Introduction to Prolog Language

Prolog: is a computer programming language that is used for solving problems involves objects and relationships between objects.

Some of prolog language characteristics:

1) We can solve a particular problem using prolog in less no of line of code.
2) It's an important tool to develop AI application and ES.
3) Prolog program consist of fact and rule to solve the problem and the output is all possible answer to the problem.
4) Prolog language is a descriptive language use the inference depend on fact and rule we submit to get all possible answer while in other language the programmer must tell the computer on how to reach the solution by gives the instruction step by step.

Prolog language component

1- Facts

Is the mechanism for representing knowledge in the program; syntax of fact:

1. The name of all relationship and objects must begin with a lower-case letter, for example: likes (john, marry).
2. The relationship is written first, and the objects are written separated by commas, and enclosed by a pair of round brackets. Like (john, marry)
3. The full stop character '.' Must come at the end of fact.

2-Rules

Rules are used when you want to say that a fact depends on a group of other facts, and we use the following syntax:

1. One fact represents the head (conclusion).
2. The word if used after the head and represented as ":-'.
3. One or more fact represents the requirement (condition).

The syntax of if statement

If (condition) then (conclusion)

[conclusion: - condition] rule

3-Questions

Question used to ask about facts and rules. Question look like the fact and written under the goal program section while fact and rule written under clauses section.

4-Variables

If we want to get more interest information about fact or rule, we can use variable to get more than Yes/No answer.

*variables dose not name a particular object but stand for object that we cannot name.

*variable name must begin with capital letter.

*using variable we can get all possible answer about a particular fact or rule.

*variable can be either bound or not bound.

Variable is bound when there is an object that the variable stands for.

The variable is not bound when what the variable stands for is not yet known.

Example:

Fact

like (john, mary).

like (john, flower).

like (ali, mary).

Question

1. like (john,X)

   X= mary

   X = flower

2. like(X, mary)

   X=john

3. like(X, Y)

   X=john Y=flower

   X=john Y=mary

   X=ali Y=mary

4. Type of questing in the goal

There are three type of question in the goal summarized as follow:

1. Asking with constant: prolog matching and return Yes/No answer.

2. Asking with constant and variable: prolog matching and produce result for the Variable.

3. Asking with variable: prolog produce result.

Example:

age(a,10).

age(b,20).

age(c,30).

Goal:

1. age(a,X).	Ans:X=10		Type2
2. age(X,20).	Ans:X=b		Type2
3. age(X,Y).	Ans: X=a Y=10, X=b Y=20, X=c Y=30.		Type3
4. age(_,X).	Ans:X=10 , X=20, X=30.	'_' means don't care	Type3
5. age(_,_).	Ans: Yes		Type1

Conjunctions and backtracking

Conjunctions

1. and ‘,’.

2. or ‘;’.

Used to combines facts in the rule or to combine fact in the goal to answer questions about more complicated relationship.

Example:

Facts

like (marry, food).

like(marry, wine).

like(john, marry).

Goal

like(marry, john) , like(john, marry).

We can ask dose marry like john and dose john like marry?

Now, how would prolog answer this complicated question?

Prolog answers the question by attempting to satisfy the first the first goal. if the first goal is in the database, then prolog will mark the place in the database, and attempt to satisfy the second goal.

If the second goal is satisfied, then prolog marks that goal‘s place in the database, and we have a solution that satisfy both goals.

♦ It is important to remember that each goal keeps its own place marker.

If, however, the second goals are not satisfied, then prolog will attempt to re-satisfy the previous goal.

Prolog searches the database in case it has to re-satisfy the goal at a later time. But when a goal needs to be re-satisfied, prolog will begin the search the search database completely for each goal. If a fact in the database happens to match, satisfying the goal, then prolog will mark the place in the database in case it has to re-satisfy the goal at the later time. But when a goal needs to be re-satisfied, prolog will begin the search from the goal’s own place marker, rather than from the start of database and this behavior called “backtracking”.

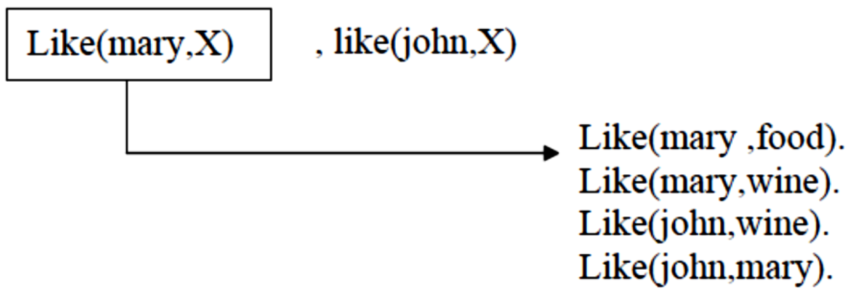
Example: about backtracking

***Facts**

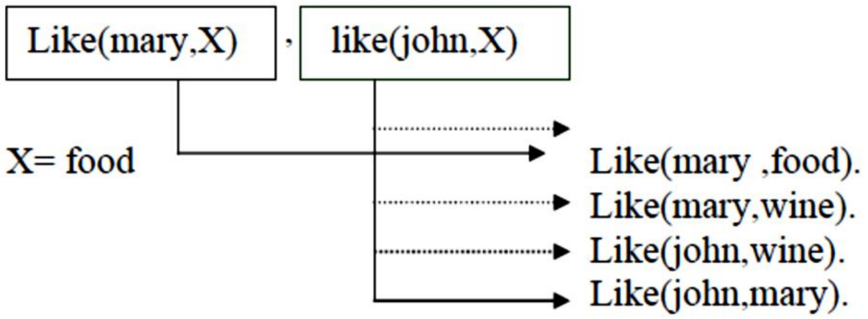
```
like(marry,food).
like(marry,wine).
like(john,wine).
like(john,marry).
```

***Goal:**

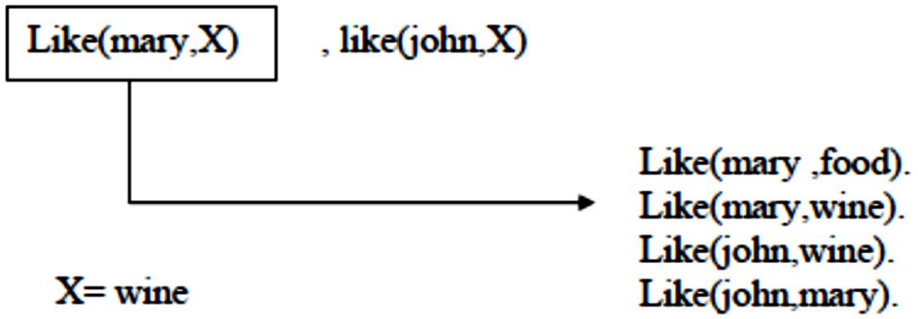
```
like(marry,X) , like(john,X).
```



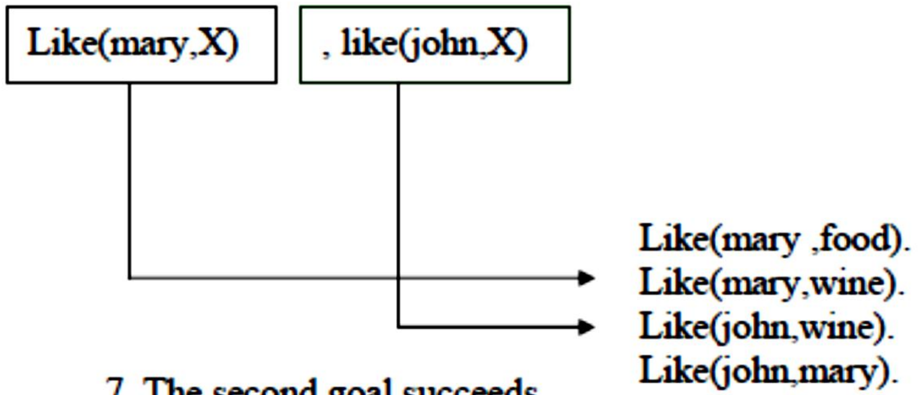
1. The first goal succeed, bound X to food.
2. Next, attempt to satisfy the second goal.



3. The second goal fails.
4. Next, backtrack: forget previous value of X and attempt to resatisfy the first goal.



5. The first goal succed agin, bund X to wine.
6. Next, attempt to satisfy the second goal.



7. The second goal succeeds.
8. Prolog notifies you of success.

Data Type

Prolog supports the following data type to define program entries.

1. **Integer**: to define numerical value like 1, 20, 0,-3,-50, etc.
2. **Real**: to define the decimal value like 2.4, 3.0, 5,-2.67, etc.
3. **Char**: to define single character, the character can be of type small letter or capital letter or even of type integer under one condition it must be surrounded by single quota. For example, ‘a’,’C’,’123’.
4. **String**: to define a sequence of character like “good” i.e. define word or statement entries the string must be surrounded by double quota for example “computer”, “134”, “a”. The string can be of any length and type.
5. **Symbol**: another type of data type to define single character or sequence of character but it must begin with small letter and don’t surround with single quota or double quota.

Program structure

Prolog program structure consists of five segments, not all of them must appear in each program. The following segment must be included in each program predicates, clauses, and goal.

1. **Domains**: define global parameter used in the program.
```
Domains
    I= integer
    C= char
    S = string
    R = real
```
2. **Data base**: define internal data base generated by the program
```
Database
    greater (integer)
```
3. **Predicates**: define rule and fact used in the program.
```
Predicates
    mark(symbol, integer).
```
4. **Clauses**: define the body of the program. For the above predicates the clauses portion may contain mark (a, 20).

5.**Goal**: can be internal or external, internal goal written after clauses portion , external goal supported by the prolog compiler if the program syntax is correct. This portion contains the rule that drive the program execution.

Mathematical and logical operation

1. **Mathematical operation:**

operation	symbol
addition	+
subtraction	-
multiplication	*
Integer part of division	div
Remainder of division	mod

2. **logical operation:**

operation	symbol
greater	>
Less than	<
Equal	=
Not equal	<>
Greater or equal	>=
Less than or equal	<=

3. **Read and write functions**

Read function:

reading (Var) : read integer variable.

readchar (Var) : read character variable.

readreal (Var) : read (decimal) variable.

readln (Var) : read string.

Write function

write (Var) : write variable of any type.

Cut function (!)

Represented as “!” is a built in function always True, used to stop backtracking and can be placed any where in the rule, we list the cases where “!” can be inserted in the rule:

1. R:-f1, f2 , !. “f1, f2 will be deterministic to one solution.
2. R:-f1, ! , f2. “ f1 will be deterministic to one solution while f2 to all .
3. R:- !,f1,f2. “R will be deterministic to one solution.

Ex1 : program without use cut.

```
Domains
    I= integer
Predicates
   no( I )
Clauses
   no (5).
   no (7).
   no (10).
Goal
   no (X).

Output:
   X=5
   X=7
   X=10
```

Ex2 : program use cut.

```
Domains
    I= integer
Predicates
   no( I )
Clauses
   no (5):- !.
   no (7).
   no (10).
Goal
   no (X).

Output:
   X=5
```

Ex3 : program without use cut.

```
Domains
     I= integer
    S= symbol
Predicates
    a (I )
    b (s )
    c ( I, s )
Clauses
    a(10).
    a(20)
    b(a)
    b(c)
    c (X, Y):- a (X), b (Y).
Goal
    c (X, Y)

Output:
    X= 10 Y=a
    X=10 Y=c
    X=20 Y=a
    X=20 Y=c
```

Ex4: using cut in the end of the rule.

```
Domains
     I= integer
    S= symbol
Predicates
    a (I )
    b (s )
    c ( I, s )
Clauses
    a(10).
    a(20)
    b(a)
    b(c)
    c (X, Y):- a (X), b (Y), !.
Goal
    c (X, Y)

Output:
    X= 10 Y=a
```

Ex 5: using cut in the middle of the rule.

```
Domains
   I= integer
  S= symbol
Predicates
  a (I )
  b (s )
  c ( I, s )
Clauses
  a(10).
  a(20)
  b(a)
  b(c)
  c (X, Y):- a (X), !, b (Y).
Goal
  c (X, Y)
  X= 10 Y=a
        Y=b
```

List in Prolog

In prolog, a list is an object that contains an arbitrary number of other objects within it. Lists correspond roughly to array in other languages but unlike array, a list doesn't require you to how big it will be before use it.

1. **Syntax of List**

   List always defined in the domains section of the program as follow:

   Ex.

   Domains

   list = integer*

- '*' refer to list object which can be of length zero or undefined.
- The type of element list can be of any standard defined data type like integer, char … etc. or user defined data type explained later.
- List element surrounded with square brackets and separated by comma as follow:

  l = [1, 2, 3, 4].
- List consist of two parts head and tail, the head represent the first element in the list and the tail represent the remainder (i.e. head is an element but tail is a list) . for the following list :

  L = [1,2,3]

H = 1 T =[2,3]

H =2 T =[3]

H =3 T=[ ]

[ ] refer to empty list.

List can be written as [H|T] in the program, if the list is nonempty then this statement decompose the list into Head and tail otherwise (if the list is empty) this statement add element to the list.

2. List and Recursion

As maintained previous list consist of many elements, therefore to manipulate each element in the list we need recursive call to the list until it become empty.