SWI-Prolog

CS5201/CS5205: Advanced Artificial Intelligence & Lab

What is Prolog?

- Prolog is a logic programming language that is used in artificial intelligence.

- It is a declarative programming language expressing logic as relations.

- A Prolog program consists of a collection of facts and rules; a query is a theorem to be proved.

Basic Definitions

Facts: These are statements that are true. They are written as a predicate followed by a dot.

For example, `likes(john, mango).` is a fact that states that John likes mango.

Rules: These are statements that define relationships between facts. They are written as a predicate followed by a body, which is a list of one or more predicates separated by commas and enclosed in parentheses, followed by a period.

For example, 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 the father of Z.

Queries: These are questions that we ask Prolog. They are written as a predicate followed by a question mark.

For example, `?- likes(john, mango).` is a query that asks if John likes mango.

Symbols in Prolog

English	Predicate Calculus	Prolog
If	-->	:-
Not	~	Not
Or	V	;
and	^	,

Guidelines to write facts

We can define fact as an explicit relationship between objects, and properties these objects might have. So facts are unconditionally true in nature.

- Names of properties/relationships begin with lower case letters.
- The relationship name appears as the first term.
- Objects appear as comma-separated arguments within parentheses.
- A period/dot "." must end a fact.
- Objects also begin with lower case letters. They also can begin with digits (like 1234), and can be strings of characters enclosed in quotes e.g. color(penink, ‘red’).
- phoneno(agnibha, 1122334455). is also called a predicate or clause.

Examples

- Tom is a cat -> *cat(tom).*
- Kunal loves to eat Pasta -> *loves_to_eat(kunal,pasta).*
- Hair is black -> *of_color(hair,black).*
- Nawaz loves to play games -> *loves_to_play_games(nawaz).*
- Pratyusha is lazy. -> *lazy(pratyusha).*

Guidelines to write Rules

An implicit relationship between objects. So facts are conditionally true. So when one associated condition is true, then the predicate is also true.

- **Lili is happy if she dances.**
- **Tom is hungry if he is searching for food.**
- **Jack and Bili are friends if both of them love to play cricket.**
- **will go to play if school is closed, and he is free.**

These are some rules that are conditionally true, so when the right hand side is true, then the left hand side is also true.

Guidelines

- Symbol (:-) will be used as “If”, or “is implied by”.
- The LHS of this symbol is called the Head, and right hand side is called Body.
- Symbol comma (,) is used as conjunction,
- Symbol semicolon(;) , is known as disjunction.

Demonstration

- **Lili is happy if she dances =** *happy(lili) :- dances(lili).*
- **Tom is hungry if he is searching for food.** *hungry(tom) :- search_for_food(tom).*
- **Jack and Bili are friends if both of them love to play cricket.**

friends(jack, bili) :- lovesCricket(jack), lovesCricket(bili).

- **Ryan will go to play if school is closed, and he is free.**

goToPlay(ryan) :- isClosed(school), free(ryan).

Guidelines to write Queries

Queries are some questions on the relationships between objects and object properties. So question can be anything, as given below

- Is tom a cat? = *?- cat(tom).*
- Does Kunal love to eat pasta? = *?- loves_to_eat(kunal, pasta).*
- Is Lili happy? = *?- happy(lili).*
- Will Ryan go to play? = *?- will_play(ryan).*

How and Where?

- **sudo apt-add-repository ppa:swi-prolog/stable**
- **sudo apt-get update**
- **sudo apt-get install swi-prolog**

After steps

1) Open a Text Editor
2) Create a New File with a .pl Extension. Example : *Program1.pl*
3) Write Facts and Rules in the File
4) Open the Terminal
5) Load the Prolog File
6) Start Querying

Program 1

Step 1: Create a Prolog File

Step 2: Write the following Prolog code in program1.pl: (Facts and Rules)

likes(john, pizza).

likes(mary, pasta).

loves(X, Y) :- likes(X, Y).

Contd.

Step 3: Load and Run the Prolog Program

Open Terminal and write below as instructed

swipl

?- [program1].

Step 4: Start Querying

?- loves(john, pizza).

?- loves(mary, pizza).

Program 2: Defining Family Relationships (Facts & Rules)

Facts: Defining parent-child relationships

parent(john, mary).

parent(john, tom).

parent(susan, mary).

parent(susan, tom).

parent(mary, alice).

parent(mary, bob).

Rules: Defining relationships

father(X, Y) :- parent(X, Y), male(X).

mother(X, Y) :- parent(X, Y), female(X).

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

Gender facts

male(john).

male(tom).

male(bob).

female(susan).

female(mary).

female(alice).

Running the Program

?- [family].

?- father(john, mary).

?- sibling(mary, tom).

?- sibling(mary, bob).

Program 3: Factorial (Recursion)

Write the below program in a new file **factorial.pl**

```
factorial(0, 1).  % Base case: factorial of 0 is 1
factorial(N, F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, F1),
    F is N * F1.
```

Running the program

?- factorial(5, X).

?- factorial(0, X).

Lists

A **list** in Prolog is a sequence of elements enclosed in square brackets:

A list has two parts:

- **Head (H)**: The first element of the list.
- **Tail (T)**: The remaining elements.

Normal list: [apple, banana, orange]

H = apple

T = [banana, orange]

Example: Checking if an Element is in a List

is_member(X, [X|_]). *X is found at the head of the list*

is_member(X, [_|T]) :- is_member(X, T). *Recurse through the tail*

?- is_member(3, [1, 2, 3, 4]).

Output: **True.**

?- is_member(5, [1, 2, 3, 4]).

Output: **False.**

Example 2: Finding the Length of a List

Code for finding length:

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

Query:

?- **list_length([a, b, c, d], X).**

Output: ***X = 4.***

Example 3: Reversing a List

```
reverse_list([], []).

reverse_list([H|T], Rev) :-
    reverse_list(T, RevT),
    append(RevT, [H], Rev).
```

Query:

```
?- reverse_list([1,2,3,4], X).

X = [4,3,2,1].
```

Cut Operator

The Cut Operator (!) in Prolog

The **cut (!)** operator **prunes** unnecessary backtracking.

- When Prolog encounters **!**, it commits to the current rule and **prevents backtracking**.
- Any alternative rules **after the cut** are **ignored**.

Example

Without Cut:

```
max(X, Y, X) :- X >= Y.

max(X, Y, Y) :- X < Y.
```

?- *max(5, 3, R).*
R = 5.

?- *max(3, 5, R).*
R = 5.

Since there's no cut, Prolog still tries the second rule even when the first one succeeds.

Example

With Cut:

max(X, Y, X) :- X >= Y,**!**.

max(X, Y, Y) :- X<Y

?- max(5, 3, R).
R = 5. *The second rule is never checked*

?- max(3, 5, R).
R = 5.

First rule fails, so we move to second rule.

Example: Using Cut in Decision-Making

```
grade(Marks, 'A') :- Marks >= 90, !.

grade(Marks, 'B') :- Marks >= 75, !.

grade(Marks, 'C') :- Marks >= 60, !.

grade(_, 'F').  % Default case
```

?- grade(85, X).
X = 'B'.

If the cut (!) was not used, Prolog would still check all rules even after finding a match.

Debugging Tools in Prolog

SWI-Prolog provides the following debugging features:

1. **`trace.`** – Enables step-by-step execution.
2. **`spy(Predicate).`** – Sets a breakpoint (spy point) on a predicate.
3. **`nospy(Predicate).`** – Removes the breakpoint.
4. **`debug.`** – Enables debugging mode.
5. **`nodebug.`** – Disables debugging mode.

Faulty Factorial Programme

```
factorial(0, 1).

factorial(N, F) :-
        N1 is N - 1,
        factorial(N1, F1),
        F is N * F1.
```

Debugging by spy(Predicate).

If we query: ***?- factorial(-1, R).***

The program enters an **infinite recursion**.

Fixing It with Debugging

1. **Set a breakpoint on factorial:**
   ?- spy(factorial).
2. Run the faulty query:
   a. *?- factorial(-1, R)*

```
[debug]  ?- factorial(-1,R).
Call: (12) factorial(-1, _24538) ?
creep
   Call: (13) _25764 is -1+ -1 ? creep
   Exit: (13) -2 is -1+ -1 ? creep
   Call: (13) factorial(-2, _27386) ?
creep
   Call: (14) _28204 is -2+ -1 ?
```

Fixed code

```
factorial(0, 1).  % Base case for factorial(0)
factorial(N, _) :- N < 0, !, fail.  % Prevent negative recursion
factorial(N, F) :-
        N1 is N - 1,
        factorial(N1, F1),
        F is N * F1.
```

?- factorial(-1,R).
false.

Practice problems

- Complete permutation problem that was kept as an exercise.
- Implement 8-queens problem in prolog.
- Implement del_duplicates without the ‘!’ and see the issues.
- Define relation ‘endTerm(Tail, List)’ so that ‘Tail’ is the last element of list List.
- Define relation ‘revList(List, RevList)’ that reverses the list. For example, revList([a,b,c], [c,b,a]).
- Define ‘palindrome(List)’ if the list is same from forward and backward direction. palindrome([m, a, d, a, m])

Thank you for your attention