Basics of PROLOG Programming

Gouravmoy Banerjee

Department of Computer Science, A.C. College, Jalpaiguri, India-735101

Introduction to PROLOG

Prolog or **PROgramming in LOGics** is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve symbolic or non-numeric computation. This is the main reason to use Prolog as the programming language in Artificial Intelligence, where symbol manipulation and inference manipulation are the fundamental tasks.

Elements of PROLOG

Prolog language basically has three different elements –

- Facts – The fact is predicate that is true, for example, if we say, "Tom is the son of Jack", then this is a fact.
- Rules – Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met.

For example, if we define a rule as –
grandfather(X, Y) :- father(X, Z), parent(Z, Y)
This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.
Questions – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

Application areas of Prolog

- Specification Language
- Expert Systems
- Medical Diagnostics
- Problem Solving
- Intelligent Database retrieval

Syntax of Facts

relation(object1,object2...).

Example:

- elephant(shiva).
- cooks(debangshu,soup).
- goodboy(nittyagopal)

Syntax of Rules

rulename(object1,object2,...):-fact/rule(object1,object2, ...)

Example:

- Suppose a clause is like :
  P :- Q;R.
  This can also be written as
  P :- Q.
  P :- R.
- If one clause is like :
  P :- Q,R;S,T,U.
  Is understood as
  P :- (Q,R);(S,T,U).
  Or can also be written as:
  P :- Q,R.
  P :- S,T,U.

Syntax of Rules (Contd...

Examples:

- angry(amitkaku) :- dirty(classroom).
- good(movie):- buy(tickets).
- goTolunch(riya) :- hungry(riya), free(riya).

DEMO

To be shown in practical.

Operators in PROLOG

Relational Operators

Operator	Meaning
X Y	X is greater than Y
X Y	X is less than Y
X = Y	X is greater than or equal to Y
X = Y	X is less than or equal to Y
X =:= Y	the X and Y values are equal
X =\= Y	the X and Y values are not equal

Operators in PROLOG (Contd...)

Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
//	Integer Division
mod	Modulus

Operators in PROLOG (Contd...)

Logical Operators

Operator	Meaning
not	logical NOT
,	logical AND
;	logical OR

Looping in PROLOG

Example:

loop(0).

loop(N) :- N>0, write('value of N is: '), write(N), nl.

S is N-1, loop(S).

Output:

?- loop(4).

value of N is: 4

value of N is: 3

value of N is: 2

value of N is: 1

yes

Demo programs

Simple programs to be shown.

Lists

Representation of Lists

The list is a simple data structure that is widely used in non-numeric programming.

List consists of any number of items, for example, red, green, blue, white, dark.

It will be represented as, [red, green, blue, white, dark].

The list of elements will be enclosed with square brackets.

A list can be either empty or non-empty. In the first case, the list is simply written as a Prolog atom, [ ]. In the second case, the list consists of two things as given below −

- The first item, called the head of the list;
- The remaining part of the list, called the tail.

Lists (Contd..)

Suppose we have a list like: [red, green, blue, white, dark]. Here the head is red and tail is [green, blue, white, dark]. So the tail is another list.

Now, let us consider we have a list, L = [a, b, c]. If we write Tail = [b, c] then we can also write the list L as L = [a | Tail]. Here the vertical bar (|) separates the head and tail parts.

So the following list representations are also valid –

[a, b, c] = [x | [b, c]]

[a, b, c] = [a, b | [c]]

[a, b, c] = [a, b, c | [ ]]

List operators

Membership Operation

During this operation, we can check whether a member X is present in list L or not? So how to check this? Well, we have to define one predicate to do so. Suppose the predicate name is listmember(X,L). The goal of this predicate is to check whether X is present in L or not.

To design this predicate, we can follow these observations. X is a member of L if either –

X is head of L, or

X is a member of the tail of L

Examples

```
listmember(X,[X|_]).
listmember(X,[_|TAIL]) :- listmember(X,TAIL).
```

List operators (Contd...)

Length Calculation

Suppose the predicate name is list_length(L,N). This takes L and N as input argument. This will count the elements in a list L and instantiate N to their number. As was the case with our previous relations involving lists, it is useful to consider two cases −
If list is empty, then length is 0.
If the list is not empty, then L = [Head|Tail], then its length is 1 + length of Tail.

Examples

```
list_length([ ],0).
list_length([_|TAIL],N) :- list_length(TAIL,N1), N is N1 + 1.
```

More operators

To be shown through code.

Thank You.