Lab 6

Manan Shah
2019130059
TE Comps

November 23, 2021

Aim

To solve problems using the Prolog Programming Language.

Theory

Prolog

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations In prolog, We declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (,). Their relation is expressed at the start and outside the parenthesis. Every fact/rule ends with a dot (.).

Problem Statement

1. Create a family tree using PROLOG. It should have rules for father, mother, brother, sister, grandparent, uncle, aunt, predecessors, successors.
2. Given a list
   [*a, a, a, a, b, b, b, c, c*]
   write a function that does the following

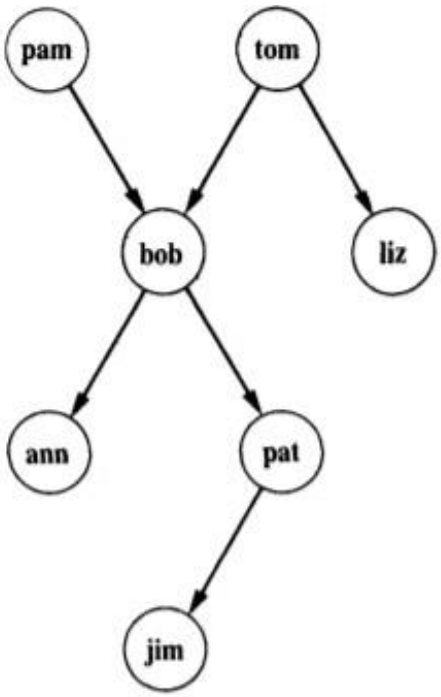
rle([*a, a, a, a, b, b, c, c*], *X*)
X : [*a, b, c*]

3. Given a list
   [*a, b, c, d, e, f, g*]
   write a function that does the following
   slice([*a, b, c, d, e, f, g*], [2, 5], *X*)
   X : [*c, d, e, f*]

4. Group list into sublists according to the distribution given. For example
   subsets([*a, b, c, d, e, f, g*], [2, 2, 3], *X,* [])
   should return *X* = [[*a, b*][*c, d*][*e, f, g*]]
   The order of the list does not matter

Family Tree

The following family tree was used.



Code

```
% parents
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

% genders
female(pam).
female(liz).
female(ann).
female(pat).
male(jim).
male(tom).
male(bob).

%  relations
mother(X, Y):- parent(X, Y), female(X).
father(X, Y):- parent(X, Y), male(X).

son(X, Y):- parent(Y, X), male(X).
daughter(X, Y):- parent(Y, X), female(X).

grandfather(X, Y):- parent(X, Z), parent(Z, Y), male(X).
grandmother(X, Y):- parent(X, Z), parent(Z, Y), female(X).

sibling(X, Y):- parent(Z, X), parent(Z, Y), X \= Y.
sister(X, Y):- sibling(X, Y), female(X).
brother(X, Y):- sibling(X, Y), male(X).

aunt(X, Y):- sister(X, Z), parent(Z, Y).
uncle(X, Y):- brother(X, Z), parent(Z, Y).

cousin(X, Y):- (aunt(Z, X);uncle(Z, X)), parent(Z, Y).

successor(X, Y):- parent(Y, X); (parent(Y, Z), successor(X, Z)).
predecessor(X, Y):- parent(X, Y); (parent(X, Z), predecessor(Z,
↔ Y)).
```

Output

```
> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [family_tree].
true.

2 ?- mother(pam, bob).
true.

3 ?- parent(X, bob).
X = pam ;
X = tom.

4 ?- aunt(liz, ann).
true .

5 ?- successor(X, pam).
X = bob ;
X = ann ;
X = pat ;
X = jim ;
false.

6 ?- grandfather(tom, X).
X = ann ;
X = pat ;
false.

7 ?- sibling(X, ann).
X = pat ;
false.

8 ?- predecessor(tom, jim).
true .
```

List Operations

Code

```
/*
Given a list
[a,a,a,a,b,b,b,c,c]
write a function that does the following
rle([a,a,a,a,b,b,c,c],X)
X: [a,b,c]
```

```
*/

% Only pick an element when the next one is different

% No next element
rle([X | []], [X]).
% Different next element
rle([X | [Y | Tail]], [X | L]) :-
    rle([Y | Tail], L),
    X \= Y.
% Same next element
rle([X | [X |  Tail]], L) :- rle([X | Tail], L).

/* Given a list
[a,b,c,d,e,f,g]
write a function that does the following
slice([a,b,c,d,e,f,g],[2,5],X)
X: [c,d,e,f]
*/

% Pick elements from index Y to the end

% If Y is 0 (or less), take all elements
sliceFrom(X, Y, X) :- Y =< 0.
% If the list is empty, answer is also an empty list
sliceFrom([], X, []) :- X > 0.
% If Y is positive, drop the first element, reduce Y by 1, and
↪  recurse
sliceFrom([_ | Tail], Y, L) :-
    Y > 0,
    sliceFrom(Tail, Y-1, L).

sliceUpto(X, Y, L) :- sliceFrom(X, Y, Suffix), conc(L, Suffix,
↪  X).

% Concatenate two lists

% If the first list is empty, answer is the second list
conc([], L, L).
% Set the first element of the first list as the Head of the
↪  answer,
% Tail of the answer is Tail of first list concatenated with the
↪  second list
conc([X | Tail], L2, [X | L3]) :- conc(Tail, L2, L3).

% Slice from Start to Finish, inclusive.
```

```
slice(L, [Start, Finish], X) :-
    % First take [0, Finish]
    sliceUpto(L, Finish+1, Prefix),
    % From this, take [Start, Finish]
    sliceFrom(Prefix, Start, X).

/*
Group list into sublists according to the distribution given
For example
subsets([a,b,c,d,e,f,g],[2,2,3],X,[])
should return X = [[a,b][c,d][e,f,g]]
The order of the list does not matter
*/

% Length of a list

% Length of an empty list is 0
len([], 0).
% Add 1 to the length of the tail of the list
len([_ | Tail], L) :- len(Tail, LMinusOne), L is LMinusOne + 1.

% If the list is empty, answer is also empty
subsets([], _, []).
% If the list is not empty but the subsets sizes are emtpy,
% answer is all the remaining elements in a list
subsets([Head | Tail], [], [[Head | Tail]]).
% If the first subset size is greater than or equal to the length
↔  of the list,
% answer is all the remaining elements in a list
subsets(X, [Head | _], [X]) :- len(X, Length), Head >= Length.
% If the first subset size is less than the length of the list,
% take a slice of the first Head elements and add it to the
↔  answer
% then recurse for the remaining elements and subset sizes.
subsets(X, [Head | Tail], [Subset|Remaining]) :-
   len(X, Length),
   Head < Length,
   sliceUpto(X, Head, Subset),
   sliceFrom(X, Head, RemainingX),
   subsets(RemainingX, Tail, Remaining).
```

Output

```
> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [list_operations].
true.

2 ?- rle([a, a, a, a, b, b, c, c], X).
X = [a, b, c] ;
false.

3 ?- slice([a, b, c, d, e, f, g], [2, 5], X).
X = [c, d, e, f] ;
false.

4 ?- subsets([a, b, c, d, e, f, g], [2, 2, 3], X).
X = [[a, b], [c, d], [e, f, g]] ;
false.
```

Conclusion

I learnt about PROLOG, and how to use it to solve logical problems using a declarative programming style.