Evan Edelstein

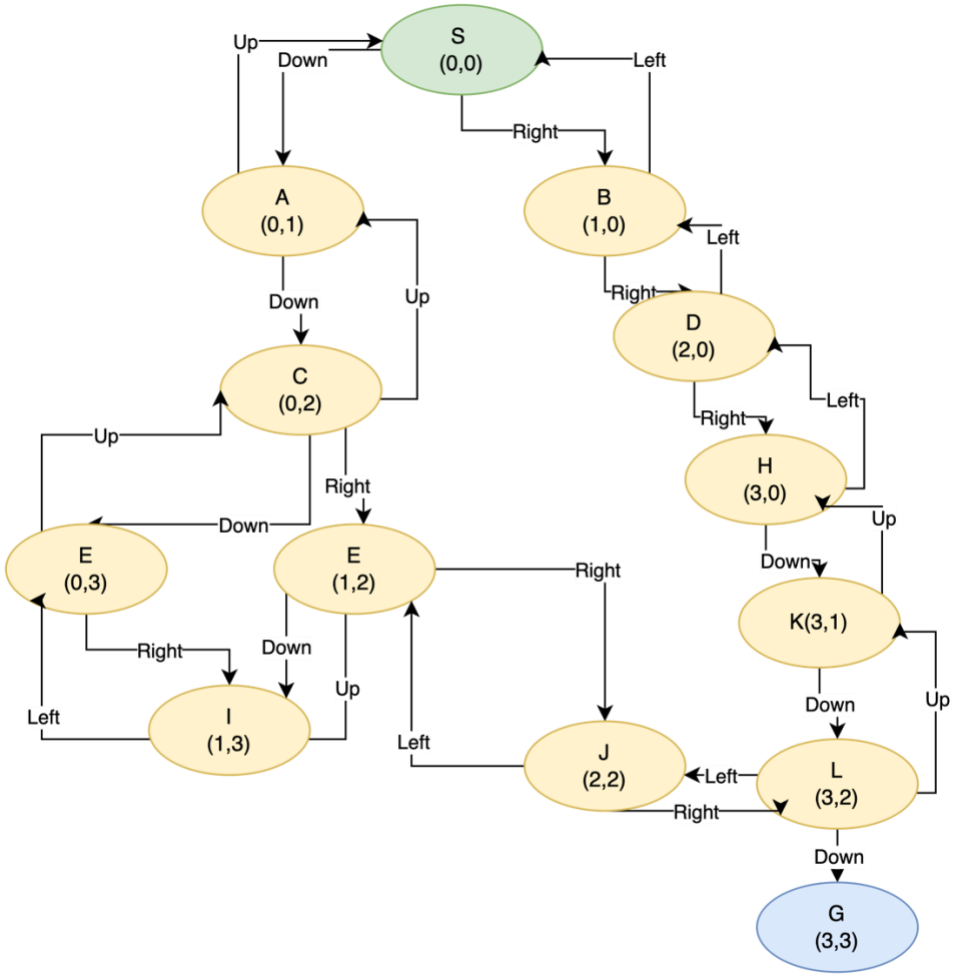
EN.605.645.82.SP26

Module 2 – Self check

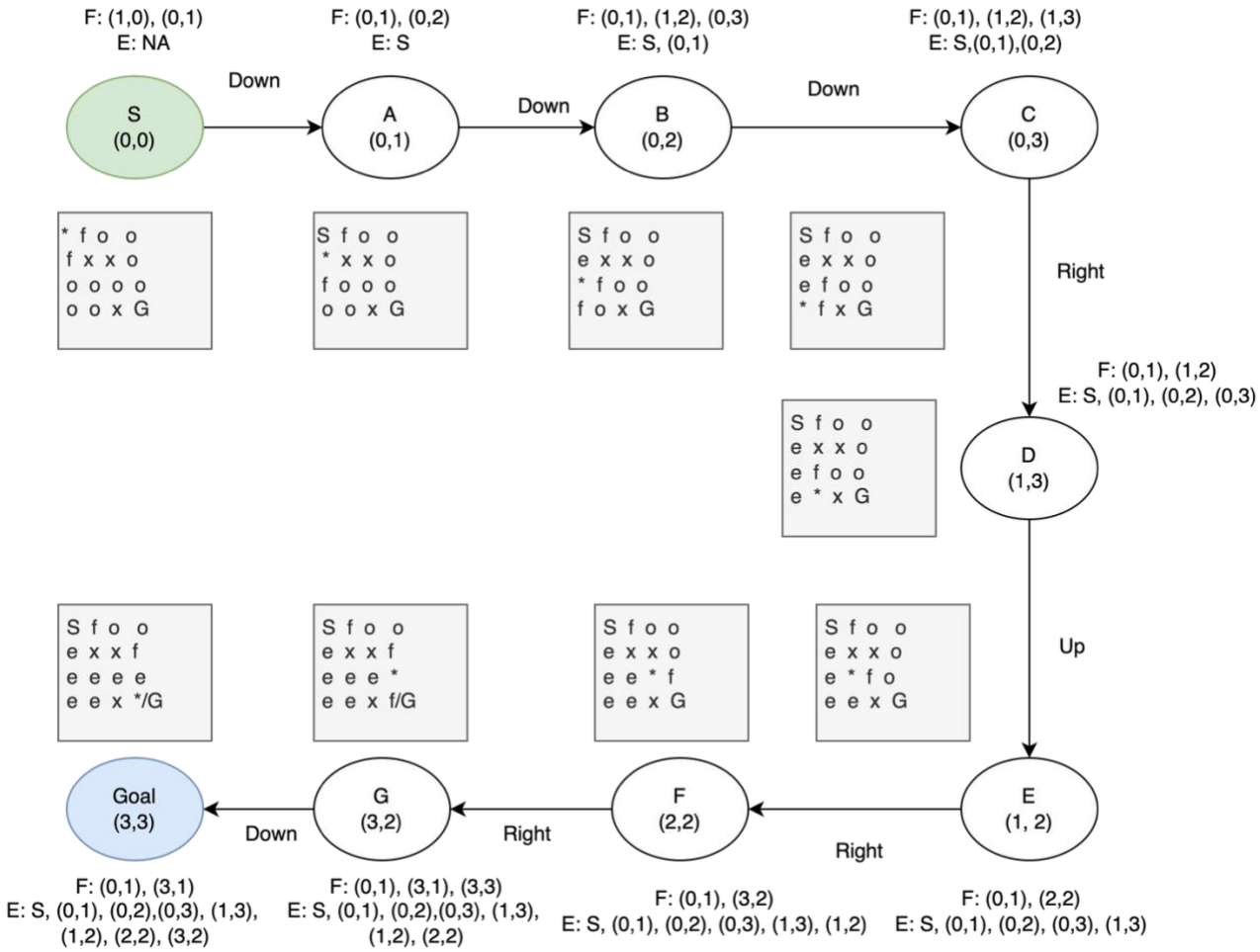
1. In this problem a state refers to a valid position of the robot in the microworld. Each state can be represented as a 2d-coordinate in the microworld space. The set of states *S* in this problem include the starting state S_0, when the robot is in the starting position "S", the goal state S_g, when the robot is in the goal position "G" and interim states as the robot traverses between S_0 and S_g. There are no failure states in this problem.

2. The set of transitions would involve the robot moving one space in the four cardinal directions, assuming they are within the bound of the microworld and not blocked ("x"). Mathematically we can denote the four moves for the robot at position (x,y) as: Right: (x,y) -> (x+1,y), Left: (x,y) -> (x-1, y), Down: (x,y) -> (x, y+1) ,Up: (x,y) -> (x, y-1). We can encode the boundary condition by bounding x and y to be between 0 and 3 (0 <= x <=3 and 0 <= y <= 3). This formula, along with blocked ("x") position checks, (microworld[y][x]!= "x"), allows us to generate transitions from any given state, avoiding the need to specify set of transitions explicitly for each space. Given the starting example, the transitions would be: Down: (0,0) -> (0,1), Right: (0,0) -> (1,0). The robot cannot move left or up since those positions are out of bounds.

3. The traversal graph of this microworld is built up piecemeal as exploring new spaces extends the frontier of positions to move to. The graph for all possible moves is:


S
(0,0)
Up
Down
Left
Right
A
(0,1)
B
(1,0)
Left
Down
Up
Right
D
(2,0)
C
(0,2)
Left
Up
Right
H
(3,0)
Right
Down
Up
Down
E
(0,3)
E
(1,2)
Right
K(3,1)
Right
Down
Up
Down
Up
Left
I
(1,3)
Left
J
(2,2)
Left
L
(3,2)
Right
Down
G
(3,3)

4. DFS:



5. Assuming each action has a cost of one, the DFS path has a cost of 8. A more direct path, i.e. right 3 and then down 3, would have a cost of 6. Therefore, the DFS path is not optimal in this case. In general, DFS does not guarantee an optimal path since it is not a complete search.

6. This fold was taken due to the successor function precedence order (west > south > east > north). Therefore, at (0,2) the successor function chose to go south instead of east, even though south would lead the robot towards a blocked route (boundary on west and south, block on east). If the precedence of east was greater than south, we could have avoided the fold altogether, as we would have taken the more direct path to the right; (0,0), (1,0), (2,0), (3,0), (3,1), (3,2), (3,3). Genrealy, we cannot create a successor state precedence that guarantees avoiding folds for any placement of G. (unless we remove a cardinal direction). Although, incorporating a cost function and/or heuristic into the successor function, like in A*, can help guide agents to avoid paths that lead towards blocked spaces or boundaries.

7. To track the path taken by the search we can store the parent coordinate for each traversed child space in an auxiliary list or table. Backtracking through this data structure would then build the path taken by the DFS.