Computer Intelligence Lab - CS354N

Weekly Assignment Report

Question 1:

```
gcd(0, X, X) :-!.
gcd(X, 0, X) :-!.
gcd(X, X, X) :-!.
gcd(M, N, X) :- N>M, Y is N-M, gcd(M, Y, X).
gcd(M, N, X) :- N<M, Y is M-N, gcd(Y, N, X).
```

Output:

```
?- gcd(24,16,G).
G = 8 .

?- gcd(36, 63, G).
G = 9 .

?- gcd(3, 3, G).
G = 3 .

?- gcd(30, 2, G).
G = 2 .
```

Question 2:

```
move(state(X, Y), fill4, state(4, Y)) :- X < 4.
move(state(X, Y), fill3, state(X, 3)) :- Y < 3.
move(state(X, Y), empty4, state(0, Y)) :- X > 0.
move(state(X, Y), empty3, state(X, 0)) :- Y > 0.
move(state(X, Y), pour4to3, state(NewX, NewY)) :-
    X > 0, Y < 3,
    Transfer is min(X, 3 - Y),
    NewX is X - Transfer,
    NewY is Y + Transfer.
move(state(X, Y), pour3to4, state(NewX, NewY)) :-
    Y > 0, X < 4,
    Transfer is min(Y, 4 - X),
    NewX is X + Transfer,
    NewY is Y - Transfer.

solve(JugStates) :- solve(state(0, 0), [], JugStates).

solve(state(2, _), _, []) :- !.
solve(CurrentState, Visited, [Action|Actions]) :-
    move(CurrentState, Action, NextState),
    \+ member(NextState, Visited),
    solve(NextState, [NextState|Visited], Actions).
```

Output:

```
?- solve(Steps).
Steps = [fill4, fill3, empty4, pour3to4, fill3, pour3to4, empty4, pour3to4] .
```

Question 3:

```
is_at(monkey, door).
is_at(box, window).
is_at(banana, middle).
hungry(monkey).

grasp(monkey, banana) :-
    hungry(monkey),
    writeln("Monkey attempts to grasp the banana."),
    climb(monkey, box),
    is_at(banana, middle).

climb(monkey, box) :-
    writeln("Monkey climbs the box."),
    is_at(monkey, box, middle).

is_at(monkey, box, middle) :-
    writeln("Monkey is now on the box in the middle."),
    push(monkey, box, middle).

push(monkey, box, middle) :-
    writeln("Monkey pushes the box from the window to the middle."),
    is_at(box, window),
    is_at(monkey, window).

is_at(monkey, window) :-
    writeln("Monkey goes to the window."),
    walk_to(monkey, window).

walk_to(monkey, window) :-
    writeln("Monkey walks from L to the window."),
    is_at(monkey, L),
    L \= window.
```

Output:

```
?- grasp(monkey, banana).
Monkey attempts to grasp the banana.
Monkey climbs the box.
Monkey is now on the box in the middle.
Monkey pushes the box from the window to the middle.
Monkey goes to the window.
Monkey walks from L to the window.
true .
```

Question 4:

```
use_module(library(lists)).

n_queen(N, Solution) :-
    length(Solution, N),
    queen(Solution, N).

up2N(N,N,[N]) :-!.
up2N(K,N,[K|Tail]) :- K < N, K1 is K+1, up2N(K1, N, Tail).

queen([],_).
queen([Q|Qlist],N) :-
    queen(Qlist, N),
    up2N(1,N,Candidate_positions_for_queenQ),
    member(Q, Candidate_positions_for_queenQ),
    check_solution(Q,Qlist, 1).

check_solution(_,[], _).
check_solution(Q,[Q1|Qlist],Xdist) :-
    Q =\= Q1,
    Test is abs(Q1-Q),
    Test =\= Xdist,
    Xdist1 is Xdist + 1,
    check_solution(Q,Qlist,Xdist1).
```

Output:

```
?- n_queen(8, Solution).
Solution = [4, 2, 7, 3, 6, 8, 5, 1] .
```

Each element in the list corresponds to a row on the chessboard, and the value of each element represents the column position of the queen in that row.

Question 5:

```
% Possible knight moves
jump(N, X/Y, U/V) :-
    member((DX, DY), [(2, 1), (1, 2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -2), (2, -1)]),
    U is X + DX, U > 0, U =< N,
    V is Y + DY, V > 0, V =< N.

% Solve knight's tour
knights_tour(N, Path) :-
    N2 is N * N,
    Path = [1/1 | Rest], % Start from top-left corner
    knights_tour(N, [1/1], Rest, N2).

knights_tour(_, Visited, [], 1) :-
    length(Visited, 1). % Base case: all squares are visited.

knights_tour(N, Visited, [Next | Rest], SquaresLeft) :-
    Visited = [Current | _],
    jump(N, Current, Next),
    \+ member(Next, Visited),
    NewSquaresLeft is SquaresLeft - 1,
    knights_tour(N, [Next | Visited], Rest, NewSquaresLeft).
```

Output:

```
?- knights_tour(1, Path).
Path = [1/1] .

?- knights_tour(2, Path).
false.

?- knights_tour(3, Path).
false.

?- knights_tour(4, Path).
false.

?- knights_tour(5, Path).
false.
```