

Class 08: Prolog (1)

Held: Wednesday, 9 February 2011

Summary: We begin our exploration of the Prolog programming language.

Related Pages:

- EBoard.
- Lab: Prolog (1).
- Reading: Tate 4.1-4.2.

Notes:

- Reports suggest that you should be able to do the first three problems on assignment 3 in under five hours, but that problem 4 is harder. I'll treat that one as extra credit.
- For problem 4 on assignment 3, you should use the notation from problem 3.
- EC for CS Extra Thursday (Weinman).
- EC for CS Table Friday.

Overview:

- Predicate Logic Programming.
- Prolog Syntax.
- Behind the Scenes: Computation in Prolog.
- Lab.

Predicate Logic Programming

- Prolog is one of the first *predicate logic programming* languages.
- That is, Prolog computes using *predicates* (true/false assertions about their arguments).
- The primary computational model is therefore the definition of predicates and then inquireis using predicates.
- In Prolog, we express definitions of predicates in *disjunctive normal form*. (an OR of ANDs).
- For example, we might say that X is a relative of Y if either (i) X is an ancestor of Y OR (ii) Y is an ancestor of X OR (iii) X and Y share a common ancestor.
- We can also do similar disjunctions using constants: X is a parent of Y if (X is alice and Y is bob) OR (X is alice and Y is bill) OR (X is bob and Y is charles) OR ...

Prolog Syntax

- Predicates and constants begin with a lowercase letter.
- Variables begin with an uppercase letter.
- Express basic facts (no consequent) with `predicate(param, param)`.
 - The period is important.
- Express more complex predicates, including preconditions, using `predicate0(param, param)`
`:- predicate1(param), predicate2(param)`.
 - Don't forget the period.
- For example

```
related(X,Y) :- ancestor(X,Y).
related(X,Y) :- ancestor(Y,X).
related(X,Y) :- ancestor(Z,Y), ancestor(Z,X).
parent(alice,bob).
parent(alice,bill).
parent(bob,charles).
```

- You must store your definitions in a file, and load it with the incredibly intuitive [*'filename'*].
- You ask a question by typing a predicate that involves constants and variables.
- You accept an answer by hitting return.
- You reject an answer by hitting semicolon.

Computation in Prolog

- So, how does Prolog use these techniques to compute?
- It turns out to be fairly straightforward, at least at first glance.
- At any time, the interpreter has
 - A database of predicate definitions [immutable]
 - A mapping of variables to values
 - A list of undefined variables
 - A set of goals left to solve.
- The basic strategy is
 - Grab the next goal left to solve.
 - For each predicate definition that matches the goal
 - Try to unify the goal and the predicate (which updates the map).
 - If they unify, put preconditions at the front of the queue and
 - If they don't unify, return failure.

Lab

- Do the lab.
- Be prepared to reflect.