Generalizing Propositional Dynamic Logic

Marc Pauly *

May 25, 1998

Abstract

A new semantics of Propositional Dynamic Logic (PDL) is given in terms of predicate transformers: Instead of associating the standard state transformer with each program, we model programs by operations on sets of states, operations which map postconditions to weakest liberal preconditions. This yields a logic more general than PDL since new program operations such as angelic choice become expressible. Three issues will be discussed: axiomatization, extension of the framework to total correctness reasoning and the relation to the μ-calculus.

1 Introduction

In Propositional Dynamic Logic (PDL), programs are modeled just like in standard denotational semantics by means of state transformers: Given a model $\mathcal{I}$ of PDL with universe (i.e. set of states) S, a function $g_\pi^{\mathcal{I}} : S \to \mathcal{P}(S)$ is associated with each program π. The intuition behind this semantic model is that programs are functions from input states to sets of possible output states, thus allowing for nondeterminism on the one hand as well as partial programs on the other hand.

Looking at the syntax of PDL however, one finds that the only statements we can make about programs are of the form $[\pi]\varphi$ which has the following semantics:

$$[\pi]\varphi^{\mathcal{I}} = \{s \in S | g_\pi^{\mathcal{I}}(s) \subseteq \varphi^{\mathcal{I}}\} \tag{1}$$

In the computational literature, $[\pi]\varphi$ has been called the *weakest liberal precondition* (WLP, for short) of π under postcondition φ. Since the language

*Center for Mathematics and Computer Science (CWI), Kruislaan 413, 1098 SJ Amsterdam. E-mail: pauly@cwi.nl

of PDL only uses programs to talk about their weakest liberal preconditions under given postconditions, one can try to define the semantics of programs directly in terms of their weakest precondition behavior. Under this approach, we view $[\pi]$ as a predicate transformer, i.e. as a function which when given a predicate φ (the postcondition) returns a predicate $[\pi]\varphi$ (the weakest liberal precondition). Hence, with each program π, a model $\mathcal{I}$ with universe S now associates a function $G_\pi^{\mathcal{I}} : \mathcal{P}(S) \to \mathcal{P}(S)$ such that

$$[\pi]\varphi^{\mathcal{I}} = G_\pi^{\mathcal{I}}(\varphi^{\mathcal{I}}) \tag{2}$$

Given the restricted language of PDL which is limited to statements about weakest liberal preconditions, one could argue that the predicate transformer semantics is the more natural model for programs, as is witnessed by the fact that (2) is a much simpler definition than (1). Outside of the context of PDL, this view has been taken most prominently by Dijkstra in [Dij76] and [DS90].

There is however a crucial question which needs to be addressed when switching from state transformers to predicate transformers. So far we have simply specified predicate transformers as operations on sets of states. This is of course a rather general characterization of programs, and the question is whether all such operations actually correspond to a state transformer. In other words, given any predicate transformer for a program π, can we find a state transformer modeling the same program?

2 Comparing Semantic Expressiveness

When we say that a state transformer g_π *corresponds* to a predicate transformer G_π, we mean that the following equation holds for all X:

$$g_\pi^{-1}[\mathcal{P}(X)] := \{s | g_\pi(s) \subseteq X\} = G_\pi(X) \tag{3}$$

Note that if two transformers correspond, equations (1) and (2) turn out to be the same.

Given a state transformer g_π, it is easy to see that we can find a corresponding predicate transformer G_π, simply by using (3) as a definition. This means that the predicate transformer semantics is at least as expressive as the standard state transformer semantics, for any state transformer has a corresponding predicate transformer. However, the opposite direction does not hold. Call a predicate transformer $G : \mathcal{P}(S) \to \mathcal{P}(S)$ *universally conjunctive* iff $G(\bigcap_{X \in V} X) = \bigcap_{X \in V} G(X)$ for all $V \subseteq \mathcal{P}(S)$. The following result is proved in [Pau97]:

Proposition 1 *State transformers and universally conjunctive predicate transformers are equally expressive, i.e. for every universally conjunctive predicate transformer there is a corresponding state transformer and vice versa.*

The proposition thus implies that unrestricted predicate transformers are more expressive than state transformers, and that a restriction to the universally conjunctive predicate transformers restricts expressiveness to the level of state transformers.

A concrete example of a non-conjunctive operator we gain when allowing for unrestricted predicate transformers is *angelic choice.* Given two programs α and β, we define

$$G_{\alpha\diamond\beta}(X) := G_\alpha(X) \cup G_\beta(X)$$

This indeed implements some kind of angelic choice: We choose the right program execution depending on the postcondition, so angelic choice introduces some kind of teleology into the language. As Hesselink points out in [Hes90], angelic choice is in some way the natural counterpart of the usual nondeterministic choice $\alpha \cup \beta$ which he calls *demonic* choice. But the symmetry on the level of predicate transformers turns out to be misleading.

Proposition 2 *Angelic choice (i.e.* $G_{\alpha\diamond\beta}$*) is not universally conjunctive.*

By the previous proposition, this means that angelic choice cannot be represented by a state transformer. But while this indicates the non-computational character of this operator, it also suggests that there are quite natural operations which can be gained by making PDL more general.

3 General Dynamic Logic

A complexer variant of the following logic was first introduced in [Pau97] as PDL^{pt}, but here we shall simply call it DL for "Dijkstra Logic" or "Dynamic Logic". The shorter name already hints at its more general nature which will be demonstrated subsequently.

Definition 1 (DL**)** *The syntax of* DL *is the same as that of* PDL*. A model for* DL *is a pair* $\mathcal{I} = (S^\mathcal{I}, m^\mathcal{I})$*, where* $S^\mathcal{I}$ *is the set of states or the universe, and for all atomic programs* a*,*

$$m^\mathcal{I}(a) = G^\mathcal{I}_a : \mathcal{P}(S^\mathcal{I}) \to \mathcal{P}(S^\mathcal{I})$$

such that $G_a^{\mathcal{I}}$ *is monotonic.* G_a *denotes the* WLP*-transformer of atomic program* a*. For all atomic propositions* p*,*

$$m^{\mathcal{I}}(p) = p^{\mathcal{I}} \subseteq S^{\mathcal{I}}$$

We extend $G_\alpha^{\mathcal{I}}$ *to non-atomic programs* α *inductively as follows:*

$$\begin{array}{rl} G^{\mathcal{I}}_{\alpha;\beta}(Y) & = G^{\mathcal{I}}_{\alpha}(G^{\mathcal{I}}_{\beta}(Y)) \\ G^{\mathcal{I}}_{\alpha\cup\beta}(Y) & = G^{\mathcal{I}}_{\alpha}(Y) \cap G^{\mathcal{I}}_{\beta}(Y) \\ G^{\mathcal{I}}_{\varphi?}(Y) & = \neg\varphi^{\mathcal{I}} \cup Y \\ G^{\mathcal{I}}_{\alpha^*}(Y) & = \nu X.\ \sigma^{\mathcal{I}}_{\alpha,Y}(X) \\ & \quad \textit{where } \sigma^{\mathcal{I}}_{\alpha,Y}(X) = X \cap Y \cap G^{\mathcal{I}}_{\alpha}(X) \end{array}$$

In this definition, $\nu X.f(X)$ *denotes the greatest fixpoint of the operator* f*, i.e. the largest set* X *such that* $f(X) = X$*. It can be shown that monotonicity is preserved under the program connectives, so this fixpoint always exists. The semantic definitions for pure boolean statements is as usual; for* $[\alpha]\varphi$*, we define* $([\alpha]\varphi)^{\mathcal{I}} = G^{\mathcal{I}}_{\alpha}(\varphi^{\mathcal{I}})$*.*

Note that we really do not assume anything about the atomic predicate transformers except their monotonicity. This means e.g. that in contrast to PDL, $[a]\top$ is not valid in DL, since $G_a^{\mathcal{I}}(true) = true$ does not generally hold anymore. We obtain various restrictions of DL by restricting the atomic predicate transformer, so that e.g. universally conjunctive DL, denoted as $\bigwedge -DL$, refers to DL with the G-transformers of all atomic programs being universally conjunctive.

As expected, PDL and $\bigwedge -DL$ are equivalent. Let τ map any PDL model to a DL model by assigning to any atomic state transformer g_a its corresponding predicate transformer WLP_a, defined in (3). Proposition 1 establishes that τ is injective and that for $\bigwedge -DL$, τ is also surjective. The fundamental result relating DL to PDL can now be stated as follows:

Proposition 3 *For all* PDL *models* $\mathcal{I}$ *and all* PDL *formulas* ϕ

$$\phi^{\mathcal{I}} = \phi^{\tau(\mathcal{I})}$$

As a consequence, ϕ *is valid in* PDL *if and only if* ϕ *is valid in* $\bigwedge -DL$*.*

A further remark on the fixpoint definition used in DL: Call a predicate transformer F and-continuous iff for all denumerable sequences of predicates $C_0 \supseteq C_1 \supseteq C_2 \supseteq \ldots$ we have $F(\bigcap_{i<\omega} C_i) = \bigcap_{i<\omega} F(C_i)$. Since it can be shown that and-continuity is preserved by the program operations, a continuity restriction on the atomic predicate transformers will guarantee an ω-bound on the greatest fixpoint in question. In $\bigwedge -DL$, such a bound can always be guaranteed, since universal conjunctivity is a stronger property than and-continuity. As a consequence, and-continuous DL (i.e. DL with atomic predicate transformers being and-continuous) allows for non-conjunctive operations like angelic choice while still preserving the simpler fixpoint characterization. See [Pau98] for more detail on the issue of continuity.

4 Axiomatization

One would expect that an axiomatization of DL should not differ drastically from the standard PDL axiomatization. In terms of the axioms, this turns out to be the case. The axioms of figure 1 are all valid in DL. For all axioms except the Mix axiom, this is an immediate consequence of the semantic definitions of G_π.

All propositional tautologies	
$[\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$	Composition
$[\alpha \cup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$	Union
$[\psi?]\varphi \leftrightarrow (\psi \rightarrow \varphi)$	Test
$[\alpha^*]\varphi \rightarrow (\varphi \wedge [\alpha][\alpha^*]\varphi)$	Mix

Figure 1: Axioms

The only difference with the axioms of PDL is that we left out the induction axiom:

$$(\varphi \wedge [\alpha^*](\varphi \rightarrow [\alpha]\varphi)) \rightarrow [\alpha^*]\varphi \tag{4}$$

Proposition 4 *The induction axiom 4 is not valid in DL.*

Nonetheless, the axioms of figure 1 together with the inference rules of figure 2 provide a complete axiomatization for DL, as has been shown independently by [Par85] and [Pau].

In [Pau] it is shown that this axiomatization can be extended to various stronger systems up to PDL itself by adding the axioms $[\alpha]\top$ and $[\alpha]\varphi \wedge [\alpha]\psi \rightarrow [\alpha](\varphi \wedge \psi)$. Similarly for angelic choice, we maintain a complete axiomatization by adding $[\alpha\Diamond\beta]\varphi \leftrightarrow [\alpha]\varphi \vee [\beta]\varphi$ as an additional axiom.

$\frac{\varphi,\ \varphi\rightarrow\psi}{\psi}$	Modus Ponens
$\frac{\varphi\rightarrow\psi}{[\alpha]\varphi\rightarrow[\alpha]\psi}$	Monotonicity
$\frac{\psi\rightarrow(\varphi\wedge[\alpha]\psi)}{\psi\rightarrow[\alpha^*]\varphi}$	Transitive Closure

Figure 2: Inference Rules

5 Total Correctness

It has been well-known (see e.g. [Har84]) that PDL is insufficient for reasoning about the total correctness of nondeterministic programs. Since nonterminating programs such as

`while` *true* `do skip`

have infinite computation sequences, checking for termination in PDL involves checking for an infinite branch (or cycle) in the model. For a deterministic program π, the PDL formula $\langle\pi^*\rangle\neg\langle\alpha\rangle\top$ accomplishes the task, but for nondeterministic programs, checking for infinite branches exceeds the expressive power of PDL.

One well-known solution to the problem adds a new predicate $loop(\pi)$ to the language which holds of program π in case the program has at least one diverging execution. The dynamic logic implementing this idea has been proposed already in [HP78], and we present here its more widely used reformulation $LPDL$ (see e.g. [Har84]).

Definition 2 **(**$LPDL$**)** *For every program α, we add $loop(\alpha)$ to the language of PDL. Models of $LPDL$ are just PDL models, and the semantics of the new predicate is defined as follows:*

$$\begin{aligned}
loop(a)^{\mathcal{I}} &= \emptyset \text{ for atomic programs } a\\
loop(p?)^{\mathcal{I}} &= \emptyset\\
loop(\alpha;\beta)^{\mathcal{I}} &= loop(\alpha)^{\mathcal{I}} \cup \langle\alpha\rangle loop(\beta)^{\mathcal{I}}\\
loop(\alpha\cup\beta)^{\mathcal{I}} &= loop(\alpha)^{\mathcal{I}} \cup loop(\beta)^{\mathcal{I}}\\
loop(\alpha^*)^{\mathcal{I}} &= \langle\alpha^*\rangle loop(\alpha)^{\mathcal{I}} \cup \{s_0\in S | \exists s_1\exists s_2\cdots\forall i\geq 0 : (s_i,s_{i+1})\in R_\alpha^{\mathcal{I}}\}
\end{aligned}$$

Note that "looping" here really means divergence. Given that divergence is only caused by some kind of repetition or iteration, the definition stipulates that atomic programs do not diverge. The definition of $loop(\alpha^*)^{\mathcal{I}}$ is crucial:

It states that a program α^* loops in state s if there is an infinite α-sequence starting in s.

The *loop* predicate can also be added to DL. In fact, the only clause in definition 2 which needs to be modified is the one for iteration.

Definition 3 **($DL + loop$)** *$DL{+}loop$ is defined just like DL, except that we add a $loop(\pi)$ predicate for every program π. The semantics of this predicate is defined just as in (2), except that*

$$\begin{array}{rcl} loop(\alpha^*)^{\mathcal{I}} & = & \neg TERM_\alpha^{\mathcal{I}} \\ TERM_\alpha^{\mathcal{I}} & = & \mu X.\ \rho_\alpha^{\mathcal{I}}(X) \\ & & \textit{where } \rho_\alpha^{\mathcal{I}}(X) = X \cup \{s \in S^{\mathcal{I}} | s \notin loop(\alpha)^{\mathcal{I}} \ \wedge \ s \in G_\alpha^{\mathcal{I}}(X)\} \end{array} \tag{5}$$

$\mu X.f(X)$ denotes the least fixpoint of the operator f, i.e. the smallest set X such that $f(X) = X$.

Like in the case of DL and PDL, it can be demonstrated (see [Pau97]) that universal conjunctivity restricts the expressiveness of $DL + loop$ to the level of $LPDL$. In fact, once this is proved, the corresponding result for DL is obtained as a corollary.

Proposition 5 *For all $LPDL$ models $\mathcal{I}$ and all $LPDL$ formulas ϕ*

$$\phi^{\mathcal{I}} = \phi^{\tau(\mathcal{I})}$$

As a consequence, ϕ is valid in $LPDL$ if and only if ϕ is valid in $\bigwedge -DL +$ loop.

Regarding the fixpoint definition used to define $loop(\alpha^*)$, one can again investigate the continuity restriction (or-continuity) which guarantees an upper bound on the fixpoint. It turns out that the programs satisfying this continuity restriction are precisely the bounded nondeterministic programs, i.e. programs with only finitely many different executions. Again, the interested reader is referred to [Pau98] for a more detailed discussion.

6 The μ-calculus

The μ-calculus (see e.g. [Koz83], [Str85]) is a propositional program logic which includes a fixpoint operator over propositions. Programs themselves are always atomic, but besides allowing for modal expressions such as $[a]\varphi$, we also have least-fixpoint expressions such as $\mu Q.f(Q)$ where $f(Q)$ is a formula monotone in the propositional variable Q, denoting the least Q such that $f(Q) = Q$. Since PDL can be viewed as a less expressive restriction of the μ-calculus, some remarks about the relation between DL and the μ-calculus are in order.

First of all, programs of the μ-calculus are relations on states. In that sense, the μ-calculus is no different from PDL, and hence a similar generalization of the μ-calculus which works with predicate transformers instead of state transformers is possible. This means that unlike PDL, DL cannot be regarded as a less expressive restriction of the μ-calculus, since the semantic models are now different.

Secondly, if one is interested in a less fine-grained comparison in terms of validities only, we can still see DL as a restricted μ-calculus. For modal logic, it has been noted in [GH96] that minimal models (i.e. predicate transformer models) can be translated into standard Kripke models such that validity is preserved. The trick is to introduce a representative for every set of states into the domain, and to add a new modality associated to the relation of set-membership. The same trick can be applied to translate DL-models into μ-calculus models (see [Par85]). The resulting Kripke model is an exponential blow up of the original DL-model, and through a suitable translation τ, it can be shown that a formula φ is valid in DL iff $\tau(\varphi)$ is valid in the μ-calculus.

Considering $\bigwedge -DL$, i.e. DL where all atomic predicate transformers are required to be universally conjunctive, yet a different third perspective emerges. As mentioned before, $\bigwedge -DL$ is nothing but PDL, and hence a restriction of the μ-calculus. Adding constructs such as angelic choice to $\bigwedge -DL$ can now be regarded as isolating a richer sub-logic of the μ-calculus: The standard translation of PDL into the μ-calculus (see [Str85]) can easily be extended to angelic choice by translating $[\alpha \Diamond \beta]\varphi$ as $[\alpha]\varphi \vee [\beta]\varphi$. In this light, one can view DL as a PDL-style framework for studying fragments of the μ-calculus.

7 Conclusions & Future Work

The technical results obtained and their discussion have provided evidence that DL provides a useful general framework for thinking about programs. In summary, (1) DL allows us to define operators which were beyond the scope of PDL, (2) while being more general, PDL corresponds to a specific restriction of DL (note the analogy to modal logic where different logics correspond to different frame conditions for the basic system), (3) through its use of fixpoints, DL puts the issue of continuity on the agenda and hence uncovers subtleties invisible under PDL, (4) $\bigwedge -DL$ provides a framework for studying restrictions of the μ-calculus more expressive than PDL.

Besides theoretical issues such as the exact relation between different restrictions of DL, there are also two lines of investigation which go more towards applications. Firstly, we introduced angelic choice as an example of an operator which we gain by switching from PDL to DL. Other (non-computable) operators which model cognitive behavior of some sort and which can be modeled through predicate transformers would further increase the applicability of DL as well as its credibility as a general logic for reasoning about actions. Interaction with research from related disciplines could further enrich DL beyond its computational predecessor. Secondly, it should be realized that the main area where Dijkstra's healthiness conditions (such as universal conjunctivity) have been dropped has been program specification & verification, in particular the tradition which derives programs through stepwise refinement from their specifications (see e.g. [Mor90] and [MV92]). In this approach, specifications are considered on a par with implementable programs. From this perspective, $LPDL$ and DL are progressive steps from a logic of programs towards a logic of specifications. Maybe DL could eventually prove useful in such a context.

References

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1976.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer Verlag, 1990.

[GH96] Olivier Gasquet and Andreas Herzig. From classical to normal modal logics. In Heinrich Wansing, editor, *Proof Theory of Modal Logic.* Kluwer, 1996.

[Har84] David Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II. D. Reidel Publishing Company, 1984.

[Hes90] Wim H. Hesselink. Modalities of nondeterminacy. In W. H. J. Feijen et al., editors, *Beauty is our business: a birthday salute to*

Edsger W. Dijkstra, Texts and Monographs in Computer Science. Springer Verlag, 1990.

[HP78] D. Harel and V. Pratt. Nondeterminism in logics of programs. In *Proceedings of the 5th annual ACM Symposium on Priinciples of Programming Languages*, pages 203–213, 1978.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Mor90] Carroll C. Morgan. *Programming from Specifications*. Prentice-Hall, 1990.

[MV92] C. Morgan and T. Vickers, editors. *On the Refinement Calculus*. Springer Verlag, 1992.

[Par85] Rohit Parikh. The logic of games and its applications. In Marek Karpinski and Jan van Leeuwen, editors, *Topics in the Theory of Computation*, volume 24 of *Annals of Discrete Mathematics*. Elsevier, 1985.

[Pau] Marc Pauly. Axiomatizing weak propositional dynamic logic. Unpublished manuscript.

[Pau97] Marc Pauly. Transforming predicates or updating states - total correctness in dynamic logic and structured programming. Master's thesis, Institute for Logic, Language and Computation, Amsterdam, 1997.

[Pau98] Marc Pauly. Divergence and dynamic logic: A predicate transformer approach. In *Proceedings of Accolade 1997*, 1998.

[Str85] Robert Streett. Fixpoints and program looping: Reductions from the propositional mu-calculus into propositional dynamic logics of looping. In Rohit Parikh, editor, *Logics of Programs*, pages 359–372. Springer, 1985.