Practical No. - 8

Name- Bhavesh Kewalramani

Roll No.- A-25

Section- A

Semester- 6th

Shift- 1st

Aim:

Introduction to Prolog [Family problem] and solve 8-queen problem by using prolog.

Code:

Family Tree

```
%Defining Males

male(bill).

male(john).

male(frank).

male(joe).

male(larrie).

male(philip).

male(kevin).

male(chris).

male(danny).


%Defining Females
```

```
female(anne).
female(dorothy).
female(sharon).
female(rochelle).
female(danelle).
female(marg).
female(connie).
female(corrie).

%Defining Parents
parent(bill,john).
parent(bill,frank).
parent(bill,joe).
parent(anne,john).
parent(anne,frank).
parent(anne,joe).
parent(john,larrie).
parent(john,sharon).
parent(john,philip).
parent(john,kevin).
parent(dorothy,larrie).
parent(dorothy,sharon).
parent(dorothy,philip).
```

```
parent(dorothy,kevin).
parent(larrie,connie).
parent(larrie,corrie).
parent(larrie,chris).
parent(marg,connie).
parent(marg,corrie).
parent(marg,chris).
parent(danny,rochelle).
parent(danny,danelle).
parent(sharon,rochelle).
parent(sharon,danelle).

%Defining father and mother predicates
father(X,Y) :- parent(X,Y) , male(X).
mother(X,Y) :- parent(X,Y) , female(X).

%Defining son and daughter predicates
son(X,Y) :- parent(Y,X) , male(X).
daughter(X,Y) :- parent(Y,X) , female(X).

%Defining brother and sister predicates
brother(X,Y) :- parent(Z,X) , son(Y,Z) , X \= Y.
sister(X,Y) :- parent(Z,X) , daughter(Y,Z) , X \= Y.
```

```
%Defining grandmother and grandfather predicates
grandfather(X,Y) :- parent(Z,Y) , father(X,Z).
grandmother(X,Y) :- parent(Z,Y) , mother(X,Z).

%Defining grandson and granddaughter predicates
grandson(X,Y) :- parent(Y,Z) , parent(Z,X), male(X).
granddaughter(X,Y) :- parent(Y,Z) , parent(Z,X), female(X).

%Defining uncle and aunt predicates
uncle(X,Y) :- parent(Z,Y), brother(Z,X).
aunt(X,Y) :- parent(Z,Y), sister(Z,X).

%Defining nephew predicate
nephew(X,Y) :- brother(Z,Y), son(X,Z).
nephew(X,Y) :- sister(Z,Y), son(X,Z).

%Defining niece predicate
niece(X,Y) :- brother(Z,Y), daughter(X,Z).
niece(X,Y) :- sister(Z,Y), daughter(X,Z).

%Defining cousin predicate
cousin(X,Y) :- uncle(Z,Y), son(Z,X).
```

```
cousin(X,Y) :- aunt(Z,Y), son(Z,X).

cousin(X,Y) :- uncle(Z,Y), daughter(Z,X).

cousin(X,Y) :- aunt(Z,Y), daughter(Z,X).


%Defining ancestor predicate

ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(Z,Y) , ancestor(X,Z).


%Defining descendant predicate

descendant(X,Y) :- son(X,Y).

descendant(X,Y) :- daughter(X,Y).

descendant(X,Y) :- son(X,Z) , descendant(Z,Y).

descendant(X,Y) :- daughter(X,Z) , descendant(Z,Y).
```

Output:

```
% c:/Users/bhave/OneDrive/Desktop/prolog/family.pl compiled 0.00 sec, 67 clauses
[1]  ?- son(X,Y).
X = john,
Y = bill ;
X = frank,
Y = bill ;
X = joe,
Y = bill ;
X = john,
Y = anne ;
X = frank,
Y = anne ;
X = joe,
Y = anne ;
X = larrie,
Y = john ;
X = philip,
Y = john ;
X = kevin,
Y = john ;
X = larrie,
Y = dorothy ;
X = philip,
Y = dorothy ;
X = kevin,
Y = dorothy ;
X = chris,
Y = larrie ;
X = chris,
Y = marg ;
false.
```

```
[1]  ?- granddaughter(X,Y).
X = sharon,
Y = bill ;
X = sharon,
Y = anne ;
X = connie,
Y = john ;
X = corrie,
Y = john ;
X = rochelle,
Y = john ;
X = danelle,
Y = john ;
X = connie,
Y = dorothy ;
X = corrie,
Y = dorothy ;
X = rochelle,
Y = dorothy ;
X = danelle,
Y = dorothy ;
false.
```

8-Queen's Problem

```
%Depth First Search strategy.

:- dynamic dfs/2, depth_first/3, check_safety/2, move/3.


%predicate to solve the depth_first of a list of nodes.


dfs(Node, Result):-

   depth_first(Node, Path_list, 8), %This returns a list of lists of nodes visited in the path.

   last(Path_list, Result). %since last list will be our output, returning that.


%predicate to solve this problem using DFS strategy.

% If current node is a goal state, then stop.


depth_first(_,[],0).

%Else continue to search, move to NextNode and collect all the nodes in the path.

% here 'V' is an accumulator.


depth_first(Node, [NextNode|Path], V):-

   move(Node, NextNode, V),

   V1 is V-1,

   depth_first(NextNode, Path, V1).
```

```
%predicate to check whether the queen position is safe or not.

check_safety(_,[]). %base case.

check_safety(X/Y, [X1/Y1|Rs]):-

    not(Y = Y1), %not in same row

    not(X = X1), %not in same column

    X1-X =\= Y1-Y, %not in ascending diagonal

    X1-X =\= Y-Y1, %not in descending diagonal

    check_safety(X/Y, Rs).


%predicate which adds safe position (X/Y) of a queen each time to the resultant
list.

move(Qlist, [X/Y|Qlist], V):-

    X is V, %each X takes the values from 8 to 1.

    member(Y, [1,2,3,4,5,6,7,8]), %getting some value for y.

    check_safety(X/Y, Qlist). %checking if position is safe or not.
```

Output:

```
?- dfs(X,Y).
X = [],
Y = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1] ;
X = [],
Y = [1/5, 2/2, 3/4, 4/7, 5/3, 6/8, 7/6, 8/1] ;
X = [],
Y = [1/3, 2/5, 3/2, 4/8, 5/6, 6/4, 7/7, 8/1] ;
X = [],
Y = [1/3, 2/6, 3/4, 4/2, 5/8, 6/5, 7/7, 8/1] ;
X = [],
Y = [1/5, 2/7, 3/1, 4/3, 5/8, 6/6, 7/4, 8/2] ;
X = [],
Y = [1/4, 2/6, 3/8, 4/3, 5/1, 6/7, 7/5, 8/2] ;
X = [],
Y = [1/3, 2/6, 3/8, 4/1, 5/4, 6/7, 7/5, 8/2] ;
X = [],
Y = [1/5, 2/3, 3/8, 4/4, 5/7, 6/1, 7/6, 8/2] ;
X = [],
Y = [1/5, 2/7, 3/4, 4/1, 5/3, 6/8, 7/6, 8/2] ;
X = [],
Y = [1/4, 2/1, 3/5, 4/8, 5/6, 6/3, 7/7, 8/2] ;
X = [],
Y = [1/3, 2/6, 3/4, 4/1, 5/8, 6/5, 7/7, 8/2] ;
X = [],
Y = [1/4, 2/7, 3/5, 4/3, 5/1, 6/6, 7/8, 8/2] ;
X = [],
Y = [1/6, 2/4, 3/2, 4/8, 5/5, 6/7, 7/1, 8/3] ;
X = [],
Y = [1/6, 2/4, 3/7, 4/1, 5/8, 6/2, 7/5, 8/3] ;
X = [],
Y = [1/1, 2/7, 3/4, 4/6, 5/8, 6/2, 7/5, 8/3] ;
X = [],
Y = [1/6, 2/8, 3/2, 4/4, 5/1, 6/7, 7/5, 8/3] ;
X = [],
Y = [1/6, 2/2, 3/7, 4/1, 5/4, 6/8, 7/5, 8/3] ;
X = [],
Y = [1/4, 2/7, 3/1, 4/8, 5/5, 6/2, 7/6, 8/3] ;
X = [],
Y = [1/5, 2/8, 3/4, 4/1, 5/7, 6/2, 7/6, 8/3] ;
X = [],
Y = [1/4, 2/8, 3/1, 4/5, 5/7, 6/2, 7/6, 8/3] ;
X = [],
Y = [1/2, 2/7, 3/5, 4/8, 5/1, 6/4, 7/6, 8/3] ;
X = [],
Y = [1/1, 2/7, 3/5, 4/8, 5/2, 6/4, 7/6, 8/3] ;
X = [],
Y = [1/2, 2/5, 3/7, 4/4, 5/1, 6/8, 7/6, 8/3] ;
```

```
X = [],
Y = [1/5, 2/1, 3/8, 4/6, 5/3, 6/7, 7/2, 8/4] ;
X = [],
Y = [1/1, 2/5, 3/8, 4/6, 5/3, 6/7, 7/2, 8/4] ;
X = [],
Y = [1/3, 2/6, 3/8, 4/1, 5/5, 6/7, 7/2, 8/4] ;
X = [],
Y = [1/6, 2/3, 3/1, 4/7, 5/5, 6/8, 7/2, 8/4] ;
X = [],
Y = [1/7, 2/5, 3/3, 4/1, 5/6, 6/8, 7/2, 8/4] ;
X = [],
Y = [1/7, 2/3, 3/8, 4/2, 5/5, 6/1, 7/6, 8/4] ;
X = [],
Y = [1/5, 2/3, 3/1, 4/7, 5/2, 6/8, 7/6, 8/4] ;
X = [],
Y = [1/2, 2/5, 3/7, 4/1, 5/3, 6/8, 7/6, 8/4] ;
X = [],
Y = [1/3, 2/6, 3/2, 4/5, 5/8, 6/1, 7/7, 8/4] ;
X = [],
Y = [1/6, 2/1, 3/5, 4/2, 5/8, 6/3, 7/7, 8/4] ;
X = [],
Y = [1/8, 2/3, 3/1, 4/6, 5/2, 6/5, 7/7, 8/4] ;
X = [],
Y = [1/2, 2/8, 3/6, 4/1, 5/3, 6/5, 7/7, 8/4] ;
X = [],
Y = [1/5, 2/7, 3/2, 4/6, 5/3, 6/1, 7/8, 8/4] ;
X = [],
Y = [1/3, 2/6, 3/2, 4/7, 5/5, 6/1, 7/8, 8/4] ;
X = [],
Y = [1/6, 2/2, 3/7, 4/1, 5/3, 6/5, 7/8, 8/4] ;
X = [],
Y = [1/3, 2/7, 3/2, 4/8, 5/6, 6/4, 7/1, 8/5] ;
X = [],
Y = [1/6, 2/3, 3/7, 4/2, 5/4, 6/8, 7/1, 8/5] ;
X = [],
Y = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/1, 8/5] ;
X = [],
Y = [1/7, 2/1, 3/3, 4/8, 5/6, 6/4, 7/2, 8/5] ;
X = [],
Y = [1/1, 2/6, 3/8, 4/3, 5/7, 6/4, 7/2, 8/5] ;
X = [],
Y = [1/3, 2/8, 3/4, 4/7, 5/1, 6/6, 7/2, 8/5] ;
X = [],
Y = [1/6, 2/3, 3/7, 4/4, 5/1, 6/8, 7/2, 8/5] ;
X = [],
Y = [1/7, 2/4, 3/2, 4/8, 5/6, 6/1, 7/3, 8/5] ;
X = [],
Y = [1/4, 2/6, 3/8, 4/2, 5/7, 6/1, 7/3, 8/5] ;
```

```
X = [],
Y = [1/7, 2/4, 3/2, 4/5, 5/8, 6/1, 7/3, 8/6] ;
X = [],
Y = [1/8, 2/2, 3/4, 4/1, 5/7, 6/5, 7/3, 8/6] ;
X = [],
Y = [1/7, 2/2, 3/4, 4/1, 5/8, 6/5, 7/3, 8/6] ;
X = [],
Y = [1/5, 2/1, 3/8, 4/4, 5/2, 6/7, 7/3, 8/6] ;
X = [],
Y = [1/4, 2/1, 3/5, 4/8, 5/2, 6/7, 7/3, 8/6] ;
X = [],
Y = [1/5, 2/2, 3/8, 4/1, 5/4, 6/7, 7/3, 8/6] ;
X = [],
Y = [1/3, 2/7, 3/2, 4/8, 5/5, 6/1, 7/4, 8/6] ;
X = [],
Y = [1/3, 2/1, 3/7, 4/5, 5/8, 6/2, 7/4, 8/6] ;
X = [],
Y = [1/8, 2/2, 3/5, 4/3, 5/1, 6/7, 7/4, 8/6] ;
X = [],
Y = [1/3, 2/5, 3/2, 4/8, 5/1, 6/7, 7/4, 8/6] ;
X = [],
Y = [1/3, 2/5, 3/7, 4/1, 5/4, 6/2, 7/8, 8/6] ;
X = [],
Y = [1/5, 2/2, 3/4, 4/6, 5/8, 6/3, 7/1, 8/7] ;
X = [],
Y = [1/6, 2/3, 3/5, 4/8, 5/1, 6/4, 7/2, 8/7] ;
X = [],
Y = [1/5, 2/8, 3/4, 4/1, 5/3, 6/6, 7/2, 8/7] ;
X = [],
Y = [1/4, 2/2, 3/5, 4/8, 5/6, 6/1, 7/3, 8/7] ;
X = [],
Y = [1/4, 2/6, 3/1, 4/5, 5/2, 6/8, 7/3, 8/7] ;
X = [],
Y = [1/6, 2/3, 3/1, 4/8, 5/5, 6/2, 7/4, 8/7] ;
X = [],
Y = [1/5, 2/3, 3/1, 4/6, 5/8, 6/2, 7/4, 8/7] ;
X = [],
Y = [1/4, 2/2, 3/8, 4/6, 5/1, 6/3, 7/5, 8/7] ;
X = [],
Y = [1/6, 2/3, 3/5, 4/7, 5/1, 6/4, 7/2, 8/8] ;
X = [],
Y = [1/6, 2/4, 3/7, 4/1, 5/3, 6/5, 7/2, 8/8] ;
X = [],
Y = [1/4, 2/7, 3/5, 4/2, 5/6, 6/1, 7/3, 8/8] ;
X = [],
Y = [1/5, 2/7, 3/2, 4/6, 5/3, 6/1, 7/4, 8/8] ;
false.
```