

Pranav Nair  
2019130042  
TE Comps Batch – C  
22<sup>nd</sup> November, 2021

## Experiment-6

**Aim:** To solve problems using Prolog Programming.

**Problem statements:**

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

**Code:**

```
parent(kamla,padmesh).
parent(chandru,padmesh).
parent(chandru,manoj).
parent(kamla,manoj).
parent(harsha, pranav).
parent(padmesh, pranav).
parent(shobha, rahul).
parent(murli, rahul).
parent(narayan, harsha).
parent(lakshmi, harsha).
parent(lakshmi, shobha).
parent(narayan, shobha).
parent(shobha, nikhil).
parent(murli, nikhil).
parent(harsha, shruti).
parent(padmesh, shruti).
female(kamla).
female(harsha).
female(shruti).
female(shobha).
female(lakshmi).
male(murli).
male(manoj).
male(padmesh).
male(nikhil).
male(pranav).
male(rahul).
male(narayan).
male(chandru).

%rules
predecessor(X, Y) :- parent(X, Y).
predecessor(X, Y) :- parent(X, A),predecessor(A, Y).
successor(X, Y):- son(X,Y).
```

successor(X, Y):- daughter(X, Y).  
 successor(X, Y):- son(X, A), successor(A, Y).  
 successor(X, Y):- daughter(X, A), successor(A, Y).











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

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

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

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

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

 <code>predecessor(narayan,X)</code> <code>X = harsha</code> <code>X = shobha</code> <code>X = pranav</code> <code>X = shruti</code> <code>X = rahul</code> <code>X = nikhil</code> <hr/> <code>?- predecessor(narayan,X)</code>	 <code>successor(pranav,X)</code> <code>X = harsha</code> <code>X = padmesh</code> <code>X = narayan</code> <code>X = lakshmi</code> <code>X = kamla</code> <code>X = chandru</code> <hr/> <code>?- successor(pranav,X)</code>	 <code>grandfather(chandru,X)</code> <code>X = pranav</code> <code>X = shruti</code> <hr/> <code>?- grandfather(chandru,X)</code>	 <code>grandmother(X,nikhil)</code> <code>X = lakshmi</code> <hr/> <code>?- grandmother(X,nikhil)</code>
 <code>mother(harsha,X)</code> <code>X = pranav</code> <code>X = shruti</code> <hr/> <code>?- mother(harsha,X)</code>	 <code>father(X,harsha)</code> <code>X = narayan</code> <hr/> <code>?- father(X,harsha)</code>	 <code>aunt(X,shruti)</code> <code>X = shobha</code> <hr/> <code>?- aunt(X,shruti)</code>	 <code>uncle(manoj,X)</code> <code>X = pranav</code> <code>X = shruti</code> <hr/> <code>?- uncle(manoj,X)</code>
		 <code>sister(shruti,X)</code> <code>X = pranav</code> <hr/> <code>?- sister(shruti,X)</code>	 <code>son(pranav,X)</code> <code>X = harsha</code> <code>X = padmesh</code> <hr/> <code>?- son(pranav,X)</code>

- 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,b,c,c],X) X: [a,b,c]

### Code:

%stopping condition  
 remove\_duplicates([X],[X]).

%if next element is same  
 remove\_duplicates([H, H|T],X) :-  
     remove\_duplicates([H|T],X).

%if next element not same  
 remove\_duplicates([H, P|T],[H|Y]) :-  
     H\=P,  
     remove\_duplicates([P|T],Y).



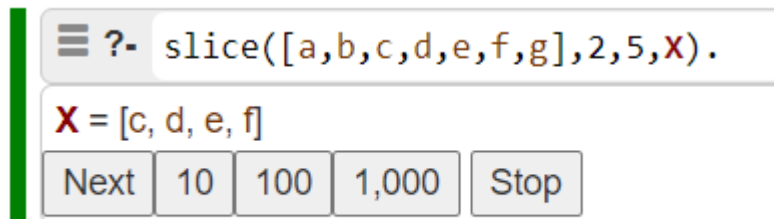
- 3) Given a list [a,b,c,d,e,f,g] write a function that does the following  
 slice([a,b,c,d,e,f,g],2,5,X) X: [c,d,e,f]

**Code:**

% puts last value of slice in output list  
 slice([X|\_], 0, 0, [X]).

% called until last slice index is reached  
 slice([X|T], 0, L, [X|T\_new]) :- L > 0, L\_new is L - 1, slice(T, 0, L\_new, T\_new).

% called until initial slice index is reached  
 slice(\_[T], F, L, Output) :- F > 0, F\_new is F - 1, L\_new is L - 1, slice(T, F\_new, L\_new, Output).



- 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,[]).

```
subsets([a,b,c,d,e,f,g],[2,2,3],X,[])
```

```
X = [[a, b], [c, d], [e, f, g]]
```

```
?- subsets([a,b,c,d,e,f,g],[2,2,3],X,[])
```

- 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).
```

```
 huffman([fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)],_)
```

```
[hc(a, 0), hc(b, 101), hc(c, 100), hc(d, 111), hc(e, 1101), hc(f, 1100)]
```

```
true
```

```
?- huffman([fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)],_)
```

### Conclusion:

In this experiment, the aim was to learn prolog programming, prolog has been used for logic programming. Here I performed list operations using prolog such as removing duplicates from list, slicing a list, creating subsets of list of given size and also to generate huffman code for the given symbols. Also I used prolog to generate a family tree and wrote the required rules and then executed the queries. Basically the three basic constructs of Prolog are rules, facts and queries. The Prolog programs are basically knowledge bases, consisting of rules, facts and queries are asked to which the answer is sought from this knowledge base and a boolean value is sent.

GitHub: <https://github.com/pranav567/AI-ML-Lab/tree/master/experiment-6>