FIT5047
Assignment 1:
Dishi Jain
30759307

Question-1: Problem Solving as Search

For this question we have used the A* Algorithm to search for the best path to take starting from the start node and to reach the goal node. Each node that ROBBIE should take up from the start position which costs him the least is calculated in this algorithm. It also stores the direction of each move taken up by ROBBIE, its surrounding nodes, its parent node and other such information that is required to make the best decision of choosing a node. This algorithm uses a Tree for its execution. A heuristic function has also been used to calculate the distance.

Algorithm Used to Solve the Search :

For the implementation of this problem, the Algorithm that has been used is as follows -

Step 1.

The position of the starting point is taken from the map as a unique identifier like the coordinates of the start node. This start node is then assigned as the current node and inserted into the open list.

Step 2.

A check for the open list is created which tells us whether the open list is empty or has some value. If the open list is empty at any instance then the output returned is No Path Found. Here we stop the algorithm.

Step 3.

For every node n present inside the open list, the f value of that node n is calculated and the node with the minimum f value is taken up as the new current node. This is then popped out from the open list.

Step 4.

The current node which was popped out from the open list is now inserted into the closed list

Step 5.

The current node that was inserted into the closed list is taken up and checked whether this node is the Goal node in the map. If it is the Goal node then we directly calculate the path from the start node to this current node i.e. Goal node. Here we stop the algorithm.

Step 6.

If the current node was not the Goal node, then we take up this current node and find out all its valid neighbours from the map. The valid neighbours include the consecutive 8 (or less) nodes around the current node n by considering the boundaries of the map. Valid neighbours shouldn’t contain any mountain nodes (represented by X in the map) as well as the nodes just adjacent to the mountain nodes. Valid neighbours should also not contain the nodes that are already present in the open list or the closed list.

Step 7.
Every valid neighbour of the current node is generated and its f,g,h values i.e. total cost,backward cost, forward cost respectively are calculated. These nodes are then inserted inside the open list. (f = g + h)
Step 8.
Return to Step 2

Heuristic Function Used:

The heuristic function used in this algorithm is the Euclidean distance. The heuristic function gives us the best possible distance from the current node n till the goal node G. The Euclidean distance is used as the heuristic function which is basically the distance formula between two points. It is given by the formula -
((y2-y1)^2 + (x2-x1)^2)
Where, x1,y1 are the coordinates of the current node and x2,y2 are the coordinates of the goal node. The values of x1,y1 and x2,y2 can be calculated from the map given to us as the coordinates of the node in the map as the map is taken as a dataframe.
The heuristic function is admissible as it never overestimates the distance i.e. the g value between any two nodes. Here, the heuristic function never overestimates the distance present between the current node n to the goal node G. This can be proved by calculating the heuristic distance when the current node is the goal node. By the formula x1 = x2 and y1 = y2. Hence the heuristic distance comes out to be 0. Hence we can say that it never overestimates.
The heuristic function used is also monotonic. It is because the value of f calculated i.e. the total cost to reach the goal node from the start node/current node will never decrease as we move along the best found path. Hence the heuristic function is monotonic as well.
As it has been proved and implemented in the program, the heuristic function is admissible and monotonic.

Tie Breaking Rules:

A tie will occur when the f values calculated for the nodes inside the open list come out to be equal for two or more nodes. When this condition occurs, we can use a tie breaking rule by considering only the g values of the nodes with equal f values. The node which then has the minimum g value is selected as g is the distance from start node to the current node. In a case where two nodes have the same f value as well as the same g value, in that case the node that was present in the open list in the first order is taken as the current node.

Question 2: First order logic, representation

a. $\forall x(Male(x) \rightarrow \neg(Butcher(x) \wedge Vegetarian(x))$

b. $\forall x \forall y(Male(x) \wedge \neg Butcher(x) \wedge Vegetarian(y) \rightarrow Like(x,y))$

c. $\forall x(Vegetarian(x) \wedge Butcher(x) \rightarrow Female(x))$

d. $\forall x \forall y(Male(x) \wedge Female(y) \wedge Vegetarian(y) \rightarrow \neg(Like(x,y))$

Question 3: Unification

a. P(x, f(x), A, A) and P(y, f(A), y)

A pair of expressions can be unified if the number of terms inside the expressions are the same. For this question as the number of terms present inside the two expressions are not the same hence it cannot be unified. The first expression contains 4 variables while the second expression contains only 3 variables. As they are not the same in number hence the expressions can not be unified.

b. P(x, f(x,y), g(x,w)) and P(A, f(w,B), g(w,x))]

This pair of expressions can be unified as the number of terms inside each expression is the same. Hence we can unify them.

By substituting {x / A} the expression becomes -

P(A, f(A,y), g(A,w)) and P(A, f(w,B), g(w,A))

Then by substituting {w / A} the expression becomes -

P(A, f(A,y), g(A,A)) and P(A, f(A,B), g(A,A))

Finally substituting {y / B} the expression becomes -

P(A, f(A,B), g(A,A)) and P(A, f(A,B), g(A,A))

Hence, the pair of expressions have been unified by following the shown substitutions.

Question 4: Resolution refutation

a.

1. ∀x1 (HIGH-GRADES(x1) → SUCCESSFUL(x1))
2. ∀x2(BRIGHT(x2) ^ WORK-HARD(x2) → HIGH-GRADES(x2))
3. ∀x3(¬BRIGHT(x3) → ¬PASS(x3))
4. ∀x4(¬WORK-HARD(x4) →HAS-FUN(x4))
5. ¬HAS-FUN(JAMES)
6. PASS(JAMES)

b.

Using the concept that (A -> B) i.e. A implies B can be written as (¬A ∨ B)

1. ¬ HIGH-GRADES(x1) ∨ SUCCESSFUL(x1)
2. ¬ BRIGHT(x2) ∨ ¬ WORK-HARD(x2) ∨ HIGH-GRADES(x2)
3. BRIGHT(x3) ∨ ¬PASS(x3)
4. WORK-HARD(x4) ∨ HAS-FUN(x4)
5. ¬HAS-FUN(JAMES)

6. PASS(JAMES)

c.
GOAL: SUCCESSFUL(JAMES)
NEGATED GOAL: 7. ¬ SUCCESSFUL(JAMES)
7 and 1: 7: ¬ SUCCESSFUL(JAMES) 1: ¬ HIGH-GRADES(x1) ∨ SUCCESSFUL(x1)
mgu:{x1|JAMES}
resolvent:8. ¬HIGH-GRADES(JAMES)
8 and 2: 8: ¬HIGH-GRADES(JAMES) 2: ¬ BRIGHT(x2) ∨ ¬ WORK-HARD(x2) ∨ HIGHGRADES(x2)
mgu:{x2|JAMES}
resolvent:9. ¬ BRIGHT(JAMES) ∨ ¬ WORK-HARD(JAMES)
9 and 3: 9. ¬ BRIGHT(JAMES) ∨ ¬ WORK-HARD(JAMES) 3. BRIGHT(x3) ∨ ¬PASS(x3)
mgu:{x3|JAMES}
resolvent:9. ¬ WORK-HARD(JAMES) 3. ¬PASS(JAMES)
9 and 3 and 4: 9. ¬ WORK-HARD(JAMES) 3. ¬PASS(JAMES) 4.WORK-HARD(x4) ∨ HASFUN(x4)
mgu:{x4|JAMES}
resolvent:3. ¬PASS(JAMES) 4.HAS-FUN(JAMES)
3 and 4 and 5: 3. ¬PASS(JAMES) 4.HAS-FUN(JAMES) 5. ¬HAS-FUN(JAMES)
resolvent:3. ¬PASS(JAMES)
3 and 6: 3. ¬PASS(JAMES) 6.PASS(JAMES)
resolvent: NIL
Hence we can see that James is a successful student.
Hence Proved.

Question 5: Resolution refutation

a.
1. ∀x1(BOY(x1) ∨ GIRL(x1) → CHILD(x1))
2. ∀x2 (CHILD(x2) → GET-DOLL(x2) ∨ GET-TRAIN(x2) ∨ GET-COAL(x2))
3. ∀x3 (BOY(x3) → ¬ GET-DOLL(x3))
4. ∀x4 (CHILD(x4) ^ GOOD(x4) → ¬GET-COAL(x4))

b
1. (¬BOY(x1) ^ ¬GIRL(x1) ∨ CHILD(x1))
   This can be split as :
   1.1: ¬ BOY (x1) ∨ CHILD(x1)
   1.2: ¬ GIRL (x1) ∨ CHILD (x1)
2. (¬CHILD(x2) ∨ GET-DOLL(x2) ∨ GET-TRAIN(x2) ∨ GET-COAL(x2))
3. (¬BOY(x3) ∨ ¬ GET-DOLL(x3))
4. (¬CHILD(x4) ∨ ¬ GOOD(x4) ∨ ¬GET-COAL(x4))

c.
GOAL: ∀g ((CHILD (y) => ¬ GET-TRAIN (y)) => (BOY (y) => ¬ GOOD (y))
Converting this to clauses we will get -
(¬ CHILD (y) ∨ ¬ GET-TRAIN (y)) => (¬ BOY (y) ∨ ¬ GOOD (y))
¬ (¬ CHILD (y) ∨ ¬ GET-TRAIN (y)) ∨ (¬ BOY (y) ∨ ¬ GOOD (y))
(CHILD (y) ∧ GET-TRAIN (y)) ∨ (¬ BOY (g) ∨ ¬ GOOD (y))
Negated Goal:
¬ [(CHILD (y) ∧ GET-TRAIN (y)) ∨ (¬ BOY (y) ∨ ¬ GOOD (y))]
¬ CHILD (y) ∨ ¬ GET-TRAIN (y) ∧ BOY (y) ∧ GOOD (y)
Splitting these into 3 as
5.1 {¬ CHILD (y) ∨ ¬ GET-TRAIN (y) }
5.2 { BOY (y) }
5.3 { GOOD(y) }
From 4 and 5.3 we get
(¬CHILD(x4) ∨ ¬ GOOD(x4) ∨ ¬GET-COAL(x4)) GOOD(y)
mgu:{x4|y}

6.resolvent: ¬ CHILD (y) ∨ ¬ GETCOAL (y)
From 6 and 1.1 we get,
¬ CHILD (y) ∨ ¬ GETCOAL (y) ¬ BOY (x1) ∨ CHILD(x1)
mgu:{x1|y}
7. resolvent¬ BOY (y) ∨ ¬ GETCOAL (y)

From 7 and 5.2 we get,
¬ BOY (y) ∨ ¬ GETCOAL (y) BOY (y)
8. resolvent ¬ GETCOAL (y)

From 5.1 and 1.1 we get,
{¬ CHILD (y) ∨ ¬ GET-TRAIN (y) } ¬ BOY (x1) ∨ CHILD(x1)
mgu{x1|y}
9. resolvent ¬ BOY (y) ∨ ¬ GET-TRAIN (y)

From 9 and 5.2 we get,
¬ BOY (y) ∨ ¬ GET-TRAIN (y) BOY (y)
10. resolvent ¬ GET-TRAIN (y)
From 3 and 5.2 we get,
(¬BOY(x3) ∨ ¬ GET-DOLL(x3)) BOY (y)
mgu{x3|y}
11. resolvent ¬ GET-DOLL (y)
From 2 and 8 we get
(¬CHILD(x2) ∨ GET-DOLL(x2) ∨ GET-TRAIN(x2) ∨ GET-COAL(x2)) ¬ GETCOAL(y)
mgu{x2|y}

12. resolvent ¬ CHILD (y) ∨ GET-DOLL (y) ∨ GET-TRAIN (y)
From 12 and 10 we get,
¬ CHILD (y) ∨ GET-DOLL (y) ∨ GET-TRAIN (y) ¬ GET-TRAIN (y)
13 resolvent ¬ CHILD (y) ∨ GET-DOLL (y)
From 13 an 11 we get,
¬ CHILD (y) ∨ GET-DOLL (y) ¬ GET-DOLL (y)
14. resolvent ¬ CHILD (y)

From 14. and 1.1 we get,
¬ CHILD (y) ¬ BOY (x1) ∨ CHILD(x1)
mgu{x1|y}
15. resolvent ¬ BOY (y)

15 and 5.2 we get,
BOY (y) ¬ BOY (y)
Nil (empty)
Hence Proved