

Action terms and Symbol Predicates

- directions (dir, dx, dy) is an action term which defines the shift in the direction of x,y with dir can be either {left, right,up,down}

directions(up,-1,0).

directions(down,1,0).

directions(left,0,-1).

directions(right,0,1).

- isValid(X,Y) returns true if X,Y are valid position in the grid.
- contains(Item,List) true if List contains Item.
- skip(X,Y,L,Rem) removes the location X,Y from the List L and returns the remaining list in Rem.
- canCarry(X,Y,Hostages) return true if there is a hostage in X,Y in the list Hostages.
- newUndroppedHostages(OriginalHostages,CarriedHostages,UndroppedHostages). true when UndroppedHostages is the set of hostages in OriginalHostages but not in CarriedHostages.
- base(X,Y,CarriedHostage,UndroppedHostages ,C) returns true if X,Y is Neo's Location and carriedHostages is [] and UndroppedHostages is the original list of hostages, C is the full Capacity

Successor-State Axioms

- haveCarriedHostages(S) is a successor state axiom that returns true if Neo has carried hostages if the sequence of actions in state S contains at least one carry not followed by a drop. If $S = \text{result}(a, S')$ we check if action **a** is carry or **a** is not drop and check that S' haveCarriedHostages.
- neoArrived(X, Y, CarriedHostages, UndroppedHostages, Capacity, S) Is true if Neo can reach X, Y in state S starting from S_0 and carrying all Hostages in CarriedHostages with remaining undropped hostages in UndroppedHostages list In all cases except base case we check that this state is not the initial state.

- Base case where X,Y are Neo's start location, CarriedHostages is empty list, and UndroppedHostages is the whole original list of hostages, Capacity is the original Capacity that Neo can hold, S is the s0.
- If $S = \text{result}(\text{drop}, S')$ we check that X,Y is the location of the booth, Capacity is the original capacity, and check if Neo have carried Hostages in state S then we check $\text{neoArrive}(X, Y, \text{CarriedHostages}, _, C, S')$, We make UndroppedHostages to be the list of hostages that are in Original List of hostages but not in CarriedHostages.
- If $S = \text{result}(\text{carry}, S')$ we check if $\text{neoArrived}(X, Y, \text{CarriedHostages}', \text{UndroppedHostages}, \text{Capacity}', S')$ and $\text{Capacity}' > 0$ and set $\text{Capacity} = \text{Capacity}' - 1$ then we get the list of CarriedHostages by adding [X,Y] to CarriedHostages' and check if this list contains a hostage at location [X,Y].
- If $S = \text{result}(a, S')$ where **a** is either {left,right,up,down}. We check if this action is valid (i.e. will lead to moving to a valid location in the grid) then we check $\text{neoArrived}(\text{NewX}, \text{NewY}, \text{CarriedHostages}, \text{UndroppedHostages}, \text{Capacity}, S')$ where NewX is X-Dx, NewY is Y-Dy.

Goal predicate

A state S is goal if Neo can arrive $\text{neoArrived}(X, Y, L, [], S)$ where X,Y is the telephone booth location, L is the original List of hostages (i.e. in state S all hostages have been carried), and the UndroppedHostages in state is the empty list (i.e. there are no UndroppedHostages in state S) and S ends with drop.

Running Examples

- We will run our code on the following knowledge base with depth limit of 13.
`grid(4,4).`
`neo_loc(0,0).`
`hostages_loc([[1,1],[1,2]]).`

booth(0,2).

capacity(1).

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(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(right,result(down,s0))))))))))
 - S =
result(drop,result(up,result(carry,result(down,result(drop,result(up,result(right,result(carry,result(down,result(right,s0))))))))))
- We will run our code on the following knowledge base with depth limit of 10.
grid(4,4).
neo_loc(0,0).
hostages_loc([[1,1],[1,2]]).
booth(0,2).
capacity(2).

Output:

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

