AI Project 2

1 A brief discussion of the problem

In this project, we are aiming to implement a simplified logic-based version of the Coast Guard agent. This agent reasons using the situation calculus. The project will be implemented in Prolog. In this grid, there is only one station, exactly two ships, each of them has exactly one passenger who does not expire with time. There are no black boxes to retrieve. The agent's capacity can be either 1 or 2. The grid size is 3×3 or 4×4.

2 Prolog Files

This project contains two Prolog files:

- **KB.pl :** contains a sample Knowledge Base with the initial state of the world.
- **CG.pl :** contains our implementation.

3 KB.pl

This file contains five predicates :-

- $grid(N, M)$ **:** a predicate represents the grid dimensions.
- $agent\_loc(X, Y)$ **:** a predicate represents the agent location.
- $ships\_loc([[SH1x, SH1y], [SH2x, SH2y]])$ **:** a predicate represents all ships in a list of lists,each list contains x,y positions.
- $station(Sx, Sy)$ **:** a predicate represents a station location.
- $capacity(C)$ **:** a predicate represents the guard capacity.

This file can be modified such that we can change the grid dimensions, agent initial location, station location, guard capacity,number of ships and their locations.

4 CG.pl

Initially, this file consults KB.pl to be able to use this knowledge base to initialize our matrix elements.
This file contains three main predicates :-

- $goal(S)$ **:** this predicate generates a plan using the agent's KB, and the result will be a situation described as the result of doing sequence of actions from the initial state s0. It calls "ids" predicate with a state contains an empty set of ships and zero carried passengers initially.

- $state(GuardX, GuardY, CurrentCapacity, Ships, S)$ **:** this predicate infers a new state using successor state axiom on the current situation. It checks if a specific operator is valid and then create a new state with this operator. GuardX, GuardY are the guard locations, CurrentCapacity represents number of passengers that are carried by the guard. Ships is a list of lists contains the remaining ships and their locations in that state. S represents the current situation. This predicate has two cases. The first case is to initialize the gurad location, the ships list and make current capacity equals zero. The second case is recusively infers a new state using successor state axiom.
  Now, we will discuss set of operators/actions that can be made to expand the current state :

  - **pickup :** First we need to check if the guard is on the same cell of a ship, and this check will be done using *isShip* predicate. Then, we need to see if the guard's current capacity is less than the maximum capacity, and this is done by *haveCapacity* predicate. If the previous predictes are true, then we will perform the pickup action, which is done by *pickup* predicate. This action will increase the current capacity by one and remove this ship from list of ships. finally, we will use *copy* predicate to copy the current location to the new created state, and the action will be equal to pickup.
  - **drop :** First we need to check if the guard is in the same cell with a station, and this is done by *isStation* predicate. Then, we need to check

if the guard is carrying at least one passenger using *havePassengers* predicate. If the previous predicates are true, then we will perform drop action. This action will reset the current capacity to be zero. After that we will copy the guard location and the list of ships to the new state, and the action will be equal to drop.

  - **right :** First we need to check if moving to right is a valid action, and this is done by *validRight* predicate. If yes, then we will use *moveRight* predicate to update the guard's location, and then copy the capacity and ships to the new state, and this action will be right.
  - **left :** First we need to check if moving to left is a valid action, and this is done by $validLeft$ predicate. If yes, then we will use $moveLeft$ predicate to update the guard's location, and then copy the capacity and ships to the new state, and this action will be left
  - **down :** First we need to check if moving to down is a valid action, and this is done by *validDown* predicate. If yes, then we will use *moveDown* predicate to update the guard's location, and then copy the capacity and ships to the new state, and this action will be down.
  - **up :** First we need to check if moving to up is a valid action, and this is done by *validUp* predicate. If yes, then we will use *moveUp* predicate to update the guard's location, and then copy the capacity and ships to the new state, and this action will be up.

- $ids(X, L)$ **:** this predicate uses *call_with_depth_limited* built-in predicate to do iterative deepening search when solving for $goal(S)$. This predicate prevents the program from running forever.

Now, let's discuss some helper predicates we made:

- $isShip(Xloc, YLoc, Ships)$ **:** this predicate checks if the list of x, y locations is a member of ships list.

- $havaCapacity(CurrentCapacity)$ **:** this predicate checks if the current capacity is less than the maximum.

- $pickup(XLoc, YLoc, Capacity, NewCapacity, Ships, RemainingShips)$: this predicate perform pickup action by increasing the capacity by one and then remove the current ship from the ships list using removeShip predicate.

- $isStation(XLoc, YLoc)$: this predicate checks if the current location represents a station or not.

- $havePassengers(Capacity)$: this predicate checks if the current capacity is greater than zero, which means that the guard is carrying at least one passenger.

- $drop(NewCapacity)$: this predicate perform drop action by resetting the capacity to be zero.

- $removeShip(Shx1, Sh1y, [H, T], T)$: this predicate removes ship that is located at (Shx1,Shy1) from list of ships.

- $validRight(GuardY1)$: this predicate checks if the current y location is less than width of the grid, so the guard can move to right.

- $validLeft(GuardY1)$: this predicate checks if the current y location is greater than zero, so the guard can move to left.

- $validDown(GuardX1)$: this predicate checks if the current x location is less than the height of the grid, so the guard can move down

- $validUp(GuardX1)$: this predicate checks if the current x location is greater than zero, so the guard can move up.

- $copy(In, Out)$: this predicate copies value of In variable to be in Out variable.

- $moveRight(GuardX1, GuardY1, GuardX, GuardY)$: this predicate updates the y value by adding one to it and leave the x to be the same.

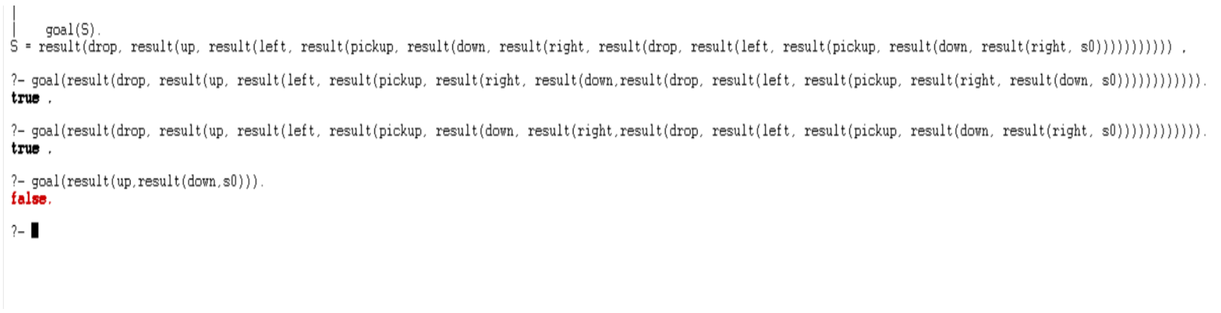
- $moveLeft(GuardX1, GuardY1, GuardX, GuardY)$: this predicate updates the y value by subtracting one from it and leave the x to be the same.

- $moveDown(GuardX1, GuardY1, GuardX, GuardY)$ **:** this predicate updates the x value by adding one to it and leave the y to be the same.
- $moveUp(GuardX1, GuardY1, GuardX, GuardY)$ **:** this predicate updates the x value by subtracting one from it and leave the y to be the same.

5 Examples

$$
\begin{aligned}
&grid(3,3).\\
&agent\_loc(0,1).\\
&ships\_loc([[2,2],[1,2]]).\\
&station(1,1).\\
&capacity(1).
\end{aligned}
$$

Using the above knowledge base, we run some queries to test the implementation. The below figure shows four queries and their outputs. All of these queries run in less than two seconds.



```
|
|    goal(S).
S = result(drop, result(up, result(left, result(pickup, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0))))))))))) .

?- goal(result(drop, result(up, result(left, result(pickup, result(right, result(down,result(drop, result(left, result(pickup, result(right, result(down, s0)))))))))))).
true .

?- goal(result(drop, result(up, result(left, result(pickup, result(down, result(right,result(drop, result(left, result(pickup, result(down, result(right, s0)))))))))))).
true .

?- goal(result(up,result(down,s0))).
false.

?- █
```

Figure 1: A screenshot for the queries and their outputs