Write a program to solve missionaries and cannibal problem

```
% Initial and goal states
start([3,3,left,0,0]).
goal([0,0,right,3,3]).

% Check if a state is legal
legal(CL, ML, CR, MR) :-
   CL >= 0, ML >= 0, CR >= 0, MR >= 0,
   (ML >= CL ; ML = 0),
   (MR >= CR ; MR = 0).

% Possible moves
move([CL, ML, left, CR, MR], [CL, ML2, right, CR, MR2]) :- % Two missionaries cross left to right
   ML2 is ML - 2, MR2 is MR + 2, legal(CL, ML2, CR, MR2).
move([CL, ML, left, CR, MR], [CL2, ML, right, CR2, MR]) :- % Two cannibals cross left to right
   CL2 is CL - 2, CR2 is CR + 2, legal(CL2, ML, CR2, MR).
move([CL, ML, left, CR, MR], [CL2, ML2, right, CR2, MR2]) :- % One missionary and one cannibal cross left to right
   CL2 is CL - 1, CR2 is CR + 1, ML2 is ML - 1, MR2 is MR + 1, legal(CL2, ML2, CR2, MR2).
move([CL, ML, left, CR, MR], [CL, ML2, right, CR, MR2]) :- % One missionary crosses left to right
   ML2 is ML - 1, MR2 is MR + 1, legal(CL, ML2, CR, MR2).
move([CL, ML, left, CR, MR], [CL2, ML, right, CR2, MR]) :- % One cannibal crosses left to right
   CL2 is CL - 1, CR2 is CR + 1, legal(CL2, ML, CR2, MR).
move([CL, ML, right, CR, MR], [CL, ML2, left, CR, MR2]) :- % Two missionaries cross right to left
   ML2 is ML + 2, MR2 is MR - 2, legal(CL, ML2, CR, MR2).
move([CL, ML, right, CR, MR], [CL2, ML, left, CR2, MR]) :- % Two cannibals cross right to left
   CL2 is CL + 2, CR2 is CR - 2, legal(CL2, ML, CR2, MR).
move([CL, ML, right, CR, MR], [CL2, ML2, left, CR2, MR2]) :- % One missionary and one cannibal cross right to left
   CL2 is CL + 1, CR2 is CR - 1, ML2 is ML + 1, MR2 is MR - 1, legal(CL2, ML2, CR2, MR2).
move([CL, ML, right, CR, MR], [CL, ML2, left, CR, MR2]) :- % One missionary crosses right to left
   ML2 is ML + 1, MR2 is MR - 1, legal(CL, ML2, CR, MR2).
move([CL, ML, right, CR, MR], [CL2, ML, left, CR2, MR]) :- % One cannibal crosses right to left
   CL2 is CL + 1, CR2 is CR - 1, legal(CL2, ML, CR2, MR).

% Recursive pathfinding
path(State, Goal, Visited, Moves) :-
```

```
    move(State, NewState),
    \+ member(NewState, Visited),
    path(NewState, Goal, [NewState | Visited], [[NewState, State] | Moves]).
path(Goal, Goal, _, Moves) :-
    output(Moves).

% Output solution steps
output([]) :- nl.
output([[A, B] | Moves]) :-
    output(Moves),
    write(B), write(' -> '), write(A), nl.

% Find solution
find :-
    start(Start), goal(Goal),
    path(Start, Goal, [Start], []).
```

Quires:

?- find.

[3,3,left,0,0] -> [1,3,right,2,0]

[1,3,right,2,0] -> [2,3,left,1,0]

[2,3,left,1,0] -> [0,3,right,3,0]

[0,3,right,3,0] -> [1,3,left,2,0]

[1,3,left,2,0] -> [1,1,right,2,2]

[1,1,right,2,2] -> [2,2,left,1,1]

[2,2,left,1,1] -> [2,0,right,1,3]

[2,0,right,1,3] -> [3,0,left,0,3]

[3,0,left,0,3] -> [1,0,right,2,3]

[1,0,right,2,3] -> [1,1,left,2,2]

[1,1,left,2,2] -> [0,0,right,3,3]

true