Jaiwin Shah

2019130058

Batch C

AIML lab

Aim: Prolog Problem 1(Family tree)

Theory:

Prolog:

Prolog is a logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

The language was developed and implemented in Marseille, France, in 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses.

Prolog was one of the first logic programming languages and remains the most popular such language today, with several free and commercial implementations available. The language has been used for theorem proving, expert systems, term rewriting, type systems, and automated planning, as well as its original intended field of use, natural language processing. Modern Prolog environments support the creation of graphical user interfaces, as well as administrative and networked applications. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Implementation:

1. Create a family tree using PROLOG. It should have rules for father, mother, brother, sister, grandparent, uncle, aunt, predecessors, successors.

   Code:

```
female(arachana).
female(neha).
female(jyoti).
male(jaiwin).
male(shrey).
male(atul).
male(abhay).
male(jolly).
parent(atul,jaiwin).
parent(arachana,jaiwin).
parent(atul,shrey).
parent(arachana,shrey).
parent(jolly,atul).
parent(neha,atul).
parent(jolly,abhay).
parent(neha,sangita).
parent(jolly,jyoti).
parent(neha,jolly).
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).
sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.
grandparent(X,Y):- parent(X,Z),parent(Z,Y).
successor(X,Y):- parent(Y,X).
successor(X,Y):- parent(Z,X),successor(Z,Y).
predecessor(X,Y):- parent(X,Y).
predecessor(X,Y):- parent(Y,Z),parent(Z,X).
uncle(X,Y):- brother(X,Z),parent(Z,Y).
aunt(X,Y):- sister(X,Z),parent(Z,Y).
```

Output:

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]

Code:

```
rle([],[]). rle([X],[X]).
rle([X,X|Xs],Zs) :-
rle([X|Xs],Zs).
rle([X,Y|Ys],[X|Zs]) :- X \= Y, rle([Y|Ys],Zs).
```

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)

   Code:

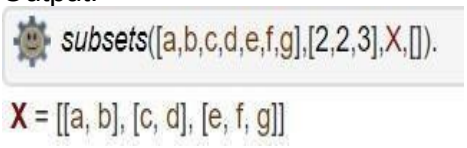
```
slice([X|_],1,1,[X]).
slice([X|Xs],1,K,[X|Ys]) :- K > 1,
K1 is K - 1, slice(Xs,1,K1,Ys).
slice([_|Xs],I,K,Ys) :- I > 1,
I1 is I - 1, K1 is K - 1, slice(Xs,I1,K1,Ys).
```

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

   Code:

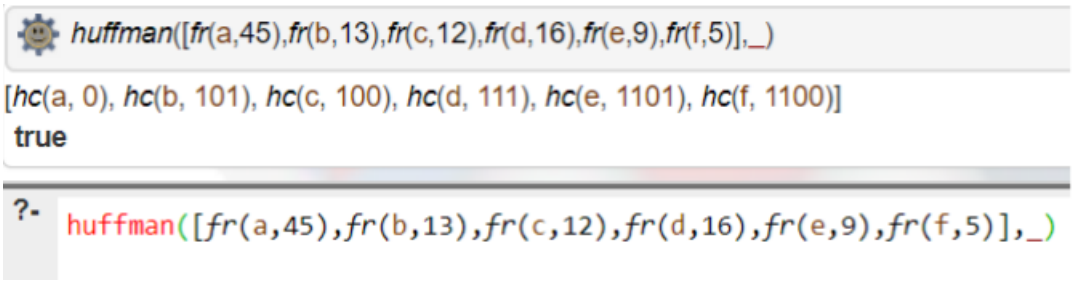
```
el(X,[X|L],L).
el(X,[_|L],R) :- el(X,L,R).
selectN(0,_,[]) :- !.
selectN(N,L,[X|S]) :- N >
0,    el(X,L,R),    N1 is N-
1,   selectN(N1,R,S).
subsets([],[],[],[]).
subsets(G,[N1|Ns],[G1|Gs],[]) :-
selectN(N1,G,G1),
subtract(G,G1,R),
subsets(R,Ns,Gs,[]).
```

   Output:



5. Huffman Code We suppose a set of symbols with their frequencies, given as a list of fr(S,F) terms. Example: [fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)]. Our objective is to construct a list hc(S,C) terms, where C is the Huffman code word for the symbol S. In our example, the result could be Hs =[hc(a,'0'), hc(b,'101'), hc(c,'100'), hc(d,'111'), hc(e,'1101'), hc(f,'1100')]. The task shall be performed by the predicate huffman/2 defined as follows: % huffman(Fs,Hs) :- Hs is the Huffman code table for the frequency table Fs
Code:

```
huffman(Fs,Cs) :-
    initialize(Fs,Ns),
    make_tree(Ns,T),
    traverse_tree(T,Cs).

initialize(Fs,Ns) :- init(Fs,NsU), sort(NsU,Ns).

init([],[]).
init([fr(S,F)|Fs],[n(F,S)|Ns]) :- init(Fs,Ns).

make_tree([T],T).
make_tree([n(F1,X1),n(F2,X2)|Ns],T) :-
    F is F1+F2,
    insert(n(F,s(n(F1,X1),n(F2,X2))),Ns,NsR),
    make_tree(NsR,T).

insert(N,[],[N]) :- !.
insert(n(F,X),[n(F0,Y)|Ns],[n(F,X),n(F0,Y)|Ns]) :- F < F0, !.
insert(n(F,X),[n(F0,Y)|Ns],[n(F0,Y)|Ns1]) :- F >= F0, insert(n(F,X),Ns,Ns1).

traverse_tree(T,Cs) :-
    traverse_tree(T,'',Cs1-[]), sort(Cs1,Cs),
    write(Cs).

traverse_tree(n(_,A),Code,[hc(A,Code)|Cs]-Cs) :-
    atom(A).

traverse_tree(n(_,s(Left,Right)),Code,Cs1-Cs3) :-
    atom_concat(Code,'0',CodeLeft),
    atom_concat(Code,'1',CodeRight),

traverse_tree(Left,CodeLeft,Cs1-Cs2),
traverse_tree(Right,CodeRight,Cs2-Cs3)
```



Conclusion:

From this experiment, I learned about prolog and the implementation of family tree. I performed list operations like removing duplicates from list, slicing a list, creating subsets of list of a given size. Also, I used prolog to generate a family tree and wrote the required rules and then executed the queries. The three basic contructs of Prolog are rules, facts and queries. The Prolog programs are knowledge bases, consisting of rules, facts and queries are asked to which the answer is found from this knowledge base and a Boolean value is sent.