Concepts of Programming Languages, CSCI 305, Fall 2021
Lab 6, Prolog – tracing programs
Section 12.2.4 Search/Execution Order, Oct. 22

Commands:

```
?- trace.   % To turn off: ?- notrace.
```

Standard ports:

call, exit, redo, fail

creep - continue

The tracer displays the port name, the current depth of the recursion and the goal. For example, when tracing is on, at the query:

```
?- factorial(3,X).
```

the following is displayed:

```
Call: (10) factorial(3, _13646) ? creep
Call: (11) 3>0 ? creep
Exit: (11) 3>0 ? creep
Call: (11) _16346 is 3+ -1 ? creep
Exit: (11) 2 is 3+ -1 ? creep
Call: (11) factorial(2, _17856) ? creep
```

1. Enter and run the following program:

```
factorial(0,1).
factorial(N,Ans) :-
       N>0,
       K is N-1,
       factorial(K, SubAns),
       Ans is SubAns * N.
```

2. Turn on tracing and execute factorial(3,X). Make sense out of the "calls", "exits", levels and goals, so that given a snippet of code and a query, you could tell how Prolog would resolve the query.

factorial(3,X)
Starts at level 8
Call factorial(3,_1336)

    Call 3>0?
    Exit 3>0?
    Call _1560 is 3+-1
    Exit 2 is 3+-1
    Call factorial(2,_1562)

        Call 2>0
        Exit 2>0
        Call _1566 is 2+-1
        Exit 1 is 2+-1
        Call factorial(1, _1568)

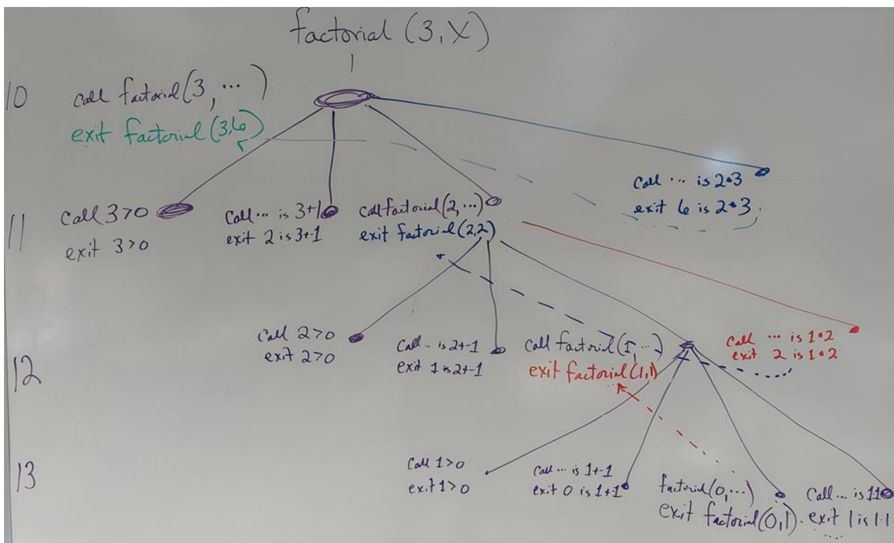
Call 1>0
Exit 1>0
Call _1572 is 1 +-1
Exit 0 is 1+-1
Call factorial(0, _1574)
Exit factorial(0,1)
Call _1578 is 1*1
Exit 1 is 1*1

Exit factorial(1,1)
Call _1584 is 1*2
Exit 2 is 1*2

Exit factorial(2,2)
_1336 is 2*3
6 is 2*3

Call factorial(3,6)

X=6



3. The following database represents unidirectional edges in a graph.

```
% Specify edges.
edge(a, b).
edge(a, c).
edge(a, d).
edge(b, e).
edge(e, f).
edge(e, g).
edge(g, h).
edge(g, i).
edge(d, j).
```

4. Enter the above facts into the database. Define a path function which takes a source and destination vertex and returns true if a path exists between the source and destination, and false otherwise. (If source=destination, return true).

```
path(S,S).
path(S,D) :- edge(S,X), path(X,D).
```

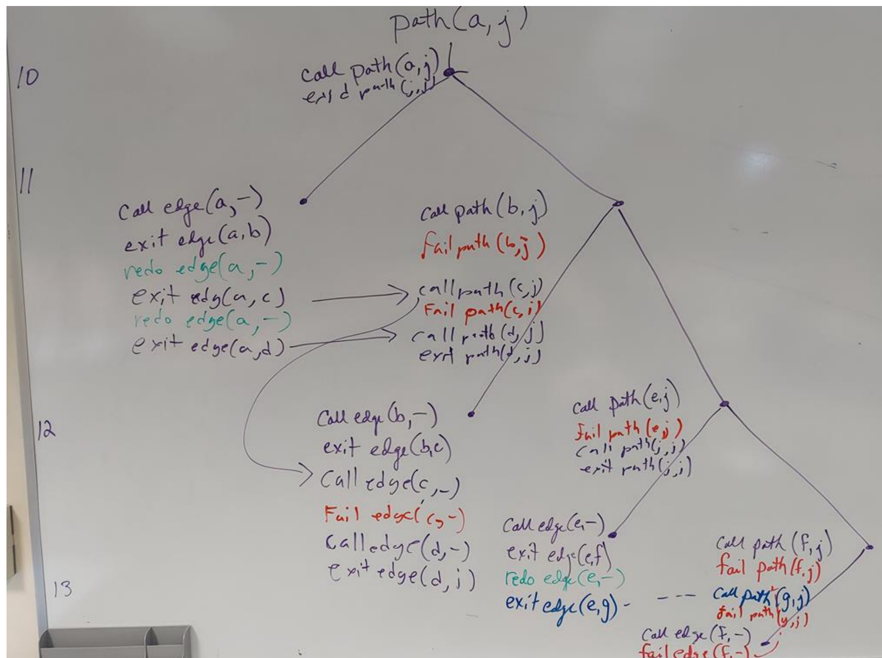
5. Resolving the query

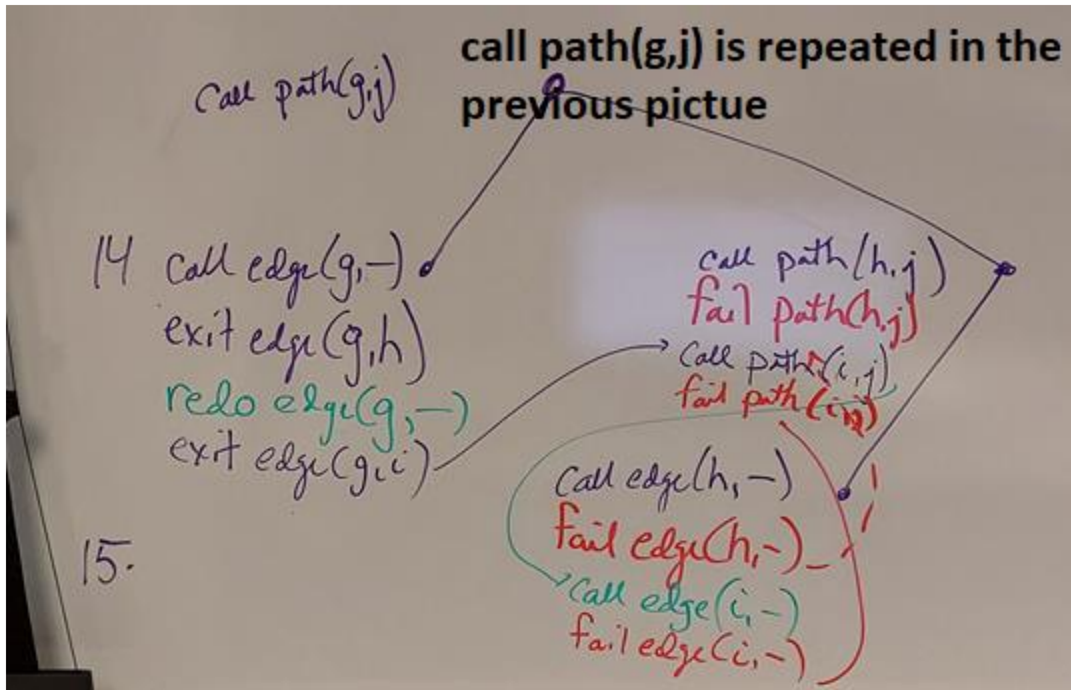
```
?- path(a,j).
```

will use the “fail” and “redo” ports, in addition to “calls”, “exits”. Understand the ports, levels and goals, so that given a snippet of code and a query, you could tell how Prolog would resolve the query.

```
Call: (8) path(a, j) ? creep
        Call: (9) edge(a, _874) ? creep
        Exit: (9) edge(a, b) ? creep
        Call: (9) path(b, j) ? creep
                Call: (10) edge(b, _874) ? creep
                Exit: (10) edge(b, e) ? creep
                Call: (10) path(e, j) ? creep
                        Call: (11) edge(e, _874) ? creep
                        Exit: (11) edge(e, f) ? creep
                        Call: (11) path(f, j) ? creep
                                Call: (12) edge(f, _874) ? creep
                                Fail: (12) edge(f, _874) ? creep
                        Fail: (11) path(f, j) ? creep
                        Redo: (11) edge(e, _874) ? creep
                        Exit: (11) edge(e, g) ? creep
                        Call: (11) path(g, j) ? creep
                                Call: (12) edge(g, _874) ? creep
                                Exit: (12) edge(g, h) ? creep
                                Call: (12) path(h, j) ? creep
                                        Call: (13) edge(h, _874) ? creep
```

Fail: (13) edge(h, _874) ? creep
Fail: (12) path(h, j) ? creep
Redo: (12) edge(g, _874) ? creep
Exit: (12) edge(g, i) ? creep
Call: (12) path(i, j) ? creep
Call: (13) edge(i, _874) ? creep
Fail: (13) edge(i, _874) ? creep
Fail: (12) path(i, j) ? creep
Fail: (11) path(g, j) ? creep
Fail: (10) path(e, j) ? creep
Fail: (9) path(b, j) ? creep
Redo: (9) edge(a, _874) ? creep
Exit: (9) edge(a, c) ? creep
Call: (9) path(c, j) ? creep
Call: (10) edge(c, _874) ? creep
Fail: (10) edge(c, _874) ? creep
Fail: (9) path(c, j) ? creep
Redo: (9) edge(a, _874) ? creep
Exit: (9) edge(a, d) ? creep
Call: (9) path(d, j) ? creep
Call: (10) edge(d, _874) ? creep
Exit: (10) edge(d, j) ? creep
Call: (10) path(j, j) ? creep
Exit: (10) path(j, j) ? creep
Exit: (9) path(d, j) ? creep
Exit: (8) path(a, j) ? creep





6. Add code to the above so that the vertices traversed in the path is returned.

```
path(S,S,[S|[]).
path(S,D, Path) :-
        edge(S,X),
        path(X,D, SubPath),
        Path = [S|SubPath].
```

7. Towers of Hanoi: The objective of this famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for N=3 disks.

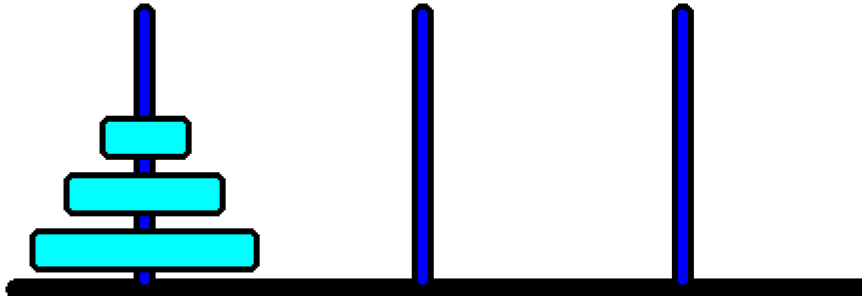


Fig. 2.3

This problem can be solved in Prolog using only two clauses:

1. A clause to move 1 disc from peg X to peg Y, using peg Z as a helper.
2. A clause to move more than one disc from peg X to peg Y, using peg Z as a helper.

Here is the first clause:

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

Complete this program.

```
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).
```

8. Try your code for N=3 and verify that it works.

```
?- move(3,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right

yes
```

https://www.cpp.edu/~jrfisher/www/prolog_tutorial/2_3.html