Fluent used:

- coast_guard(Row, Col, Cap, Ships, S)
    - What does it mean:
      It represents the state of the coast guard world at situation S, the state of the coast guard world is represented by 4 elements: the row and column in the grid where the coast guard ship is currently standing, the capacity of the coast guard ship - how many passengers is the coast guard ship holding, and lastly the ships with passengers that still need to be rescued.
    - Description of the arguments:
        - Row: the row where the coast guard ship is in; integer
        - Col: the column where the coast guard ship is in; integer
        - Cap: the capacity of the coast guard ship - how many passengers is the coast guard ship holding; integer
        - Ships: the ships with passengers on them that still need to be saved; a list of lists where each internal list represents a ship: the first element is the row where the ship is at and the second element is the column where the ship is at

Problem setup:

Before we could discuss the successor state, we need to discuss how we set up our problem, we define some actions that are allowed for our agent as well as some constraints for each of them, these actions are represented as predicates and the constraints as checks that the predicate does not hold without. The actions we are allowing are: up, down, left, right, drop and pickup.

For each of these actions' predicates we have 8 arguments: the first four are the row, column, capacity and ships after the agent has performed the action, and the second four are the same arguments but after the action have performed the action.

- Movement actions (up, down, left and right):
  The only check we care about in the movement actions is that the agent was performing this movement action when it was at a valid cell - we define a valid cell to be a cell within the grid limits.
- Drop action:
  We have to check for a couple of things when dropping: first, whether the agent is at a station or not and second, that the capacity of the agent after performing the action was reset to 0
- Pickup action:
  This is the action with most checks: we check that the capacity of the agent after performing the action did not exceed the maximum capacity allowed for the agent, that the agent is at the location of a ship, and lastly we check that the ship we picked up passengers from is no longer considered for saving.

Successor-state:
Since we have only one fluent in our implementation, we only have one successor-state axiom that we need to implement. Our implementation is fairly simple, the successor-state holds if we are at the initial conditions for the problem, or if there exists some action that can move us from the current situation to a previous one.
The code implementation for this is as following:

```
coast_guard(Row, Col, 0, Ships, s0):-
  agent_loc(Row, Col),
  ships_loc(Ships).
coast_guard(Row, Col, Cap, Ships, result(A, S)):-
  action(A, Row, Col, Cap, Ships, PRow, PCol, PCap, PShips),
  coast_guard(PRow, PCol, PCap, PShips, S).
```

agent_loc is a predicate that hold if and only if the row and column passed are the initial row and column for the agent, the initial capacity of the agent is zero so we pass a zero for the capacity argument, ships_loc is a predicate that holds if and only if the ships list passed matches the list of ships the problem starts with and lastly we have s0 as a constant that represents the initial situation.

Test cases and output:

- With KB
    - TC1:
        - Query: goal(S).
        - Output: S = result(drop, result(up, result(left, result(pickup, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0)))))))))))
        - Execution time: 1845ms
    - TC2:
        - Query: goal(result(drop, result(up, result(left, result(pickup, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0)))))))))))).
        - Output: true
        - Execution time: 0ms
    - TC5:
        - Query: goal(S).
        - Output: S = result(drop, result(up, result(left, result(pickup, result(down, result(up, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0)))))))))))))
        - Execution time: 11799ms
    - TC4:
        - Query: goal(result(drop, result(up, result(left, result(pickup, result(down, result(up, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0)))))))))))))).

        - Output: true
        - Execution time: 0ms
    - TC5:
        - Query: goal(s0).
        - Result: false.
        - Execution time: 0ms
    - TC6:
        - Query: goal(Result(drop, s0)).
        - Result: false.
        - Execution time: 0ms
- With KB2
    - TC1:
        - Query: goal(S).
        - Output: S = result(drop, result(left, result(pickup, result(up, result(up, result(pickup, result(down, result(down, result(down, s0)))))))))
        - Execution time: 689ms
    - TC2:
        - Query: goal(result(drop, result(left, result(pickup, result(up, result(up, result(pickup, result(down, result(down, result(down, s0)))))))))).
        - Output: true
        - Execution time: 0ms
    - TC3:
        - Query: goal(S).
        - Output: S = result(drop, result(up, result(left, result(down, result(pickup, result(up, result(up, result(pickup, result(down, result(down, result(down, s0)))))))))))
        - Execution time: 8286ms
    - TC4:
        - Query: goal(result(drop, result(up, result(left, result(down, result(pickup, result(up, result(up, result(pickup, result(down, result(down, result(down, s0)))))))))))).
        - Output: true
        - Execution time: 0ms
    - TC5:
        - Query: goal(s0).
        - Result: false.
        - Execution time: 0ms
    - TC6:
        - Query: goal(Result(drop, s0)).
        - Result: false.
        - Execution time: 0ms

Material that helped:

1. Measuring execution time in SWI-Prolog: https://coderwall.com/p/laduzw/how-to-measure-execution-time-in-swi-prolog