Introduction to Prolog

Prolog Language

- Prolog is unique in its ability to infer facts from the given facts and rules.
- In Prolog, an order of clauses in the program and goals in the body of a clause is fixed.
- There are two major issues in logic programming which have been resolved in Prolog.
  - First issue is the arbitrary choice of goal in the resolvent to be reduced.
  - Second one is the non deterministic choice of the clause from the program to effect the reduction.

Contd…

- In Prolog, first issue is resolved by choosing the first sub goal from the resolvent and the second issue is resolved by choosing the first clause matched searching sequentially the clauses from top to bottom in a program.
- A stack scheduling policy is adopted. It maintains the resolvent as a stack.
- It pops the top sub goal for reduction, and pushes the derived goals on the resolvent stack.
- Prolog is an approximate realization of logic programming computation model on a sequential machine.

General syntax of Prolog

- A Prolog program is a collection of facts and rules. Brief syntax of Prolog is given below:
  - Constants are numerals and symbols such as, 4, mary, etc.
  - String is a string of characters and is enclosed within single quotes e.g., 'This is a string'.
  - Function and predicate names must start with alphabet and are formed by using lower case letters, numerals and underscore (_).

Contd..

- ***Variable*** names are formed similar to function and predicate names but it must start with upper case letter. Normally, we use X, Y, Z, ... for variables.
- Clause (representing fact or rule) in Prolog is terminated by full stop (.).
- Goal to a Prolog program is given after symbols ?
- Goal might be simple or conjunction of sub goal(s) terminated by full stop.

Arithmetic Assignment

- ***Arithmetic assignment*** is achieved by a predicate called **is** which is an infix operator.
- Evaluation of goal **X is Expression** is done by first evaluating
  - Expression according to the evaluation rules of arithmetic and is unified with left hand side variable X in order to succeed the goal.
  - Left hand side should be variable with known value.
  - All variables on right hand side of **is** must be known at the time of its execution otherwise the goal will fail.

Examples:

- ***X is 2 + 5*** succeeds by unifying X with 7.
- ***Y is X -3*** succeeds by unifying Y with 4.
  - Here X = 7 is available prior to executing this goal.
- ***7 is X*** fails as left side of **is** has to be variable.

Equality Operator

- *Equality operator* is denoted by '=' and it is also an infix operator.
- When the goal $\boldsymbol{T_1 = T_2}$ (pronounced as 'T$_1$ equals T$_2$') is attempted to be satisfied by Prolog, it succeeds if T_1 and T_2 match.
  - Here T_1 and T_2 are any terms**.**
  - It is to be noted that $T_1 = T_2$ or $T_2 = T_1$ are same.

Examples

- Goal **3 = X** succeeds by unifying a variable X with 3.
- Goal **mary = mery** , fails.
- Goal **X = likes(ram, sita)** succeeds by unifying X with likes(ram, sita).
- Goal **X = 4** succeeds by unifying a variable X with 4.
- Goal **love(X, sita) = love(ram, Y)** succeeds by unifying a variable **X** with **ram** and **Y** with **sita**.

Non Equality

- *Non equality* is denoted \= which is opposite of equality. The goal T_1 \= T_2 succeeds if the goal T_1 = T_2 fails otherwise the goal T_1 \= T_2 fails.
- It is pronounced as ‘T_1 is not equals to T_2’.
- **Examples:**
  - Goal **likes(ram, X) \= likes(sita, ram)** succeeds.
  - Goal **father(X, Y) \= father(john, mery)** fails.
  - Goal **4 \= 3** succeeds.
  - Goal **mary = mery** succeeds.

Prolog Program

- *Prolog program* is a finite sequence of program clauses.
- Informally, we can define Prolog program to be a set of facts and rules. It is based on Horn clauses.
- Representation of a **Horn clause** in Prolog:

$$A \;\text{:-}\; B_1, B_2, \ldots, B_n \,. \qquad (1)$$

  - The symbol $\leftarrow$ of logic programming is represented by :- in Prolog .
  - 'A' is positive atom and B_1, …and B_n are negative atoms.

Contd..

- Clause given in (1) is a *conditional rule* which is interpreted as "A is true if $B_1 \Lambda \ldots \Lambda B_n$ are simultaneously true".
- A clause with no antecedent is a *fact* that is always true and is denoted by **A.** .
- :- $B_1, \ldots, B_n$. means a conjunctive query interpreted as
  - "if $B_1, \ldots$ and B_n are true, then what is the consequent?".
- The goal in Prolog is represented as ?- $B_1, \ldots, B_n$.
  - Here B_i, $(1 \leq i \leq n)$ are called *sub goals*.

Contd..

- The symbols A and B_i , ($1 \leq i \leq n$) represent predicate names with arguments, if any.
  - The scope of variables is only a clause in which it is appearing.
- Comments in Prolog are preceded by percentage symbol '%' if single line and enclosed within /* comments */ for multiple lines.
- **Examples:**
  - Prolog rule for computing square of a number.
  - % square(X, Y) - succeeds if Y is a square of X.
    - Rule -- square(X, Y) :- Y is X * X.
    - Query: ?- square(2, Y).
    - Answer: Y = 4

Contd..

- The goals
  - ?- square(X, 9) and ?- square(3, 9) fail because left side of **is** has to be a variable and the variables appearing in the right side of 'is' should be bound.
- If we define above rule using equality operator (=) such as

  *square(X, Y) :- Y = X * X.*

  - Then the goal **?- square(X, 3 * 3)** succeeds with X = 3 whereas the goal **?- square(X, 9)** fails because equality is for comparing the terms and not for evaluating an expression on right side of equal operator.

Contd..

- Prolog rule to compare two terms for equality.

```
% equal(X, Y)- succeeds if terms X and Y are equal
      equal(X, Y)  :-        X = Y.
```

Goals:

```
?- equal(5, 5).       Answers:      Yes
?-   equal(5, X).     Answers:      X = 5
?- equal (5, 6).      Answers:      No
```

Prolog Control Strategy

- While executing Prolog goal, leftmost goal is chosen instead of an arbitrary one as is done in logic programming.
- Prolog is non deterministic because a goal can be satisfied using more than one rules.
- Non determinism is resolved by choosing a program clause using sequential search in a program from top to bottom for a unifiable clause.

Contd..

- Prolog contains **three basic control** strategies.
  - Forward movement
  - Unification (Matching)
  - Backtracking (Backward movement)

Forward Movement

- Choose a rule by searching sequentially in the program from top to bottom whose head matches with the goal with possible unifier.
- Remember the position of the matched rule.Join the rule body in front of the sequence of sub goals to be solved.
- Repeat until either goal is satisfied or is failed.

Unification

- Unification is a process of matching or finding the most general unifier (mgu).
- Constant matches with the same constant.
- A variable matches with any constant and becomes instantiated.
- A variable matches with another variable.

Examples of Unification

- *love(X, sita)* and *love(X, Y)* unify with **love(ram, sita)** with
  - **mgu** as {X / ram} and {X / ram, Y / sita} respectively.
- *love(ram, sita)* does not unify with
  - love(sita, ram), love(X, mary), love(john, X) etc.

Backtracking

- Backtracking in Prolog takes place in two situations viz., shallow and deep.
- First situation(Shallow) is when a sub goal fails and another rule if exist is tried.
- Second situation (Deep) is when if the failed sub goal does not have any rule, then it backtracks to previous sub goal and repeat the process.

Shallow Backtracking

- It occurs when some sub goal fails.
- Another rule is tried to be unified (if any) from the last place marker.
- If we are able to satisfy all sub goals then solution is displaced.
- For alternative solution, shallow and deep backtracking are applied.

Deep Backtracking

- It occurs as a result of their failure as shallow.
- In this case it backtracks to previous sub goal.
- Remove bindings generated by the sub goals introduced due to the current sub goal.
- Look for an alternative rule (with the head matching goal, if any) after the place marker for previous sub goals.

Example

- Let us consider the following query to explain both types of backtracking more clearly.

  ?- $G_1, \ldots, G_{i-1}, G_i, \ldots, G_n$

- Suppose at some point we have satisfied $G_1, \ldots, G_{i-1}$, with some common bindings and now trying to satisfy sub goal G_i.
- Choose a rule by searching sequentially from top to bottom whose head matches with sub goal G_i with some bindings.

Contd…

- If the sub goal G_i is satisfied, then proceed further to satisfy G_{i+1} and continue the process in forward direction till we reach G_n
- If we fail G_i after trying all the rules, *backtrack* to G_{i-1}.
  - Remove the bindings created earlier due to G_{i-1} and try to satisfy G_{i-1} again by using an alternative rule, if it exists, from last place marker of G_{i-1}.
  - If sub goal Gi-1 is satisfied then move forward to Gi, else backtrack to Gi-2.
  - The process of backtracking continues till we reach G1.
  - If G_1 fails for all possible definitions, then entire query fails.

Contd...

- In case, if entire conjunction of sub goals succeed, then the solution is displayed and in order to find alternative solution, the sub goal G_n is tried to be satisfied with alternative definition, if it exist, after last place marker (Shallow backtracking) of G_n .
- Otherwise it *backtracks* to previous sub goal G_{n-1} (Deep backtracking).
- Repeat the process till all possible solutions are generated.

Execution of a Prolog Query

- A computation of a ***ground query*** G (with no variables) generates a proof tree. If the query succeeds, then answer is ‘yes’ otherwise it is ‘no’.
- A computation of a ***non ground query*** G (with variables) generates ***search tree*** and finds of all possible solutions of G with respect to a program P.
- As a result of computation, all possible values of the variables involved in query G are generated and displayed.

Conventions used

- Start generating tree with root as a goal G to be satisfied.
- Down ward arrow indicates the reduction of goal.
- Clause number on the left side of an arrow.
- Possible bindings enclosed within curley brackets (if necessary) on right of an arrow.
- Leaf of the tree is labeled either succeeds or fails.

Simple Prolog Program

- Consider the following Prolog program consisting of one rule for defining *grandfather* and five facts about *father* relations.

```
grandfather(X, Y) :- father(X, Z), parent(Z, Y).   (1)
parent(X, Y)    :-  father(X, Y).                  (2)
parent(X, Y)    :-  mother(X, Y).                  (3)
father(james, robert).                             (4)
father(mike, william).                             (5)
father(william, james).                            (6)
father(robert, hency).                             (7)
father(robert, cris).                              (8)
```

Ground Query Proof Tree

?- grandfather(james, hency).

Proof tree:

$$\begin{tikzcd}
\text{?- grandfather(james, hency).} \arrow[d, "(1)"'] \\
\text{?- father(james, Z), parent(Z, hency).} \arrow[d, "(4)"', "\{ Z = \text{robert} \}"] \\
\text{?- parent(robert, hency).} \arrow[d, "(2)"'] \\
\text{?- father(robert, hency).} \arrow[d, "(7)"'] \\
\Box \\
\text{succeeds}
\end{tikzcd}$$

Answer: Yes

Non Ground Query Search tree

?- grandfather(william, X).

Search tree:

$$\begin{tikzcd}
 & \text{?- grandfather(william, X).} \arrow[d, "(1)"'] & \\
 & \text{?- father(william, Z), parent(Z, X).} \arrow[d, "(6)"', "\{Z = james\}"] & \\
 & \text{?- parent(james, X).} \arrow[dl, "(2)"'] \arrow[dr, "(3)"] & \\
\text{?- father(james, X).} \arrow[d, "(4)"', "\{X = robert\}"] & & \text{?- mother(james, X).} \arrow[d] \\
\square \quad \text{succeeds} & & \text{fails}
\end{tikzcd}$$

Answer: X = robert

Operators

- The following relational and arithmetic operators are used in formulation of goals. The values of X and Y are compared and the goals succeed or fail as described below:

Relational Operators: All relational operators are infix operators.

- X < Y succeeds if X is less than Y.
- X > Y succeeds if X is greater than Y.
- X =< Y succeeds if X is less than or equal to Y.
- X >= Y succeeds if X is greater than or equal to Y.
- Equality and non equality operators .

Contd..

Arithmetic Operators:

- X + Y , X – Y, X * Y, X / Y, X mod Y with usual meanings in arithmetic.

Examples:

- Define a rule to test whether a given number is positive or not.

/* positive_number(X)- succeeds if X is greater than zero otherwise it fails. */

```
        positive_number(X)    :-         X > 0.
Goals:?- positive_number(5).             Answer:  Yes
        ?- positive_number(-4).          Answer:   No
```

Prolog Program

- Rule for identifying whether a number lies in a given range or not. Here X is number and L, U are lower and upper range

```
        number_in_range(L, X, U) :- X >= L, X =< U.  (1)
Or      number_in_range(X, L, U) :- X >= L, X =< U.  (2)
Or      number_in_range(L, U, X) :- X >= L, X =< U.  (3)
```

Goals: Queries to above three versions are different.

```
(1)     ?- number_in_range(10, 23, 100).     Answer:  Yes
        ?- number_in_range(10, 23, 20).      Answer:  No
(2)     ?- number_in_range(23, 10, 100).     Answer:  Yes
        ?- number_in_range(23, 10, 20).      Answer:  No
(2)     ?- number_in_range(10, 100, 23).     Answer:  Yes
        ?- number_in_range(10, 20, 23).      Answer:  No
```