AI Project 2 Report

Rowan Amgad and Maryam ElOraby

22 December 2022

1. **Overview**
   Our implementation uses backtracking in order to reach a solution for the coast guard problem by recursively calling the following fluents and predicates.

2. **Successor State axiom**
   In our implementation, we implemented the successor state axiom using the **`cgAction(CgX, CgY, CurrShips, CurrCarriedCount, result(A, S))`** fluent.
   This fluent runs recursively starting at a specific state S till it reaches the initial state **s0** and then forms the solution by backtracking through the recursion tree. It has the following parameters:
   - CgX, CgY: Location of the coast guard rescue boat after performing action A.
   - CurrShips: List of remaining ships with unsaved passengers after performing action A.
   - CurrCarriedCount: Number of carried passengers on the coast guard rescue boat after performing action A.
   - result(A,S): The representation of the current situation of the agent after performing action A in situation S.
     Inside this fluent, all movement predicates, drop predicate and pickup predicate are called and an or-ing operation is performed on their returned results in order to get the action A that led to result(A,S). Then, the predicate calls itself recursively till reaching the base case.
     Base case:
     `cgAction(CgX,CgY,PrevShips,0,result(A,s0))`

3. **Helper Fluents**
   I. **`goal(S)`**: This is the predicate that shall be called in order to find a solution to the coast guard problem without running infinitely by getting stuck down a recursive branch, as it uses the predicate **`ids(X, L)`** to solve the problem, where X is S and L is 1.

II. **`ids(X, L)`**: This predicate solves the coast guard problem using the iterative deepening algorithm by calling the predicate **`goal2(X)`** inside prolog's predefined predicate **`call_with_depth_limit(D, L, R)`**.

III. **`goal2(S)`**: This is the predicate that actually finds a solution to the coast guard problem. It takes the current situation of the agent as a successor state axiom as a parameter and calls **`cgAction(CgX, CgY, [], 0, S)`** fluent and checking for **`station(CgX, CgY)`** in the knowledge base, where CgX and CgY are coordinates of the current location of the coast guard. **`station(CgX, CgY)`** ensures that at a goal state the coast guard is standing at a cell containing a station.

4. **Helper Predicates**

A) Movement Predicates

I. **`up(A,PrevX,CgY,CgX,CgY,PrevShips,PrevCarriedCount,PrevCarriedCount,PrevShips).`**
This predicate was used to move the coast guard from its current position upwards (*x*-axis). Only the *x*-coordinate of the coast guard changes if it doesn't exceed the grid bound, and all of the other world's parameters remain constant.
- A: To name and return the action as "up".
- PrevX: *x*-coordinate of the coast guard in the grid the step before performing up. *y-coordinate doesn't change*.
- CgX, CgY: coordinates of the coast guard in the grid after performing the up action.
- PrevShips: A list containing the ships we have in the grid.
- PrevCarriedCount: number of passengers the coast guard is carrying.

PrevShips and PrevCarriedCount remain unchanged.

II. **`down(A,PrevX,CgY,CgX,CgY,PrevShips,PrevCarriedCount,PrevCarriedCount,PrevShips).`**
This predicate was used to move the coast guard from its current position downwards (*x*-axis). Only the *x*-coordinate of the coast guard changes if it doesn't exceed the grid bound, and all of the other world's parameters remain constant.

- A: To name and return the action as “down”.

The rest of the parameters are the same as up/9.

III. **`left(A,CgX,PrevY,CgX,CgY,PrevShips,PrevCarriedCount,PrevCarriedCount,PrevShips).`**

This predicate was used to move the coast guard from its current position to the left (*y*-axis). Only the *y*-coordinate of the coast guard changes if it doesn’t exceed the grid bound, and all of the other world’s parameters remain constant.

- A: To name and return the action as “left”.
- PrevY: *y*-coordinate of the coast guard in the grid the step before performing up. *x-coordinate doesn’t change*.
- CgX, CgY: coordinates of the coast guard in the grid after performing the left action.

PrevShips and PrevCarriedCount remain unchanged.

IV. **`right(A,CgX,PrevY,CgX,CgY,PrevShips,PrevCarriedCount,PrevCarriedCount,PrevShips).`**

This predicate was used to move the coast guard from its current position to the right (*y*-axis). Only the *y*-coordinate of the coast guard changes if it doesn’t exceed the grid bound, and all of the other world’s parameters remain constant.

- A: To name and return the action as “right”.

The rest of the parameters are the same as above.

B) Pick-up Predicates

pickup(A,PrevX,PrevY,CgX,CgY,CurrCarriedCount,CurrShips,PrevShips,PrevCarriedCount)

This predicate was used to find the number of carried passengers and their locations after performing action A, where A=“pickup” in this case and *the rest of the parameters are the same as up/9.* In our implementation, there are 3 variations of this predicate:

I. **`pickup(A,CgX,CgY,CgX,CgY,CurrCarriedCount,[],[[CgX,CgY]],PrevCarriedCount)`**

In this case, only one passenger exists in the grid to be picked up and accordingly, PrevShips=[] and CurrShips will contain the location of this ship on the grid.

II. **`pickup(A,CgX,CgY,CgX,CgY,1,[Sh2],[[CgX,CgY],Sh2],0)`**

In this case, the coast guard picked up the passenger in the first ship in the CurrShips list and another one is still left to be picked up which means this was the first passenger to be picked. Accordingly, PrevCarriedCount will be 0, CurrCarriedCount will be 1 to represent this passenger that's been picked up, CurrShips will only contain the location of the remaining ship and PrevShips will contain the locations of all the grid's ships.

III. **`pickup(A,CgX,CgY,CgX,CgY,1,[Sh2],[Sh2,[CgX,CgY]],0)`**

In this case, the coast guard picked up the passenger in the second ship in the CurrShips list and another one is still left to be picked up which means this was the first passenger to be picked. Accordingly, PrevCarriedCount will be 0, CurrCarriedCount will be 1 to represent this passenger that's been picked up, CurrShips will only contain the location of the remaining ship and PrevShips will contain the locations of all the grid's ships

C) Drop Predicates

drop(A,PrevX,PrevY,CgX,CgY,CurrCarriedCount,CurrShips,PrevShips,PrevCarriedCount)

This predicate was used to find the number of carried passengers and their locations after performing action A, where A="drop". *The rest of the parameters are the same as up/9.* In our implementation, there are also 3 variations of this predicate:

I. **`drop(A,CgX,CgY,CgX,CgY, 0,[],[],2)`**

In this case, the coast guard is standing at a station cell and dropped 2 passengers, leaving the grid with no other passengers left to be picked up and hence, reaching a goal state.

III. `drop(A,CgX,CgY,CgX,CgY, 0,[],[],1)`

In this case, the grid contained only 1 ship whose passengers was previously picked up and currently got dropped at a station, hence reaching a goal state.

III. `drop(A,CgX,CgY,CgX,CgY, 0,[Sh1],[Sh1],1)`

In this case, the grid contains 2 ships and the coast guard previously picked up one of them and currently dropped them at a station, leaving another passenger on a ship in the grid that needs to be saved.

5. Examples

Please note that we used call_time/2 to calculate how long it takes to run a query.

- KB 1:

```
grid(3,3).
agent_loc(0,1).
ships_loc([[2,2],[1,2]]).
station(1,1).
capacity(1)
```

Query 1:

```
goal(S).
```

Solution:

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

The output of `call_time(goal(S),Time_s)`:

`Time_s = time{cpu:8.109375, inferences:109642209, wall:8.450376987457275}`

showing that this query took about **8s**.

Another solution:

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

```
T_s = time{cpu:0.0, inferences:393,
wall:5.888938903808594e-5}
```

Query 2:

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

Solution:

```
true.
```

```
T_s = time{cpu:0.0, inferences:5537,
wall:0.0006318092346191406}
```

Query 3:

```
goal(result(up,result(down,s0))).
```

Solution:

```
false.
```

- KB 2:

```
grid(4,4).
agent_loc(0,2).
ships_loc([[1,2], [3,2]]).
station(1,1).
capacity(2).
```

Query 1:

```
goal(S).
```

Solution:

```
S = result(drop, result(up, result(up, result(left,
result(pickup, result(down, result(down, result(pickup,
result(down, s0))))))))) ;
```

```
T_s = time{cpu:1.03125, inferences:13921616,
wall:1.0258359909057617}
```

Another solution:

```
S = result(down, result(up, result(drop, result(left,
result(up, result(up, result(pickup, result(down, result(down,
result(pickup, result(down, s0)))))))))))
```

```
T_s = time{cpu:0.015625, inferences:268585,
wall:0.018970012664794922}
```

Query 2:

```
goal(result(drop, result(left, result(up, result(up, result(down,
result(down, result(pickup, result(down, s0))))))))).
```

Solution:

```
false.
```