delta_{\mathcal{I}} D s_{i+1}$. Hence $\delta_{\mathcal{I}} C_i s_i > \delta_{\mathcal{I}} C_{i+1} s_{i+1}$ by Proposition 10.3 since $D^{\Gamma} = C_{i+1}$. □

Theorem 10.5 (Soundness) *Let $\mathcal{I}$ be a straight model of a branch Γ and let $\langle\alpha\rangle s \in C \in \Gamma$ such that Γ does not realize $C^{\langle\alpha\rangle s}$. Then there is an expansion D^t of $C^{\langle\alpha\rangle s}$ such that $\Gamma\,; D^\Gamma\,; C^{\langle\alpha\rangle s}(D^\Gamma)^t$ is a branch and $\mathcal{I}$ is a straight model of $\Gamma\,; D^\Gamma\,; C^{\langle\alpha\rangle s}(D^\Gamma)^t$.*

Proof. Since $\mathcal{I}$ satisfies C, by Proposition 10.2, there is some $\sigma \in \mathcal{L}\alpha$ and $X, Y \in |\mathcal{I}|$ be such that $|\sigma| = \delta_{\mathcal{I}} C(\langle\alpha\rangle s)$ and $\mathcal{I}, X \models C$ and $X \xrightarrow{\sigma}_{\mathcal{I}} Y$ and $\mathcal{I}, Y \models s$. Since $\langle\alpha\rangle s$ is a literal, $\alpha = a\beta$ and $\sigma = Aa\tau$ for some a, β, A, and $\tau \in \mathcal{L}\beta$. Let $Z \in |\mathcal{I}|$ be such that $X \xrightarrow{a}_{\mathcal{I}} Z$ and $Z \xrightarrow{\tau}_{\mathcal{I}} Y$. Let $B\gamma \in \mathcal{D}\beta$ such that $\tau \in \mathcal{L}(B\gamma)$. Then $\mathcal{I}, Z \models B$ and $\mathcal{I}, Z \models \langle\gamma\rangle s$ by Proposition 4.1 (ii). Moreover, $\mathcal{I}, Z \models \mathcal{R}_a C$. Thus, by property (i) for propositional DNFs, there is some $D \in \mathcal{D}(B\,; \langle\gamma\rangle s \cup \mathcal{R}_a C)$ such that $\mathcal{I}, Z \models D$. Clearly, $D^{\langle\gamma\rangle s}$ is an expansion of $C^{\langle\alpha\rangle s}$ and, since $\tau \in \mathcal{L}(B\gamma)$, $\delta_{\mathcal{I}} D(\langle\gamma\rangle s) \leq |\tau| = |\sigma| - 1 < \delta_{\mathcal{I}} C(\langle\alpha\rangle s)$. Then, by Proposition 10.3, $\delta_{\mathcal{I}} D^\Gamma(\langle\gamma\rangle s) < \delta_{\mathcal{I}} C(\langle\alpha\rangle s)$. Therefore, $\mathcal{I}$ is a straight model of $\Gamma\,; D^\Gamma\,; C^{\langle\alpha\rangle s}(D^\Gamma)^{\langle\gamma\rangle s}$. Moreover, $\Gamma\,; D^\Gamma\,; C^{\langle\alpha\rangle s}(D^\Gamma)^{\langle\gamma\rangle s}$ satisfies the nominal coherence and functionality conditions. □

11 Final Remarks

The main innovation of the present paper over our previous paper [10] is the notion of a finitary regular DNF. This makes it possible to cover all PDL programs and still have transparent correctness proofs.

It is straightforward to extend the clausal tableau method for HPDL to satisfaction formulas $@_x s$. To deal with such formulas, one adds an additional expansion rule at the modal level as presented in [10]. Also, the optimizations for the modal level of clausal tableaux discussed in [10] carry over to HPDL.

It can be seen by a straightforward analysis that the decision procedure utilizing the naive regular reasoner presented in § 5 and the tableau-based propositional reasoner sketched in Proposition 6.5 runs in NEXPTIME.

We expect that the clausal method can be extended to difference modalities. Less clear is the possibility of extending the method to converse modalities. As recently shown by Goré and Widmann [8], converse modalities can be efficiently dealt with by Pratt-style decision procedures. Their treatment of converse, however, does not seem to carry over to our approach. On the other hand, Pratt-style procedures do not seem compatible with nominals (see [10,19]). Hence, developing a gracefully degrading decision procedure for a logic featuring eventualities, nominals, and converse modalities at the same time remains a challenging open problem.

Acknowledgment

We would like to thank a referee for valuable remarks that helped to improve the paper.

References

[1] Abate, P., R. Goré and F. Widmann, *An on-the-fly tableau-based decision procedure for PDL-satisfiability*, in: C. Areces and S. Demri, editors, *Proc. 5th Workshop on Methods for Modalities (M4M-5)*, Electr. Notes Theor. Comput. Sci. **231** (2009), pp. 191–209.

[2] Areces, C. and B. ten Cate, *Hybrid logics*, in: P. Blackburn, J. van Benthem and F. Wolter, editors, *Handbook of Modal Logic*, Studies in Logic and Practical Reasoning **3**, Elsevier, 2007 pp. 821–868.

[3] Baader, F., *Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles*, Technical Report RR-90-13, DFKI (1990).

[4] Berry, G. and R. Sethi, *From regular expressions to deterministic automata*, Theor. Comput. Sci. **48** (1986), pp. 117–126.

[5] De Giacomo, G. and F. Massacci, *Combining deduction and model checking into tableaux and algorithms for converse-PDL*, Inf. Comput. **162** (2000), pp. 117–137.

[6] Fischer, M. J. and R. E. Ladner, *Propositional dynamic logic of regular programs*, J. Comput. System Sci. (1979), pp. 194–211.

[7] Goré, R. and F. Widmann, *An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability*, in: R. A. Schmidt, editor, *CADE 2009*, LNCS **5663** (2009), pp. 437–452.

[8] Goré, R. and F. Widmann, *Optimal tableaux for propositional dynamic logic with converse*, in: J. Giesl and R. Hähnle, editors, *IJCAR 2010*, LNCS **6173** (2010), pp. 225–239.

[9] Harel, D., D. Kozen and J. Tiuryn, "Dynamic Logic," The MIT Press, 2000.

[10] Kaminski, M. and G. Smolka, *Terminating tableaux for hybrid logic with eventualities*, in: J. Giesl and R. Hähnle, editors, *IJCAR 2010*, LNCS **6173** (2010), pp. 240–254.

[11] Kaminski, M. and G. Smolka, *Clausal tableaux for hybrid PDL*, Technical report, Saarland University (2011). URL www.ps.uni-saarland.de/Publications/details/KaminskiSmolka:2011:HPDL.html

[12] Kozen, D. and F. Smith, *Kleene algebra with tests: Completeness and decidability*, in: D. van Dalen and M. Bezem, editors, *CSL'96*, LNCS **1258** (1996), pp. 244–259.

[13] Kozen, D. and J. Tiuryn, *Logics of programs*, in: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, Elsevier, 1990 pp. 789–840.

[14] Passy, S. and T. Tinchev, *PDL with data constants*, Inf. Process. Lett. **20** (1985), pp. 35–41.

[15] Passy, S. and T. Tinchev, *An essay in combinatory dynamic logic*, Inf. Comput. **93** (1991), pp. 263–332.

[16] Perrin, D., *Finite automata*, in: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, Elsevier, 1990 pp. 1–57.

[17] Pratt, V. R., *A near-optimal method for reasoning about action*, J. Comput. System Sci. **20** (1980), pp. 231–254.

[18] Sattler, U. and M. Y. Vardi, *The hybrid μ-calculus*, in: R. Goré, A. Leitsch and T. Nipkow, editors, *IJCAR 2001*, LNCS **2083** (2001), pp. 76–91.

[19] Widmann, F., "Tableaux-based Decision Procedures for Fixed Point Logics," Ph.D. thesis, Australian National University (2010).