

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

[www.elsevier.com/locate/entcs](http://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 Π1-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 Π1.

1

1

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

# 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](#_bookmark17)] and Kozen [[6](#_bookmark14)] 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](#_bookmark10)], with striking decidability and undecidability results in subsequent works. We consider a syntactically uniform formalism for such extensions of PDL, *μ*PDL, in which a *μ* operator over programs is used. Although *μ*PDL is Π1-complete [[3](#_bookmark10)], 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.

1

1571-0661 © 2008 Published by Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2008.10.014

The main focus of this paper is a further generalization of *μ*PDL, namely the extension QPDL of PDL with unrestricted quantification over programs. Extend- ing propositional modal logics with quantifiers over propositions goes back at least to [[10](#_bookmark19)], which considered quantification in temporal logic; a deductive calculus for this logic was developed in [[5](#_bookmark15)].

Enhancing PDL with *propositional* quantification already yields an undecidable formalism, as observed in [[9](#_bookmark18)]. 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 *μ*PDL, as well as propositional quantification and the global box operator (used jointly in [[9](#_bookmark18)] to interpret Kozen’s *μ*-calculus).

Finally, we define a deterministic variant of QPDL, and observe that its validity problem is Π1. 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.

1

# PDL with recursive procedures

* 1. *Syntax and semantics of μPDL*

The formalism *μ*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](#_bookmark11)]), though we are not aware of any common syntactic framework for them to date.

The syntax of *μ*PDL differs from the syntax of PDL only in the formation rules for programs. As for PDL, we have atomic-program identifiers *a, b.. .*. Also, we have for each propositional formula *ϕ* the program “test *ϕ*”, which we write as ?*ϕ*. 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 *α*; *β*, *α* ∪ *β*, and *μa.α*, where *a* is an atomic program-identifier. For example, the program *μa.*(?T∪ (*a*; *β*)) is the same, under the intended semantics to be defined momentarily, as *β*∗. Similarly, *μc.*(?T∪*a*; *c*; *b*) is the same as the program *a*Δ*b*Δ = {*anbn* | *n* ≥ 0}.

More generally, we may consider simultaneous recursion, *μa*1 *... am.*(*α*1 *... αm*)*.i* (for *i* = 1*..m*). [1](#_bookmark0) Simultaneous recursion can be used to define any context-free program (in the sense of [[2](#_bookmark11)]); for example (*μs, a, b.*(?T∪ *Ab* ∪ *Ba, As* ∪ *Baa, Bs* ∪ *Abb*)*.*1 is the program *P* ⊆ {*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](#_bookmark11)]. 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 *μa.α* 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*) = ∅. For a program-identifier *a P* (*a*)= {*a*} and *N* (*a*)= ∅.
* *P* (*ψ* → *ϕ*)= *N* (*ψ*) ∪ *P* (*ϕ*), *N* (*ψ* → *ϕ*)= *P* (*ψ*) ∪ *N* (*ϕ*).
* If *ψ* is a formula, then ?*ψ* is a program, and *P* (?*ψ*)= *P* (*ψ*), *N* (?*ψ*)= *N* (*ψ*).
* If *α* is a program and *ϕ* a formula, then [*α*]*ϕ* is a formula, with *P* ([*α*]*ϕ*) =

*N* (*α*) ∪ *P* (*ϕ*) and *N* ([*α*]*ϕ*)= *P* (*α*) ∪ *N* (*ϕ*).

* If *α* is a program, and *a* /∈ *N* (*α*), then *μa.α* is a program, with *P* (*μa.α*) =

*P* (*α*) − {*a*} and *N* (*μa.α*)= *N* (*α*).

More generally, if *α* = *α*1 *... αm* are programs, and *a* = *a*1 *... am* where *ai* /∈

∪*jN* (*αj*), then *μa.α.i* is a program (*i* = 1 *... m*).

The semantics of *μ*PDL in a transition (Kripke) structure K is defined like for PDL, with *μa.α* defined as the Knaster-Tarski fixpoint ∪*nαn* where *α*0 = ∅ and *αn*+1 = *α*(*αn*) ≡ [*αn/a*]*α*. More generally, *μa.α* ≡ *μa*1 *... am.*(*α*1 *... αm*) is the 2*m*-ary relation on the set |K| of states, obtained as the fixpoint ∪*nαn*, where *α*0 = ∅, *αn*+1 = [*αn/a*]*α*. naturally, *μa.α.i* is the binary relation obtained as the projection of the 2*m*-ary *μa.α* on the *i*’th and (*m* + *i*)’th arguments.

* 1. *The expressive power of μPDL*

Theorem 2.1 *The μ-calculus is interpretable in μ*PDL*.*

Proof. The proof idea is to represent the proposition-fixpoint *μp.ϕ* 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 *ϕ* = *ϕ*(*p*). Let *ϕ*0 = ⊥, and *ϕn*+1 = *ϕ*(*ϕn*). Thus *μp.ϕ* is semantically equivalent to the infinite disjunction

∨*nϕn*. We use the formula ⟨*a*⟩T (*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 *α*(*a*) be the program ? *ϕ*(⟨*a*⟩ T). Let *α*0 = ∅, *αn*+1 = *α*(*αn*). Thus *μa.α* is semantically equivalent to the infinite union ∪*nαn*. We prove that the formula *ϕn* is semantically equivalent to ⟨*αn*⟩ T, proceeding by induction on *n*. The case *n* =0

is immediate. Assuming ⟨*αn*⟩T ≡ *ϕn*, we have

⟨*αn*+1⟩ T ≡ ⟨ *α*(*αn*) ⟩T

≡ ⟨ ? *ϕ*(⟨*αn*⟩ T) ⟩T (Dfn of *α*)

≡ *ϕ*(⟨*αn*⟩ T) (semantic of tests)

≡ *ϕ*(*ϕn*) (IH)

≡ *ϕn*+1

We thus have the semantic equalities

*μp ϕ* ≡ V*n ϕn*

≡ V*n* ⟨*αn*⟩T

≡ ⟨∪*αn*⟩T

≡ ⟨*μa.α*(*a*)⟩T

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

Theorem 2.2 [[3](#_bookmark10)] *The validity problem for formulas of μ*PDL *is* Π1*-complete.*

1

This high undecidability of *μ*PDL does not void, however, the value of deductive calculi for *μ*PDL, 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 *μ*PDL in [[7](#_bookmark16)].

# Quantification over programs

* 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 ⟨*a*⟩ (*a* a program identifier), and quan- tification: if *ϕ* is a formula, then so are 6*a ϕ*(*a*) and E*a ϕ*(*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 K, the semantic interpretation of a program con- stant 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 K and at a state *s* relative to an environment *η* is then given by the obvious clauses; in particular, [2](#_bookmark5)

K*, s,η* |= 6*a ϕ* iff K*, s, η*[*a* := *A*] |= *ϕ* for all *A* ⊆ |K|2

* 1. *Expressive power of program quantiﬁcation*

Quantification over propositions is considered in [[9](#_bookmark18)], where the modal operator of global truth is also used, to allow the definition of the propositional *μ*-operator. Here *ϕ* is true in a state *s* of a transition structure K iff *ϕ* is true in all states.

Proposition 3.1 *Quantiﬁcation over propositions, as well as the global operator*

*, are interpretable in* QPDL*.*

Proof. A formula 6*p ϕ*(*p*) is semantically equivalent to 6*a ϕ*(⟨*a*⟩T), and *ϕ* is semantically equivalent to 6*a* [*a*]*ϕ*.

To interpret program *μ*-operators we observe that containment between pro- grams, *α* ⊆ *β*, is expressible as

6*p* ⟨*α*⟩ *p* → ⟨*β*⟩ *p*

Indeed, if *α* ⊆ *β*, then the formula above holds trivially. For the converse, suppose towards contradiction that *s, t* are states of a transition structure K, such that *s*−*α*→*t*,

*β*

but not *s*−→*t*. Let *p*0 be the proposition true in *t* only. Then ⟨*α*⟩ *p*0 → ⟨*β*⟩ *p*0 fails

in *s*, and so the formula above fails in K.

Lemma 3.2 *Suppose a is positive in α* = *α*(*a*)*. If β* ⊆ *γ then α*(*β*) ⊆ *α*(*γ*)*.*

Proof. Straightforward induction on *α*.

Lemma 3.3 *Suppose a is positive in α* = *α*(*a*)*. If d is a program for which α*(*d*) ⊆ *d*

*in all states, then μa.α* ⊆ *d.*

Proof. We show that *αn* ⊆ *d* for all *n*, by induction on *n*, where *α*0 = ∅ and

*αn*+1 = *α*(*αn*). For *n* = 0 we trivially have ∅⊆ *d*.

Assuming *αn* ⊆ *d*, We have *αn*+1 = *α*(*αn*) ⊆ *α*(*d*) by Lemma [3.2](#_bookmark2). Since we assume *α*(*d*) ⊆ *d*, it follows that *αn*+1 ⊆ *d*.

* 1. *μPDL is interpretable in QPDL*

Theorem 3.4 QPDL+*, and hence also μ*PDL*, are interpretable in* QPDL*.*

Proof. We prove by induction on syntax that every formula *ϕ* of QPDL+ is se- mantically equivalent to a formula of QPDL.

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

2 We write |K| for the set of states of K.

readily follows that if all subformulas of ⟨*α*⟩*ϕ* are expressible in QPDL, then so is

⟨*α*⟩*ϕ* 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 ⟨?*ψ*⟩*ϕ* is equivalent to *ψ* ∧ *ϕ*, which is expressible in QPDL

by assumption.

* Composition: ⟨*β*; *γ*⟩*ϕ* is equivalent to ⟨*β*⟩⟨*γ*⟩*ϕ*. ⟨*γ*⟩*ϕ ϕ* is expressible, by IH applied to *γ*, and so ⟨*β*⟩⟨*γ*⟩*ϕ* is expressible by IH applied to *β*.
* Union: ⟨*β* ∪ *γ*⟩*ϕ* is equivalent to ⟨*β*⟩*ϕ* V ⟨*γ*⟩*ϕ*, which is expressible by IH.
* Recursion Note that a formula ⟨ *μa.α*(*a*) ⟩ *ϕ* is semantically equivalent to

6*d*( (*α*(*d*) ⊆ *d*) → ⟨*d*⟩ *ϕ* ) (1) On the one hand, instantiating *d* in ([1](#_bookmark6)) with *μa.α*(*a*) we obtain (*α*(*μa.α*) ⊆

*μa.α*) → ⟨*μa.α*⟩ *ϕ*. Since the premise holds by the definition of *μa.α*, we obtain

⟨*μa.α*(*a*)⟩ *ϕ*.

Conversely, if *d* satisfies *α*(*d*) ⊆ *d*, then by Lemma [3.3](#_bookmark3) we have *μa.α* ⊆ *d*, so

⟨ *μa.α*(*a*) ⟩ *ϕ* implies ⟨*d*⟩ *ϕ*.

# Super-analytical complexity of QPDL

* 1. *Interpreting general grammars*

Recall that a *(general) grammar G* = (*A,* S*, R*) over an alphabet Σ consists of an alphabet *A* ⊃ Σ, a distinguished S ∈ *A* − Σ, and a set *R* of rules, i.e. pairs (*w, v*) where *w* ∈ *A*+ − Σ+ and *v* ∈ *A*∗. (*N* = *A* − Σ is the set of *non-terminals*. A *yield* relation ⇒*G* on *A*∗ is generated inductively from *R*: if *w* → *v* is a rule in *R*, then *xwy* ⇒ *xvy* for all *x, y* ∈ *A*∗. The language *generated by G* is L(*G*) =

{*w* ∈ Σ∗ | S ⇒∗

*G*

*w*}. Recall that a language is generated by a grammar iff it is

semi-decidable (i.e. RE) [[4](#_bookmark13), §9.2]. In particular, acceptance [3](#_bookmark7) 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*' ⇒*M C*.

Given an alphabet *A* = Σ ∪ *N* as above, we consider its symbols as atomic program identifiers, and each string *w* = *ξ*1 ··· *ξk* ∈ *A*∗ as the program *ξ*1; ··· ; *ξk* . In particular, the empty string *λ* is construed as the program skip, i.e. ?T.

Fix a propositional identifier *p*. For a rewrite rule *ρ* = (*w* → *v*) over *A*, let *ϕρ*

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

be the QPDF+ formula and consider the formula

6*a, b* ⟨*avb*⟩ *p* → ⟨*awb*⟩ *p*

*ϕG* ≡ (∧*ρ*∈*G ϕρ*) → *p* → ⟨S⟩ *p*

Proposition 4.1 *For each general grammar G, λ* ∈ L(*G*) *iff ϕG is valid.*

Proof. Suppose *w* ⇒*n*

*G*

*v*. Consider a transition structure K and state *s* therein.

If the premise of *ϕG* holds at *s*, then so does ⟨*v*⟩ *p* → ⟨*w*⟩ *p*, by a trivial induction

on *n*. Assuming *λ* ∈ L(*G*), i.e. S →∗

*G*

*λ*, we therefore get in *s* that ⟨*λ*⟩ *p* → ⟨S⟩ *p*.

Since ⟨*λ*⟩ *p* is equivalent to *p*, we conclude *p* → ⟨S⟩*p*, i.e. the conclusion of *ϕG*. Thus

K*,s* |= *ϕG*.

Conversely, suppose *λ* /∈ L(*G*). Consider the transition structure F whose states are the strings *w* ∈ *A*∗, with each program identifier *ξ* ∈ *A* interpreted as the

mapping *w* '→ *ξw*, and with *p* true at *w* iff *w* ⇒∗ *λ*. Then *ϕG* fails at *λ*, since both

*G*

∧*ρ*∈*G ϕρ* and *p* are true there, but ⟨S⟩ *p* is not.

* 1. *Oracle Grammars*

Of course, Proposition [4.1](#_bookmark8) implies that QPDL is undecidable, which we already know from Harel’s Theorem [2.2](#_bookmark1) above. The interest in Propostion [4.1](#_bookmark8), 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, P and 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 ⇒*G* above is augmented as follows: given a language *W* ⊆ Σ∗ (the “oracle”) the relation ⇒*G,W* is generated by clausess as above, as well as *x*P*y* ⇒*G,L xwy* for *w* ∈ *W* , and *x*N*y* ⇒*G,L xwy* for *w* /∈ *W* .

The language *generated by G modulo orcale W* is

L*P* (*W* )= {*w* ∈ Σ∗ | S ⇒∗ *w*}

*G,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 P*i,* N*i* (*i* = 1 *... k*). We obtain:

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

6*W*1E*W*2 · · · ··· *Wk* ( *λ* ∈ LW (*G*))

*is* Π1*-complete.*

*k*

This underlies a representation of oracle grammars in QPDL, generalizing Proposition [4.1](#_bookmark8), by using the program indeitifiers P*i* ad N*i* to represent oracle *Wi*, i.e.

we intend to have *w* ∈ *Wi* just in case *w* = *ξ*1 ··· *ξn*, as the composition of atomic programs, is “in P*i*”, and *w* /∈ *Wi* when *w* is “in 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.

* 1. *Expressing properties of structures and programs in* QPDL

We use the following abbreviations, where *A* = {*ξ*1*,... , ξm*} (excluding the oracle identifiers).

*α* ⊆ *β* for 6*p* (⟨*α*⟩ *p* → ⟨*β*⟩ *p*) Det(*α*) for 6*p* (⟨*α*⟩ *p* → [*α*] *p*) Det(*A*) for ∧*i*Det(*ξi*)

⟨*A*⟩ *ϕ* for ⟨*ξ*1 ∪ ··· ∪ *ξm*⟩ *ϕ*

≡ V*i*⟨*ξi*⟩ *ϕ*

[*A*] *ϕ* for ∧*i*[*ξi*] *ϕ*

*α* ⊆ *A*∗ for 6*p* (( (*p* → [*A*] *p*) → (*p* → [*α*] *p*) *α* D *A*∗ for (*α* ⊆ *A*∗) ∧ Det(*α*)

*α* ⊕ *β* = *A*∗ for (*α* ⊆ *A*∗) ∧ (*β* ⊆ *A*∗)

∧ 6*a* (*a* D *A*∗) → (*a* ⊆ *α*) V¯ (*a* ⊆ *β*)

(⊕ and V¯ are disjoint union and disjoint disjunction)

For *W* = (*W*1 *... Wk*), where *Wi* ⊆ *A*∗, let *η*W be the environment defined by

*η*W (P*i*)= [[*Wi*]]K and *η*W (N*i*)= **[***W*¯ *i*]]K.

Lemma 4.3 *Let G be a k-oracle grammar, W*1 *... Wk* ⊆ *A*∗*. If λ* ∈ LW (*G*) *then*

K*,η*

W

|= (∧*ρ*∈*G ϕρ*) → *p* → ⟨S⟩ *p*

Proof. More generally, we prove by induction on *n* that if *w* ⇒*n*

*G,*W

*v*, then

K*,η*

W

|= (∧*ρ*∈*G ϕρ*) → ⟨*v*⟩ *p* → ⟨*w*⟩ *p*

Note that oracle-productions such as *x*P*y* ⇒ *xwy* (*w* ∈ *W* ) are not represented textually in *ϕG*, but rather in the fact that ⟨*xwy*⟩ *p* implies semantically ⟨*x*P*y*⟩ *p* in the environment *η*W .

Consider the canonical structure F (defined in the proof of proposition [4.1](#_bookmark8)).

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

F*,η* |= (P*i* ⊕ N*i* = *A*∗) ∧ (∧*ρ*∈*G ϕρ*) ∧ (6*p p* → ⟨S⟩ *p*)

*then λ* ∈ LW (*G*)*, where Wi* = *η*(P*i*)*.*

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

6*W* E*W* ·· ·· · · *W* (*λ* ∈ LW (*G*)) is true, just in case the QPDL formula

1 2 *k*

6P1*,* N1 (P1 ⊕ N1 = *A*∗ → EP2*,* N2 (P2 ⊕ N2 = *A*∗ ∧

·· ·

(∧*ρ*∈*G ϕρ*) → 6*p p* → ⟨S⟩ *p* ··· ))

is valid.

Combining this with Proposition [4.2](#_bookmark9), we conclude:

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

* 1. *Deterministic* QPDL

A well-known deterministic variant of PDL is obtained by replacing the branch- ing and iterative commands ∪ 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 pro- gram construct case, in place of ∪. It is easy to see that the proof of Theorem [3.4](#_bookmark4) 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* Π1*.*

1

Proof. For any given countable deterministic structure K the truth of Σ*k* DQPDL formulas *ϕ* is at level Σ*k* of the arithmetical hierarchy, and is therefore Π1. Quan- tifying over sets to verify universal validity, we obtain that the validity problem of formulas in deterministic structures is Π1.

1

1

# 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](#_bookmark12)] 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 *μ*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 *μ*PDL, is decidable (an analogous result for propositional quantification is in [[9](#_bookmark18)]). 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.

1. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison- Wesley, Reading, MA, 1979.
2. Yonit Kesten and Amir Pnueli. A complete proof system for QPTL. *Journal of Logic and Computation*, 12(5):701–745, 2002.
3. Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
4. Daniel Leivant. Propositional dynamic logic for recursive procedures. Proceedings of the Second IFIP Conference on Veried Software (VSTTE08), to appear.
5. 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.
6. N.V. Shilov. Program schemata vs. automata for decidability of program logics. *Theoretical Computer Science*, 175:15–27, 1997.
7. S.P. Sistla, M.Y. Vardi, and P. Wolper. The completementation problem for Bu¨chi automata with application to termporal logic. *Theoretical Computer Science*, 49:217–237, 1987.