COSI-112, COSI-140a: PDL in Prolog

Alex Plotnick

Fall 2006

1 Introduction

This document describes a Prolog implementation of *Propositional Dynamic Logic*[2] (henceforth PDL). The program parses and evaluates PDL formulæ in a user-supplied Kripke frame, and includes a simple *read-eval-print-loop* (REPL) for interactive queries.

2 Syntax

Due to the limited character set available for interactive use (i.e., ASCII), a special syntax is employed for PDL formulæ; the following table gives the mapping between the traditional mathematical notation and the syntax required by the interpreter. Whitespace may be used freely in any formula.

$[\alpha]\varphi$	`[a]p`	Necessity
$\langle\alpha\rangle\varphi$	`<a>p`	Possibility
$\alpha;\beta$	`a;b`	Composition
$\alpha \cup \beta$	`a|b`	Choice
$\alpha*$	`a*`	Iteration
$\varphi?$	`p?`	Test
$\varphi \vee \psi$	`p+q`	Disjunction
$\varphi \wedge \psi$	`p&q`	Conjunction
$\neg\varphi$	`-p`	Negation
$\varphi \rightarrow \psi$	`p->q`	Implication
$\varphi \leftrightarrow \psi$	`p=q`	Equivalence
$\mathbf{0}$	`0`	Falsity

The interpreter also enforces a particular naming convention: the (lowercase) letters `a`–`o` stand for atomic programs; the letters `p`–`t` stand for atomic propositions, and the letters `u`–`z` stand for worlds in the Kripke frame: e.g., the formula `[p]q` is not well-formed, since only a program may be in a modal operator.

3 Semantics

The interpreter works by evaluating the Kripke semantics of the given expression for the supplied model. That is, it constructs a set of worlds or edges between worlds for each atom in the expression, and uses the standard inductively-defined PDL semantics to evaluate non-atomic formulæ. The result of evaluating an expression will always be either the set of worlds in which the expression is true, or the atom `true`, indicating universal truth.

Models are specified as Kripke frames: $\mathfrak{k} = (\mathcal{K}, \mathfrak{m})$, where $\mathcal{K}$ is a set of worlds (or states) $\{u, v, w, \ldots\}$, and $\mathfrak{m}$ is the meaning function that assigns a subset of $\mathcal{K}$ to each atomic proposition and a binary relation on $\mathcal{K}$ to each atomic program. These frames are expressed in the program as Prolog clauses: the clause `k(Worlds)` defines the set of worlds (as a list), and `m/2` defines the meaning function, where `m(Prop, World)` defines an atomic proposition, and `m(Prog, [World1|World2])` defines an atomic program.

For example, consider the frame $\mathfrak{k} = (\mathcal{K}, \mathfrak{m})$ given in [1, p. 132], where:

$$\begin{aligned} \mathcal{K} &= \{u, v, w\} \\ \mathfrak{m}(p) &= \{u, v\} \\ \mathfrak{m}(a) &= \{(u,v), (u,w), (v,w), (w,v)\} \end{aligned}$$

we would use the following program:

```
k([u,v,w]).
m(p, u).
m(p, v).
m(a, [u|v]).
m(a, [u|w]).
m(a, [v|w]).
m(a, [w|v]).
```

4 Usage

Use of the interpreter is perhaps best demonstrated with an example. The file `model1.pl` contains the Prolog code for the model defined in the previous section, and `pdl.pl` contains the code for the interpreter. The following is an actual interaction with Prolog, eliding only compilation messages and blank lines for readability.

```
$ gprolog
GNU Prolog 1.2.16
By Daniel Diaz
Copyright (C) 1999-2002 Daniel Diaz
| ?- [model1].
(1 ms) yes
| ?- [pdl].
```

```
(10 ms) yes
| ?- pdl.
<a>-p & <a>p
 => [u]
[a]-p
 => [v]
[a]p
 => [w]
<a*>[(aa)*]p & <a*>[(aa)*]-p
 => [u,v,w]


(8 ms) no
| ?-
```

The predicate `pdl` enters the REPL, and any non-WFF exits it (here, we've simply used a blank line to terminate the session). Formulæ are typed one per line, and the result is printed on the following line, preceded by `=>`. Expressions of essentially arbitrary complexity are allowed.

The reader may wish to experiment with the slightly more complex four-state model given in [1, p. 134]; in the author's testing, all of the example formulæ given for this model produced the expected results.

5   Limitations

- The parser, being a DCG, is not particularly resilient to errors. Most non-WFFs cause it to fail, but there may be some invalid inputs that cause it to loop. Modulo bugs, the parse tree it produces should be correct with regard to the normal precedence rules of PDL. For debugging purposes, the parse tree may be examined like so:

  ```
  | ?- parse(E).
  <a*>[(aa)*]p

  E = pos(star(a),nec(star(seq(a,a)),p)) ? ;

  no
  | ?-
  ```

- The Algol-like keywords commonly used in PDL programs (e.g., **if, while**, etc.) have not been implemented, but would be simple to add.

- No attempt is made at implementing the decision procedure for PDL: only explicitly specified models are supported by this implementation.

References

[1] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984.

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