Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 218 (2008) 231–240

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Propositional Dynamic Logic with Program Quantifiers

Daniel Leivant

Indiana University, Bloomington

Abstract

We consider an extension **QPDL** of Segerberg-Pratt's Propositional Dynamic Logic **PDL**, with program quantification, and study its expressive power and complexity. A mild form of program quantification is obtained in the calculus μ**PDL**, extending **PDL** with recursive procedures (i.e. context free programs), which is known to be Π^1_1-complete. The unrestricted program quantification we consider leads to complexity equivalent to that of second-order logic (and second-order arithmetic), i.e. outside the analytical hierarchy. However, the deterministic variant of **QPDL** has complexity Π^1_1.

Keywords: Propositional dynamic logic with program quantification, μ**PDL, QPDL**

1 Introduction

One reason for studying propositional modal logics is to distill the essential logical components of the topic considered, e.g. time, certainty, knowledge, or the effect of imperative programs. Another is to develop practical tools for reasoning and implementation. Decidability, preferably of manageable complexity, is central to the second goal, but not to the first. Adding program quantifiers to **PDL** is of interest as a powerful conceptual framework. Its complexity, however, is on par with full second order logic, far exceeding other propositional formalisms.

An implicit and limited form of higher-order quantification is present already in fixpoints. The propositional fixpoint logics of Pratt [8] and Kozen [6] incorporate fixpoint over propositions, and are both well-known to be decidable. An extension of **PDL** with context-free programs seems to have been proposed first in [3], with striking decidability and undecidability results in subsequent works. We consider a syntactically uniform formalism for such extensions of **PDL**, $\boldsymbol{\mu}$**PDL**, in which a μ operator over programs is used. Although $\boldsymbol{\mu}$**PDL** is Π^1_1-complete [3], it is of great interest because it represents the essence of recursive procedures, i.e. is a prime case of the first rationale above for propositional logics of programs.

doi:10.1016/j.entcs.2008.10.014

The main focus of this paper is a further generalization of $\boldsymbol{\mu}$**PDL**, namely the extension **QPDL** of **PDL** with unrestricted quantification over programs. Extending propositional modal logics with quantifiers over propositions goes back at least to [10], which considered quantification in temporal logic; a deductive calculus for this logic was developed in [5].

Enhancing **PDL** with *propositional* quantification already yields an undecidable formalism, as observed in [9]. We show that quantification over *programs* leads to undecidability of virtually the worst kind: the set of valid formulas is not in the analytical hierarchy (i.e. is not definable in second-order arithmetic). We also show in passing how to interpret in **QPDL** the formalism $\boldsymbol{\mu}$**PDL**, as well as propositional quantification and the global box operator (used jointly in [9] to interpret Kozen's μ-calculus).

Finally, we define a deterministic variant of **QPDL**, and observe that its validity problem is Π^1_1. This is of interest because, in contrast, no complexity penalty is paid for the presence of nondeterminism in **PDL** or in Kozen's μ-calculus.

2 PDL with recursive procedures

2.1 Syntax and semantics of μPDL

The formalism $\boldsymbol{\mu}$**PDL** is an extension of Segerberg-Pratt's propositional dynamic logic **PDL** with definition of programs by (simple) recursion, i.e. where the set of possible traces of each program is a context-free language over atomic programs. Context-free programs have been studied extensively (see e.g.[2]), though we are not aware of any common syntactic framework for them to date.

The syntax of $\boldsymbol{\mu}$**PDL** differs from the syntax of **PDL** only in the formation rules for programs. As for **PDL**, we have atomic-program identifiers $a, b \ldots$. Also, we have for each propositional formula φ the program "test φ", which we write as $?\varphi$. Programs are generated inductively from the atomic identifiers and tests by three operations: composition, union, and fixpoint. That is, if α and β are programs, then so are $\alpha;\beta$, $\alpha \cup \beta$, and $\mu a.\alpha$, where a is an atomic program-identifier. For example, the program $\mu a.(?\top \cup (a;\beta))$ is the same, under the intended semantics to be defined momentarily, as β^*. Similarly, $\mu c.(?\top \cup a; c; b)$ is the same as the program $a^\Delta b^\Delta = \{a^n b^n \mid n \geqslant 0\}$.

More generally, we may consider simultaneous recursion, $\mu a_1 \ldots a_m.(\alpha_1 \ldots \alpha_m).i$ (for $i = 1..m$). [1] Simultaneous recursion can be used to define any context-free program (in the sense of [2]); for example $(\mu s, a, b.(?\top \cup Ab \cup Ba,\ As \cup Baa,\ Bs \cup Abb).1$ is the program $P \subseteq \{A, B\}^*$ consisting of traces with an equal number of A's an B's.

As for **PDL**, we may generalize the definitions above, and allow as tests arbitrary formulas of **PDL**, rather than purely propositional formulas. Such tests are often referred to as *rich tests* [2]. Thus, formulas and programs are defined by a joint

[1] Of course, concrete syntax would require parentheses, and the usual precedence conventions would spare the need to display all of them.

structural recurrence. However, since a μ-binding $\mu a.\alpha$ should apply only when α is monotone with respect to a, we need to define, as part of the recurrence, program-identifier occurring positively and negatively. (Note that, absent rich tests, atomic-programs are always positive in programs, so this issue is moot.) If e is a program or a formula, we write $P(e)$ for the set of atomic-program identifiers occurring positively in e, and $N(e)$ for the ones occurring negatively.

Thus, one defines by joint syntactic recurrence the programs and formulas e, as well as the sets $P(e)$ and $N(e)$. The salient cases are these.

- For a propositional-identifier p, $P(p) = N(p) = \emptyset$. For a program-identifier a $P(a) = \{a\}$ and $N(a) = \emptyset$.
- $P(\psi \rightarrow \varphi) = N(\psi) \cup P(\varphi)$, $N(\psi \rightarrow \varphi) = P(\psi) \cup N(\varphi)$.
- If ψ is a formula, then $?\psi$ is a program, and $P(?\psi) = P(\psi)$, $N(?\psi) = N(\psi)$.
- If α is a program and φ a formula, then $[\alpha]\varphi$ is a formula, with $P([\alpha]\varphi) = N(\alpha) \cup P(\varphi)$ and $N([\alpha]\varphi) = P(\alpha) \cup N(\varphi)$.
- If α is a program, and $a \notin N(\alpha)$, then $\mu a.\alpha$ is a program, with $P(\mu a.\alpha) = P(\alpha) - \{a\}$ and $N(\mu a.\alpha) = N(\alpha)$.

  More generally, if $\boldsymbol{\alpha} = \alpha_1 \ldots \alpha_m$ are programs, and $\boldsymbol{a} = a_1 \ldots a_m$ where $a_i \notin \cup_j N(\alpha_j)$, then $\mu\boldsymbol{a}.\boldsymbol{\alpha}.i$ is a program ($i = 1 \ldots m$).

The semantics of $\boldsymbol{\mu}$**PDL** in a transition (Kripke) structure $\mathcal{K}$ is defined like for **PDL**, with $\mu a.\alpha$ defined as the Knaster-Tarski fixpoint $\cup_n \alpha^n$ where $\alpha^0 = \emptyset$ and $\alpha^{n+1} = \alpha(\alpha^n) \equiv [\alpha^n/a]\alpha$. More generally, $\mu\boldsymbol{a}.\boldsymbol{\alpha} \equiv \mu a_1 \ldots a_m.(\alpha_1 \ldots \alpha_m)$ is the $2m$-ary relation on the set $|\mathcal{K}|$ of states, obtained as the fixpoint $\cup_n \boldsymbol{\alpha}^n$, where $\boldsymbol{\alpha}^0 = \emptyset$, $\boldsymbol{\alpha}^{n+1} = [\boldsymbol{\alpha}^n/\boldsymbol{a}]\boldsymbol{\alpha}$. naturally, $\mu\boldsymbol{a}.\boldsymbol{\alpha}.i$ is the binary relation obtained as the projection of the $2m$-ary $\mu\boldsymbol{a}.\boldsymbol{\alpha}$ on the i'th and $(m+i)$'th arguments.

2.2 The expressive power of μPDL

Theorem 2.1 *The μ-calculus is interpretable in $\boldsymbol{\mu}$**PDL**.*

Proof. The proof idea is to represent the proposition-fixpoint $\mu p.\varphi$ by a program-fixpoint, by representing a propositional identifier p (whose semantics is simply a set of states) by a program identifier a. Suppose p is positive in $\varphi = \varphi(p)$. Let $\varphi^0 = \bot$, and $\varphi^{n+1} = \varphi(\varphi^n)$. Thus $\mu p.\varphi$ is semantically equivalent to the infinite disjunction $\vee_n \varphi^n$. We use the formula $\langle a \rangle \top$ (a a fresh program identifier) to represent, p, with the set of states where p is true intended to correspond to the set of states where a is active.

Let $\alpha(a)$ be the program $?\,\varphi(\langle a \rangle \top)$. Let $\alpha^0 = \emptyset$, $\alpha^{n+1} = \alpha(\alpha^n)$. Thus $\mu a.\alpha$ is semantically equivalent to the infinite union $\cup_n \alpha^n$. We prove that the formula φ^n is semantically equivalent to $\langle \alpha^n \rangle \top$, proceeding by induction on n. The case $n = 0$

is immediate. Assuming $\langle\alpha^n\rangle\top \equiv \varphi^n$, we have

$$\begin{aligned}
\langle\alpha^{n+1}\rangle\top &\equiv \langle\,\alpha(\alpha^n)\,\rangle\top \\
&\equiv \langle\,?\,\varphi(\langle\alpha^n\rangle\top)\,\rangle\top && \text{(Dfn of } \alpha) \\
&\equiv \varphi(\langle\alpha^n\rangle\top) && \text{(semantic of tests)} \\
&\equiv \varphi(\varphi^n) && \text{(IH)} \\
&\equiv \varphi^{n+1}
\end{aligned}$$

We thus have the semantic equalities

$$\begin{aligned}
\mu p\,\varphi &\equiv \textstyle\bigvee_n \varphi^n \\
&\equiv \textstyle\bigvee_n \langle\alpha^n\rangle\top \\
&\equiv \langle\cup\alpha^n\rangle\top \\
&\equiv \langle\mu a.\alpha(a)\rangle\top
\end{aligned}$$

□

In stark contrast with Kozen's μ-calculus, which is decidable, **μPDL** is highly undecidable:

Theorem 2.2 [3] *The validity problem for formulas of* **μPDL** *is* Π^1_1*-complete.*

This high undecidability of **μPDL** does not void, however, the value of deductive calculi for **μPDL**, just as deductive calculi for Arithmetic and for higher-order logic (both highly undecidable) remain of both practical and conceptual interest. We discuss the axiomatics of **μPDL** in [7].

3 Quantification over programs

3.1 Syntax and semantics of **QPDL**

QPDL is obtained from **PDL** by allowing quantifiers ranging over programs. We refer to identifiers for atomic-programs, with no program constructs. Thus, **QPDL** formulas are generated inductively from propositional identifiers by propositional connectives, the modal operators $[a]$ and $\langle a\rangle$ (a a program identifier), and quantification: if φ is a formula, then so are $\forall a\ \varphi(a)$ and $\exists a\ \varphi(a)$. We also refer to a more user-friendly variant of **QPDL**, which we denote **QPDL**$^+$, with programs generated from atomic programs and tests using composition, union, and μ. We shall see, however, that **QPDL**$^+$ is interpretable in **QPDL**.

It is convenient to distinguish between atomic program constants and variables. In a transition (Kripke) structure $\mathcal{K}$, the semantic interpretation of a program constant is given as part of the structure. The semantics of a program variable is given by an *action environment* η, i.e. a mapping that assigns a binary relation on states to each free program-variable present.

The semantics of a formula φ in a transition structure $\mathcal{K}$ and at a state s relative to an environment η is then given by the obvious clauses; in particular, [2]

$$\mathcal{K}, s, \eta \models \forall a\ \varphi \qquad \text{IFF} \qquad \mathcal{K}, s, \eta[a := A] \models \varphi \quad \text{for all } A \subseteq |\mathcal{K}|^2$$

3.2 Expressive power of program quantification

Quantification over propositions is considered in [9], where the modal operator $\Box$ of global truth is also used, to allow the definition of the propositional μ-operator. Here $\Box\,\varphi$ is true in a state s of a transition structure $\mathcal{K}$ iff φ is true in all states.

Proposition 3.1 *Quantification over propositions, as well as the global operator $\Box$, are interpretable in* **QPDL**.

Proof. A formula $\forall p\ \varphi(p)$ is semantically equivalent to $\forall a\ \varphi(\langle a\rangle\top)$, and $\Box\varphi$ is semantically equivalent to $\forall a\ [a]\varphi$. $\Box$

To interpret program μ-operators we observe that containment between programs, $\alpha \subseteq \beta$, is expressible as

$$\Box\,\forall p\ \langle\alpha\rangle\,p \to \langle\beta\rangle\,p$$

Indeed, if $\alpha \subseteq \beta$, then the formula above holds trivially. For the converse, suppose towards contradiction that s, t are states of a transition structure $\mathcal{K}$, such that $s\xrightarrow{\alpha}t$, but not $s\xrightarrow{\beta}t$. Let p_0 be the proposition true in t only. Then $\langle\alpha\rangle\,p_0 \to \langle\beta\rangle\,p_0$ fails in s, and so the formula above fails in $\mathcal{K}$.

Lemma 3.2 *Suppose a is positive in $\alpha = \alpha(a)$. If $\beta \subseteq \gamma$ then $\alpha(\beta) \subseteq \alpha(\gamma)$.*

Proof. Straightforward induction on α. $\Box$

Lemma 3.3 *Suppose a is positive in $\alpha = \alpha(a)$. If d is a program for which $\alpha(d) \subseteq d$ in all states, then $\mu a.\alpha \subseteq d$.*

Proof. We show that $\alpha^n \subseteq d$ for all n, by induction on n, where $\alpha^0 = \emptyset$ and $\alpha^{n+1} = \alpha(\alpha^n)$. For $n = 0$ we trivially have $\emptyset \subseteq d$.

Assuming $\alpha^n \subseteq d$, We have $\alpha^{n+1} = \alpha(\alpha^n) \subseteq \alpha(d)$ by Lemma 3.2. Since we assume $\alpha(d) \subseteq d$, it follows that $\alpha^{n+1} \subseteq d$. $\Box$

3.3 μPDL is interpretable in QPDL

Theorem 3.4 **QPDL$^+$**, *and hence also* **μPDL**, *are interpretable in* **QPDL**.

Proof. We prove by induction on syntax that every formula φ of **QPDL$^+$** is semantically equivalent to a formula of **QPDL**.

The only non-trivial case is for φ of the form $\langle\alpha\rangle\varphi$. We claim that if φ and all tests in α are expressible in **QPDL**, then so is the formula $\langle\alpha\rangle\varphi$. From this it

[2] We write $|\mathcal{K}|$ for the set of states of $\mathcal{K}$.

readily follows that if all subformulas of $\langle\alpha\rangle\varphi$ are expressible in **QPDL**, then so is $\langle\alpha\rangle\varphi$ itself, thus establishing this main case of the induction. We prove the claim by induction on α.

The induction base, with α a program-identifier, is trivial by definition of **QPDL**. For compound α we have the following cases.

- **Test:** A formula $\langle ?\psi\rangle\varphi$ is equivalent to $\psi\wedge\varphi$, which is expressible in **QPDL** by assumption.

- **Composition:** $\langle\beta;\gamma\rangle\varphi$ is equivalent to $\langle\beta\rangle\langle\gamma\rangle\varphi$. $\langle\gamma\rangle\varphi$ φ is expressible, by IH applied to γ, and so $\langle\beta\rangle\langle\gamma\rangle\varphi$ is expressible by IH applied to β.

- **Union:** $\langle\beta\cup\gamma\rangle\varphi$ is equivalent to $\langle\beta\rangle\varphi \ \vee\ \langle\gamma\rangle\varphi$, which is expressible by IH.

- **Recursion** Note that a formula $\langle\,\mu a.\alpha(a)\,\rangle\,\varphi$ is semantically equivalent to

$$\forall d(\ (\alpha(d)\subseteq d)\ \rightarrow\ \langle d\rangle\,\varphi\ ) \tag{1}$$

  On the one hand, instantiating d in (1) with $\mu a.\alpha(a)$ we obtain $(\alpha(\mu a.\alpha)\subseteq \mu a.\alpha)\ \rightarrow\ \langle\mu a.\alpha\rangle\,\varphi$. Since the premise holds by the definition of $\mu a.\alpha$, we obtain $\langle\mu a.\alpha(a)\rangle\,\varphi$.

  Conversely, if d satisfies $\alpha(d)\subseteq d$, then by Lemma 3.3 we have $\mu a.\alpha\subseteq d$, so $\langle\,\mu a.\alpha(a)\,\rangle\,\varphi$ implies $\langle d\rangle\,\varphi$. □

4 Super-analytical complexity of QPDL

4.1 Interpreting general grammars

Recall that a *(general) grammar* $G=(A,\mathtt{S},R)$ over an alphabet Σ consists of an alphabet $A\supset\Sigma$, a distinguished $\mathtt{S}\in A-\Sigma$, and a set R of rules, i.e. pairs (w,v) where $w\in A^{+}-\Sigma^{+}$ and $v\in A^{*}$. ($N=A-\Sigma$ is the set of *non-terminals*. A *yield* relation $\Rightarrow_G$ on A^* is generated inductively from R: if $w\rightarrow v$ is a rule in R, then $xwy\Rightarrow xvy$ for all $x,y\in A^*$. The language *generated by* G is $\mathcal{L}(G)=\{w\in\Sigma^*\mid \mathtt{S}\Rightarrow_G^* w\}$. Recall that a language is generated by a grammar iff it is semi-decidable (i.e. RE) [4, §9.2]. In particular, acceptance[3] by a Turing acceptor M can be simulated as a derivation by a grammar G that starts with the accepting state of M, using rules that proceed "backwards", i.e. from each local configuration C of M to local-configurations C' for which $C'\Rightarrow_M C$.

Given an alphabet $A=\Sigma\cup N$ as above, we consider its symbols as atomic program identifiers, and each string $w=\xi_1\cdots\xi_k\in A^*$ as the program $\xi_1;\ \cdots\ ;\xi_k$. In particular, the empty string λ is construed as the program `skip`, i.e. $?\top$.

Fix a propositional identifier p. For a rewrite rule $\rho=(w\rightarrow v)$ over A, let φ_ρ

[3] One considers, w.l.o.g., Turing acceptors that erase their tape before entering the accepting state.

be the **QPDF**$^+$ formula

$$\forall a, b\ \langle avb\rangle\, p \to \langle awb\rangle\, p$$

and consider the formula

$$\varphi_G \quad \equiv \quad (\wedge_{\rho\in G}\, \varphi_\rho) \ \to\ p \to \langle \mathtt{S}\rangle\, p$$

Proposition 4.1 *For each general grammar G, $\lambda \in \mathcal{L}(G)$ iff φ_G is valid.*

Proof. Suppose $w \Rightarrow^n_G v$. Consider a transition structure $\mathcal{K}$ and state s therein. If the premise of φ_G holds at s, then so does $\langle v\rangle\, p \to \langle w\rangle\, p$, by a trivial induction on n. Assuming $\lambda \in \mathcal{L}(G)$, i.e. $\mathtt{S} \to^*_G \lambda$, we therefore get in s that $\langle\lambda\rangle\, p \to \langle\mathtt{S}\rangle\, p$. Since $\langle\lambda\rangle\, p$ is equivalent to p, we conclude $p \to \langle\mathtt{S}\rangle p$, i.e. the conclusion of φ_G. Thus $\mathcal{K}, s \models \varphi_G$.

Conversely, suppose $\lambda \notin \mathcal{L}(G)$. Consider the transition structure $\mathcal{F}$ whose states are the strings $w \in A^*$, with each program identifier $\xi \in A$ interpreted as the mapping $w \mapsto \xi w$, and with p true at w iff $w \Rightarrow^*_G \lambda$. Then φ_G fails at λ, since both $\wedge_{\rho\in G}\, \varphi_\rho$ and p are true there, but $\langle\mathtt{S}\rangle\, p$ is not. □

4.2 Oracle Grammars

Of course, Proposition 4.1 implies that **QPDL** is undecidable, which we already know from Harel's Theorem 2.2 above. The interest in Propostion 4.1, though, is that it generalizes easily to relativized computing. General grammars can be equipped with a symbolic version of orcales, as follows. Define an *oracle-grammar* to be a grammar with two additional distinguished nonterminals, $\mathtt{P}$ and $\mathtt{N}$ (intended to represent the positive and negative query-answer, respectively). These are not used, though, in the source of any rule of G. The yield relation $\Rightarrow_G$ above is augmented as follows: given a language $W \subseteq \Sigma^*$ (the "oracle") the relation $\Rightarrow_{G,W}$ is generated by clausess as above, as well as $x\mathtt{P}y \Rightarrow_{G,L} xwy$ for $w \in W$, and $x\mathtt{N}y \Rightarrow_{G,L} xwy$ for $w \notin W$.

The language *generated by G modulo orcale W* is

$$\mathcal{L}^P(W) = \{w \in \Sigma^* \mid \mathtt{S} \Rightarrow^*_{G,W} w\}$$

The "backwards" simulation of Turing acceptors by general grammars, outlined above, easily extends to a simulation of oracle Turing acceptors by oracle grammars. Oracle grammars with $k > 1$ oracles are defined similarly, using non-terminals $\mathtt{P}_i, \mathtt{N}_i$ $(i = 1 \ldots k)$. We obtain:

Proposition 4.2 *Deciding whether a given k-oracle grammar G satisfies*

$$\forall W_1 \exists W_2 \cdot\cdot\cdot \cdots W_k\ (\,\lambda \in \mathcal{L}^{\boldsymbol{W}}(G)\,)$$

is Π^1_k-complete.

This underlies a representation of oracle grammars in **QPDL**, generalizing Proposition 4.1, by using the program indeitifiers $\mathtt{P}_i$ ad $\mathtt{N}_i$ to represent oracle W_i, i.e.

we intend to have $w \in W_i$ just in case $w = \xi_1 \cdots \xi_n$, as the composition of atomic programs, is "in $\mathtt{P}_i$", and $w \notin W_i$ when w is "in $\mathtt{N}_i$". However, since programs can be far more general than sets of execution traces of atomic programs, some care is needed in setting up this representation.

4.3 Expressing properties of structures and programs in **QPDL**

We use the following abbreviations, where $A = \{\xi_1, \ldots, \xi_m\}$ (excluding the oracle identifiers).

$$\begin{array}{rll}
\alpha \subseteq \beta & \text{for} & \forall p\, \Box(\langle\alpha\rangle\, p \rightarrow \langle\beta\rangle\, p) \\
\mathrm{Det}(\alpha) & \text{for} & \forall p\, \Box(\langle\alpha\rangle\, p \rightarrow [\alpha]\, p) \\
\mathrm{Det}(A) & \text{for} & \wedge_i \mathrm{Det}(\xi_i) \\
\langle A\rangle\, \varphi & \text{for} & \langle \xi_1 \cup \cdots \cup \xi_m\rangle\, \varphi \\
 & \equiv & \vee_i \langle\xi_i\rangle\, \varphi \\
[A]\, \varphi & \text{for} & \wedge_i [\xi_i]\, \varphi \\
\alpha \subseteq A^* & \text{for} & \forall p\, ((\Box(p \rightarrow [A]\, p) \rightarrow \Box(p \rightarrow [\alpha]\, p) \\
\alpha \lhd A^* & \text{for} & (\alpha \subseteq A^*) \ \wedge\ \mathrm{Det}(\alpha) \\
\alpha \oplus \beta = A^* & \text{for} & (\alpha \subseteq A^*) \ \wedge\ (\beta \subseteq A^*) \\
 & & \wedge\, \forall a\, (a \lhd A^*) \ \rightarrow\ (a \subseteq \alpha)\, \bar{\vee}\, (a \subseteq \beta) \\
 & & \quad (\oplus \text{ and } \bar{\vee} \text{ are disjoint union and disjoint disjunction})
\end{array}$$

For $\boldsymbol{W} = (W_1 \ldots W_k)$, where $W_i \subseteq A^*$, let $\eta^{\boldsymbol{W}}$ be the environment defined by $\eta^{\boldsymbol{W}}(\mathtt{P}_i) = [\![W_i]\!]_{\mathcal{K}}$ and $\eta^{\boldsymbol{W}}(\mathtt{N}_i) = [\![\bar{W}_i]\!]_{\mathcal{K}}$.

Lemma 4.3 *Let G be a k-oracle grammar, $W_1 \ldots W_k \subseteq A^*$. If $\lambda \in \mathcal{L}^{\boldsymbol{W}}(G)$ then*

$$\mathcal{K}, \eta^{\boldsymbol{W}} \models (\wedge_{\rho\in G}\, \varphi_\rho) \ \rightarrow\ p \rightarrow \langle \mathtt{S}\rangle\, p$$

Proof. More generally, we prove by induction on n that if $w \Rightarrow^n_{G,\boldsymbol{W}} v$, then

$$\mathcal{K}, \eta^{\boldsymbol{W}} \models (\wedge_{\rho\in G}\, \varphi_\rho) \ \rightarrow\ \langle v\rangle\, p \rightarrow \langle w\rangle\, p$$

Note that oracle-productions such as $x\mathtt{P}y \Rightarrow xwy$ $(w \in W)$ are not represented textually in φ_G, but rather in the fact that $\langle xwy\rangle\, p$ implies semantically $\langle x\mathtt{P}y\rangle\, p$ in the environment $\eta^{\boldsymbol{W}}$. □

Consider the canonical structure $\mathcal{F}$ (defined in the proof of proposition 4.1).

Lemma 4.4 *Let G be a k-oracle grammar. If*

$$\mathcal{F}, \eta \models (\mathtt{P}_i \oplus \mathtt{N}_i = A^*) \ \wedge\ (\wedge_{\rho\in G}\, \varphi_\rho) \ \wedge\ (\forall p\, p \rightarrow \langle \mathtt{S}\rangle\, p)$$

then $\lambda \in \mathcal{L}^{\boldsymbol{W}}(G)$, where $W_i = \eta(\mathtt{P}_i)$. □

Combining the two Lemmas, we obtain, by induction on k, that the statement

$$\forall W_1 \exists W_2 \cdots\cdots W_k \ (\lambda \in \mathcal{L}^{\boldsymbol{W}}(G))$$

is true, just in case the **QPDL** formula

$$\begin{aligned}&\forall \mathtt{P}_1, \mathtt{N}_1 \ (\mathtt{P}_1 \oplus \mathtt{N}_1 = A^* \ \rightarrow \\ &\quad \exists \mathtt{P}_2, \mathtt{N}_2 \ (\mathtt{P}_2 \oplus \mathtt{N}_2 = A^* \wedge \\ &\qquad \cdots \\ &\qquad\qquad (\wedge_{\rho \in G} \ \varphi_\rho) \ \rightarrow \forall p \ p \rightarrow \langle \mathtt{S} \rangle \ p \cdots) \,)\end{aligned}$$

is valid.

Combining this with Proposition 4.2, we conclude:

Theorem 4.5 *The validity problem for* **QPDL** *is not in the analytical hierarchy.*

4.4 Deterministic **QPDL**

A well-known deterministic variant of **PDL** is obtained by replacing the branching and iterative commands $\cup$ and $*$ by their guarded variants, `case` and `while` respectively, and positing semantically that atomic programs are deterministic, i.e. partial functions on states, rather than relations. A deterministic variant **DQPDL** of **QPDL** is obtained just by stipulating that program identifiers are deterministic. A friendlier formalism **DQPDL**$^+$ is obtained from **QPDL**$^+$ by also using the program construct `case`, in place of $\cup$. It is easy to see that the proof of Theorem 3.4 can be replicated to yield an interpretation of **DQPDL**$^+$ in **DQPDL**.

For **PDL** the restriction to a deterministic variant is of little consequence, since the unrestricted version is already decidable, while it offers conceptual purity and elegant axiomatization. But the inclusion in the mix of program quantification changes that. The reason is that deterministic programs cannot be used to code sets of execution traces, so a quantification on programs is then no stronger than quantification over individual states.

We indeed have:

Theorem 4.6 *The validity problem for deterministic* **QPDL** *is in* Π^1_1.

Proof. For any given countable deterministic structure $\mathcal{K}$ the truth of Σ_k **DQPDL** formulas φ is at level Σ_k of the arithmetical hierarchy, and is therefore Π^1_1. Quantifying over sets to verify universal validity, we obtain that the validity problem of formulas in deterministic structures is Π^1_1. □

5 Conjectures and research directions

- ***µ*PDL** is decidable when μ is drastically restricted. For example, restricting μ to regular grammars gives only regular programs, i.e. **PDL**. But [1] show that decidability holds for **PDL** extended with context-free programs that are accepted by what they call *simple-minded* PDAs, in which the action (push or pop) is determined by just the symbol scanned, regardless of the state and the stack. A characterization of simple-minded CFLs (or of a broader class for which **PDL** is still decidable) would probably yield an interesting restriction on μ-programs.

- We conjecture that $\boldsymbol{\mu}$**PDL** is equivalent to the restriction of **QPDL** to formulas where no quantifier has its variable occur in the scope of another quantifier.
- Higher order quantification can be ramified, a notion that goes back to Whitehead and Russell's type theory, and Parson's set theory. It is similarly possible to ramify program quantifiers. We conjecture that the resulting system is decidable, and that a corresponding modification of our deductive calculus is complete.
- It is well known that the valid termination assertions in full (first-order) dynamic logic form an RE set. Similarly, the validity of termination assertions of $\boldsymbol{\mu}$**PDL**, is decidable (an analogous result for propositional quantification is in [9]). Here we may admit as "termination assertions" all formulas in which the diamond operator occurs only positively, and the box only negatively. Is there a generalization for **QPDL**? For example, what about quantified formulas with diamond and box similarly restricted? Recall that our undecidability proof depends heavily on the diamond occurring both positively and negatively.

References

[1] D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM jour. Comput.*, 22:857–874, 1993.

[2] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cabridge, MA, 2000.

[3] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logics of nonregular programs. *J. Comput. Sys. Sci.*, 25:222–243, 1983.

[4] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[5] Yonit Kesten and Amir Pnueli. A complete proof system for QPTL. *Journal of Logic and Computation*, 12(5):701–745, 2002.

[6] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[7] Daniel Leivant. Propositional dynamic logic for recursive procedures. Proceedings of the Second IFIP Conference on Veried Software (VSTTE08), to appear.

[8] Vuaghan Pratt. A decidable mu-calculus (preliminary report). In *Proceedings of the twenty-second IEEE Symposium on Foundations of Computer Science*, pages 421–427, Los Angles, 1981. Computer Society press.

[9] N.V. Shilov. Program schemata vs. automata for decidability of program logics. *Theoretical Computer Science*, 175:15–27, 1997.

[10] S.P. Sistla, M.Y. Vardi, and P. Wolper. The completementation problem for Büchi automata with application to termporal logic. *Theoretical Computer Science*, 49:217–237, 1987.