German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

Introduction to Artificial Intelligence, Winter Term 2022
Project 2: Coast Guard Plan

Due December 23rd by 23:59

1. **Project Description** In this project you will be implementing a simplified logic-based version of the Coast Guard agent. This agent reasons using the situation calculus. The project will be implemented in *Prolog*.

   We make the following simplifying assumptions.

   a) There are no black boxes to retrieve.
   b) There is only one station and a maximum of two ships.
   c) Each ship has exactly one passenger who does not expire with time.
   d) The agent's capacity can be either 1 or 2.
   e) The grid size is 3×3 or 4×4.

   To implement this agent correctly, you need to follow the following steps:

   a) You will find on the CMS a prolog file `KB.pl` containing a sample Knowledge Base with the initial state of the world. Do not add anything to this file, and create a new Prolog file `CG.pl` in which you will write your implementation. `CG.pl` must be in the same directory in which `KB.pl` lies. Import `KB.pl` at the beginning of `CG.pl` using

   ```
   :- include('KB.pl').
   ```

   b) Come up with fluents (predicates whose denotations change across situations) to describe the state of the world. For each fluent, write a successor-state axiom in `CG.pl`. It is recommended to have a maximum of two fluents and, hence, a maximum of two successor-state axioms. Whenever possible, it is preferable to use built-in predicates rather than defining your own. You are free to write any helper predicates you need.
   c) Write a predicate `goal(S)` and use it to query the agent's KB to generate a plan that the Coast Guard can follow to collect all the ship passengers and drop them at the station. You are not required to return an optimal plan. The result of the query should be a situation described as the result of doing some sequence of actions from the initial situation s_0 (as shown in the examples in the next section).

   Important Note: You might write your successor state axioms correctly, yet when you query your KB, your program might run forever. This is because Prolog uses DFS to implement backward chaining, and we know that DFS is incomplete. To solve this issue, consider using the built-in predicate `call_with_depth_limit` (`http://www.swi-prolog.org/pldoc/man?predicate=call_with_depth_limit/3`). This predicate does depth limited search to backchain on the query provided as the first argument

of the predicate. You can use this built-in predicate to implement another predicate to do iterative deepening search when solving for `goal(S)`. In this way, you will guarantee that you will reach a solution (if there is one) since IDS is complete. One way to implement IDS in prolog (you can write it in any way you like):

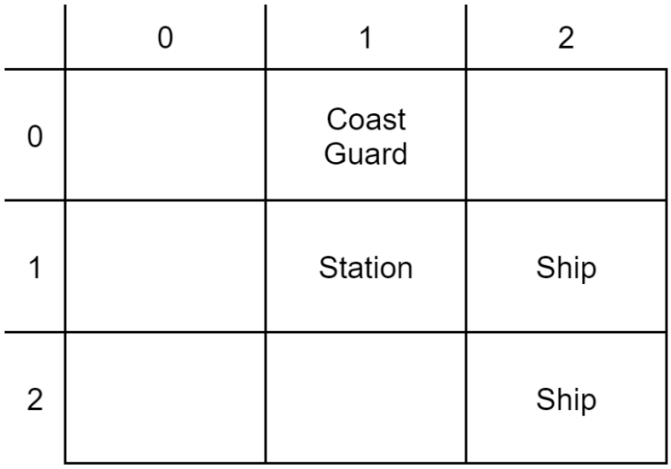
```
ids(X,L):-
(call_with_depth_limit(myPredicate(X),L,R), number(R));
(call_with_depth_limit(myPredicate(X),L,R), R=depth_limit_exceeded,
                                            L1 is L+1, ids(X,L1)).
```

2. Example:

- **Knowledge base (KB.pl)**:

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

- **Grid represented by that KB**



- **Query1**: goal(S).
- **Output1**:S = result(drop, result(up, result(left, result(pickup, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0))))))))))).
- **Query2**: goal(result(drop, result(up, result(left, result(pickup, result(down, result(right, result(drop, result(left, result(pickup, result(down, result(right, s0))))))))))).
- **Output2**: true.
- **Query3**: goal(result(drop, result(up, result(left, result(pickup, result(right, result(down, result(drop, result(left, result(pickup, result(right, result(down, s0))))))))))).
- **Output3**: true.
- **Query4**: goal(result(up,result(down,s0))).
- Output4: false.

3. **Groups:** Same teams as Project 1. If you wish to change your team, you can send your TA an email.
4. **Deliverables**
   a) Source Code: `KB.pl` and `CG.pl`
   b) Project Report, including the following:
      - fluents used in the project and what they mean and a description of their arguments.
      - an explanation of the successor state axioms you implemented.
      - some test cases, their outputs and how long it takes to run.
5. **Submission**

   Source code and Project Report On-line submission by December 23rd at 23:59. Submission form: `https://forms.gle/fTe3hyLGCbG44hJK7`. You should submit a .zip file containing `KB.pl` and `CG.pl`

   Brainstorming Session. In tutorials.