A Prolog Tutorial

Based on Clocksin and Mellish Chapter 1

What is Prolog? - 1

◊ Prolog is an example of a logic programming language
◊ Invented by Alain Colmeraurer in 1972
◊ The version implemented at the University of Edinburgh has become the standard
◊ Called Edinburgh Prolog
◊ We will use SWI Prolog

What is Prolog? - 2

◊ Like Lisp, Prolog is denotational
◊ Denotational programming:
  » Has a mathematical meaning
  » Describes what to compute not how to compute
◊ Unlike Lisp, Prolog is not a functional language, instead it is declarative or relational
◊ Switch from values of a functional expression to truth or falsity of relational expression
  » Functional expression: √2
  » Relational expression: x ≠ 0

Prolog Programming

◊ Computer programming in Prolog consists of:
  » Declaring some facts about objects and their relationships
  » Determining some rules about objects and their relationships
  » Asking questions about objects and their relationships
Facts in Prolog allow you to express arbitrary relationships between objects.

In English:
» Fido is a dog
» Fido eats dogfood

In Prolog
» dog(fido).
» eats(fido, dogfood).

Fact examples
» dog(fido).
» eats(fido, dogfood).

What you should notice:
» The name of all relationships and objects must begin with a lower-case letter.
  > From the examples above this includes dog, fido, eats, dogfood
» The relationship is written first, and the objects are written separated by commas, and the objects are enclosed by a pair of round brackets.
» The full stop character “.” must come at the end of a fact.

Fact examples
» dog(fido).
» eats(fido, dogfood).

The name of the relationship is called the predicate or functor
Facts can have an arbitrary number of arguments (the number of arguments is called the arity)
In Prolog a collection of facts (and rules which we will look at soon) is called a database

Getting into and out of the SWI Prolog Interpreter:
On Prism to run Prolog enter
% pl
The following prompt appears:
? -
To end the Prolog session
? – CTRL-d
  or
? – halt.
Prolog files end in .pro
Instructions for entering facts into Prolog:

```
?- consult(user).
... type in facts ...
?- CTRL-d OR  
?- [user].
... type in facts ...
?- CTRL-d
```

The above method is inconvenient and error prone; a better method is to put your database in a file and consult that file:

```
?- consult('ring.pro').
```

Or you can consult a whole list of files by putting them in a list:

```
?- ['ring.pro', 'utilities.pro', 'other.pro'].
```

If you have entered databases and then realize you have made a mistake, you can correct your mistake and type make. Any databases that have been changed will be consulted again.

For help:

```
?- help(topic).

or

?- apropos(word).
```

/* multi-line comments can be enclosed in these */

% this is for comments that just go to the end of the line

A query in Prolog is a predicate.

Can also have Boolean combinations of predicates (we will see an example later)

Most common is to have a single predicate.

When a question is asked, Prolog searches through the database. It looks for facts that match the query. Two facts match if their predicates are the same and their corresponding arguments each are the same.

When you query the database Prolog searches the database from top to bottom.

If Prolog finds a fact that match it will answer yes.

Otherwise Prolog will answer no.

No does not mean false, it means not provable.
An example database

Example assume the following predicates are put in a file, ‘catdog.pro’:

- dog(fido).
- dog(prince).
- cat(princess).
- cat(fluffy).
- eats(fido, dogfood).
- eats(prince, dogfood).
- eats(fluffy, catfood).
- chases(fido, tail).
- chases(fluffy, mice).
- chases(prince, fluffy).

Queries - Examples

```prolog
%pl
?- ['catdog.pro']

? - cat(fluffy).
yes.

? - chases(fido, mice).
no.

? - eats(prince, dogfood).
yes.

? - eats(fluffy, dogfood).
no.
```

Variables

- Variables are used to stand for objects to be determined by Prolog.
- Variables can be instantiated or not instantiated.
- A variable is instantiated when there is an object the variable stands for.
- A variable is not instantiated when what the variable stands for is not yet known.
- Any name beginning with a capital letter or an underline character, “_”, is a variable.

Search with Variables - 1

- We load our database and ask Prolog:
  ?- eats(X, dogfood).

- How does Prolog work?
  - When we ask the question variable X is uninstantiated
  - Prolog searches the database looking for a fact that matches the question. Prolog searches for a fact whose functor is eats and whose second argument is dogfood.
  - Prolog searches from top to bottom
  - The fact eats(fido, dogfood) is found first
  - X is instantiated to fido
  - Prolog marks the place where the match is found
  - Once Prolog finds a fact that matches it prints out the solution: X = fido
The story is not over however ...

- Now Prolog waits for instructions
- If you hit return that means you are satisfied with one answer and Prolog stops searching
- If you type ';' it means search for another answer
- Prolog searches starting from where it left the place marker
- Variable X is uninstantiated
- We say Prolog is attempting to resatisfy the question
- In this case another solution is found and Prolog returns: X = prince
- If we type in ';' again there are no other matches so Prolog returns no

Conjunctions

- Often more complex ideas can be expressed using conjunctions
- For example, what do cats eat?
  - Can ask this by asking: if X is a cat and X eats Y what is Y?
- Use comma to represent a conjunction
- All goals in the conjunction must be satisfied before the conjunction is satisfied.
- In this case we want to ask:
  ?- cat(X), eats(X,Y).

Search - Query with a conjunction - 1

- Must remember that each goal keeps its own place marker
- If a goal cannot be satisfied Prolog attempts to satisfy the previous goal
- This is called backtracking

Variables - Examples

?- cat(X).
X = princess
?- dog(Dogname).
Dogname = fido ;
Dogname = prince ;
no.
?- chases(fido, X).
X = tail ;
no.
?- chases(X, Y).
X = fido  Y = tail ;
X = fluffy  Y = mice ;
X = prince  Y = fluffy ;
no.
We ask:
?- cat(X), eats(X, Y).

How does Prolog work?

» Database is searched to match cat(X). The goal matches cat(princess). X is instantiated to princess everywhere in the query that X appears. Prolog marks the place in the data base where the goal matched.

» Now data base is searched for eats(princess, Y). No such fact exists so the goal fails. We backtrack to the first goal, X becomes uninstantiated and we attempt to resatisfy cat(X) starting from the place marker. This time X matches fluffy.

We aren’t finished yet …

» Now we want to satisfy eats(fluffy, Y). Since we are not resatisfying this goal we start from the beginning of the data base.

» The goal matches eats(fluffy, catfood). Both goals are now satisfied and Prolog will return
  
  \[ X = \text{fluffy} \quad Y = \text{catfood} \]

» The place holder for the first goal is at cat(fluffy). The place holder for the second goal is at eats(fluffy, catfood).

» What if we type “;”? ? ?

A rule is a general statement about objects and their relationships

A rule is an if … then relationship

Rules have the general structure
  
  head :- body

Only one fact can be in the head – the consequence

The body is a Boolean combination of predicates

Rules are written backward to:

  » Emphasize the backward chaining for data base search

  » Be more regular in structure since the head is only 1 predicate.

Some examples:

» X is a dog if X has fur and barks

» Use an umbrella if it is raining and you own an umbrella

In Prolog:

  dog (X) :- has(X, fur),
  barks(X).

  use(umbrella) :- weather(raining),
  own(umbrella).
father(X, Y) :- male(Y), parentof(X, Y).

What you should notice:

» Rules end in a period.
» Whenever a variable X in a rule becomes instantiated to some object, all X's are instantiated within the scope of X. The scope of a rule is the whole rule.

A predicate can be defined by a mixture of facts and rules

For example (from Clocksin and Mellish pg. 18)

/* 1 */ thief(john).
/* 2 */ likes(mary, food).
/* 3 */ likes(mary, wine).
/* 4 */ likes(john, X) :- likes(X, wine).
/* 5 */ may_steal(X, Y) :- thief(X), likes(X, Y).

Note the clauses for the predicate likes consists of 2 facts and a rule

Question: what may John steal? Translates to
?- may_steal(john, X).

How does Prolog answer this question???

» Prolog searches the database for a may_steal clause and finds clause number 5
» X in the rule is instantiated to john from the question
» X in the question and Y in the rule are shared
» Now Prolog attempts to satisfy the 2 goals in the body, one by one
» The first goal is thief(john). This goal matches clause 1 and succeeds

The second goal is likes(john, Y). This matches the head of the rule given in clause 4. The Y in the goal shares with the X in the head and both are still un-instantiated.

Now Prolog must satisfy the goal in the body, likes(X, wine).

This goal matches likes(mary, wine) in clause 3 and X is instantiated to mary

Since the goal in clause 4 succeeds the whole rule succeeds and likes(john, mary) is established.

Now the goals in clause 5 have succeeded with X instantiated to mary.

So the final solution is that John may steal Mary.
Disjunction

◊ Disjunction is indicated by a ;
◊ P :- Q; R. means P is true if Q is true or R is true
◊ This is equivalent to
  P :- Q.
  P :- R.
◊ The comma binds stronger than the semicolon. So the clause:
  P :- Q, R ; S, T, U.
  Is understood as
  P :- (Q, R) ; (S, T, U).
  And means the same as
  P :- Q, R.
  P :- S, T, U.

Prolog Programs

◊ There are several Prolog programs in the course directory on Prism
◊ Files:
  » ring.pro
  » tower.pro
  » utilities.pro (you are not ready for this yet but we will look at it next week!)
◊ Try consulting ring.pro or tower.pro and playing!