ARTIFICIAL INTELLIGENCE

Project 2 Report

Team 67

Zainab Sultan

40-2595

T12

A discussion of the syntax and semantics of the action terms and predicate symbols employed.

action(A): returns true if A is one of the valid actions: up, down, right, left, carry, drop
ids(goalHelper(S),L): performs iterative deepening search on goalHelper(S) with depth limit L. Once call_with_depth_limit returns depth limit exceeded, ids is called recursively with depth limit L+1.
goal(S): checks if S is a variable or not, if it is a variable ids is called with arguments (goalHelper(S),1). If S is not a variable call goalHelper(S).
goalHelper(S): Defines the conditions that should be satisfied in order for a sequence of situations S to be leading to a goal situation from s0. More on implementation details in a subsequent section titled: A description of the predicate goal(S) used to query the KB to generate the plan.
ethansPlace(A,R,C,B,s0,[]): The SSA's base case that defines the next situation after taking an action A from the initial situation s0. R,C represent row number and column number of the agents position after taking action A, B (on board) represents the number of IMF members the agent is carrying after taking action A, the final argument is a list of IMF member's locations that have been dropped / carried. This argument prevents the agent from carrying the same IMF member twice.
ethansPlace(A,R,C,B,S,IMF): defines the next situation after taking action A from any situation S. This is done by recursively calling ethansPlace. More details in the next section.

A discussion of the implementation of the successor-state axioms.

The SSA employed (ethansPlace) keeps information about 3 major things:

1. Location: represented by R,C
2. Number of members currently on truck carried: represented by B.
3. Which members are already carried/ dropped at submarine: represented by the list IMF.

A situation is either the initial situation s0, or a sequence of actions from the initial situation. This recursive definition allows defining ethansPlace recursively, such that the base case is ethansPlace(A,R,C,B,s0,[]), and then finding the information needed about any other situation by tracking it back to the initial situation.

To elaborate on the base case, it allows us to know the state of ethan after performing an action A in the initial situation. This is done by defining a set of constraints for each movement action such that the agent does not move outside the 4x4 grid. Once A is unified with one of the actions, the variables R and C are updated accordingly (access ethan_loc to find coordinates in s0 and determine R and C depending on direction of movement). B is unified with 0 and the IMF list is the empty list as it is impossible to carry in the initial state since no members can be at the same cell as ethan in the initial situation.

If the situation the agent is currently in is not s0, then it can be defined recursively in terms of s0. When ethansPlace(A,R,C,B,S,IMF) is called on a situation S, it first defines a situation S as the result of applying action Ai on situation Si. Then ethansPlace(Ai,Ri,Ci,Bi,Si,IMFi) is called. This recursive call allows reaching the initial state s0 then backtracking. After that Ri, Ci, Bi, and IMFi are obtained which define that status of the agent at situation S. Now the action A unifies with one of the possible 6 actions and a change is reflected on the state of the agent depending on A. The constraints on the move actions prevent the agent from moving out of the grid. In addition, constraints on drop make it only possible to drop at the submarine and constraints on carry only allow carrying if the agent is in the same cell as an IMF member who has not been carried before (coordinates are not found in IMFi list) and there is place on the truck (Bi is less than the maximum carry capacity). The changes that reflect are shown in how R, C, B and IMF change. R and C change with movement actions, however stay the same in case of carry or drop. B and IMF stay constant with movement actions but change if the agent carries or drops (carry sets B to Bi + 1 and adds the position of the member that has just been carried [Ri,Ci] to the IMF list, whereas drop sets B to 0).

The intuition behind this SSA is that either I was at some location and ended up becoming at location R, C as a result of action A (move action) or I was already at R, C but I did not perform a move action, I only carried or dropped.

A description of the predicate goal(S) used to query the KB to generate the plan.

goal(S) checks if S is a variable or not. If it is not a variable it calls goalHelper(S). If S is a variable, iterative deepening search is performed to find a solution for goalHelper(S) by calling ids(goalHelper(S),1). goalHelper(S) is a helper predicate that finds a plan and can also check if a plan is valid. Implementing goalTrue(S) is at heart defining the conditions that satisfy the goal. The goal depends on the number of IMF members on the grid in the initial situation, because if there are 0 IMF members on the grid in s0, then ethans goal is to reach the submarine.

If there is more than one IMF member on the grid, define S = result(A, Sp) and call ethansPlace(A,X,Y,0,Sp,IMF). Where A can only be a drop, and the location of the agent in situation Sp must be the submarine (X and Y equal submarine's location). Furthermore, the IMF list must contain the same elements as the list that is in the KB's members_loc. In order to check if they contain the same elements, the lists are sorted and checked if they unify. The sorting step allows the members to be carried in any order and still the lists would unify if it contains all locations of IMF members. B must be 0 as well since the last action should be dropping all members into the submarine so that means that the truck must be empty.

If there are 0 IMF members (members_loc list length == 0) then the goal is to go to the submarine. This means calling ethansPlace(A,X,Y,0,Sp,[]) with the constraint that X and Y must be equal to submarine coordinates (constraint = submarine(X,Y)).

Running examples from the implementation.

Example 1:

ethan_loc(0,0).

members_loc([[1,1],[1,2]]).

submarine(0,2).

capacity(1).

Query: goal(S)

Output:

```
S = result(drop, result(up, result(carry, result(down, result(drop, result(right, result(up,
result(carry, result(right, result(down, s0)))))))))) ;
S = result(drop, result(up, result(carry, result(down, result(drop, result(up, result(right,
result(carry, result(right, result(down, s0)))))))))) ;
S = result(drop, result(up, result(carry, result(down, result(drop, result(right, result(up,
result(carry, result(down, result(right, s0)))))))))) ;
S = result(drop, result(up, result(carry, result(down, result(drop, result(up, result(right,
result(carry, result(down, result(right, s0)))))))))) ;
```

S = result(drop, result(up, result(carry, result(down, result(drop, result(right, result(up, result(carry, result(right, result(down, s0))))))))))

Query:

goal(result(drop, result(up, result(carry, result(down,result(drop, result(up, result(right, result(carry,result(down, result(right, s0))))))))))).

Output:

true.

Example 2: (no IMF members)

ethan_loc(0,0).

members_loc([]).

submarine(0,2).

capacity(1).

Query: goal(S)

Output:

S = result(right, result(right, s0)) ;

S = result(right, result(right, s0)) ;

S = result(right, result(right, s0)) ;

S = result(right, result(right, result(up, result(down, s0)))) ;

S = result(right, result(up, result(right, result(down, s0))))

Example 3:

ethan_loc(0,0).

members_loc([[1,1],[0,3]]).

submarine(0,2).

capacity(1).

Query: goal(S).

Output:

S = result(drop, result(left, result(carry, result(right, result(drop, result(right, result(up, result(carry, result(right, result(down, s0)))))))))) ;

S = result(drop, result(left, result(carry, result(right, result(drop, result(up, result(right, result(carry, result(right, result(down, s0)))))))))) ;
S = result(drop, result(left, result(carry, result(right, result(drop, result(right, result(up, result(carry, result(down, result(right, s0)))))))))) ;
S = result(drop, result(left, result(carry, result(right, result(drop, result(up, result(right, result(carry, result(down, result(right, s0)))))))))) .