DEERWALK INSTITUTE OF TECHNOLOGY

LAB 3: PROLOG BASICS

(ARTIFICIAL INTELLIGENCE)

SUBMITTED BY:

NAME: SUSHIL AWALE

PROGRAM: B.SC.CSIT (FIFTH SEM)

ROLL NO.: 0540

SECTION: A

SUBMITTED TO:

BIRODH RIJAL

DATE: 13 April 2018

KATHMANDU, NEPAL

2018

A. Given the following directed graph:

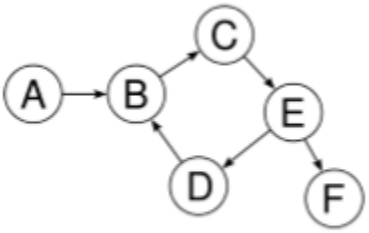

Figure 1: A directed graph with six nodes

Write a prolog program that would check if there is a path from any node X to any other node Y. Encode your database of facts like: edge('A','B'). , edge('E','F'). and so on.

Now wrie a rule for the predicate path(X,Y) that is true if there is a path from node represented by the variable X to the path represented by variable Y. You might have guessed that this requires the rule to be recursive. As you have to do while writing recursive function, think of the base case and the recursive step.

- - graph.pl - -

edge(a,b).

edge(b,c).

edge(d,b).

edge(c,e).

edge(e,d).

edge(e,f).

path(X,Y):- edge(X,Y).

path(X,Y):- edge(X,Z), path(Z,Y).

- - -

path(a,f).

true .

path(a,d).

true .

path(f, a).

false.

path(d, e).

true .

B. Using the concept of recursion add a rule predecessor(X,Y) to the family tree in the third lab that is true if X is the predecessor of Y (i.e. X comes before Y in the parent relation).
- - family.pl - -

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

predecessor(X,Y):- parent(X,W),predecessor(W,Y).

- - -

predecessor(abraham, maggie).

true .

predecessor(bart, homer).

false.

C. There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey may use.
- - monkey.pl - -

do(state(middle, onbox, middle, hasnot), grab, state(middle, onbox, middle, has)).

do(state(L, onfloor, L, Banana), climb, state(L, onbox, L, Banana)).

do(state(L1, onfloor, L1, Banana), push(L1, L2), state(L2, onfloor, L2, Banana)).

do(state(L1, onfloor, Box, Banana), walk(L1, L2), state(L2, onfloor, Box, Banana)).

canget(state(_, _, _, has)).

canget(State1) :- do(State1, Action, State2), canget(State2).

- - -

canget(state(atwindow, onfloor, atwindow, has)).

true .

canget(state(window, onfloor, atwindow, hasnot)).

true .

canget(state(atwindow, onbox, atwindow, hasnot)).

false.

D. Towers of Hanoi

To move n discs from peg A to peg C, the recursive algorithms is:

move n1 discs from A to B. This leaves disc n alone on peg A

move disc n from A to C

move n1 discs from B to C so they sit on disc n

- - Hanoi.pl - -

```
move(1,X,Y,_) :- write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N,X,Y,Z) :- N > 1, M is N-1, move(M,X,Z,Y), move(1,X,Y,_), move(M,Z,Y,X).

hanoi(N):- move(N,'A','C','B').
```

- - - -

```
hanoi(3).

Move top disk from A to C

Move top disk from A to B

Move top disk from C to B

Move top disk from A to C

Move top disk from B to A

Move top disk from B to C

Move top disk from A to C

true
```