German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

Introduction to Artificial Intelligence, Winter Term 2024
Project 2: Watersort Situation Calculus

Due November 21st by 23:59

1. Project Description In this project you will be implementing a logic-based version of the Watersort agent from the first project for a simpler version of the problem. This agent reasons using the situation calculus. The project will be implemented in *Prolog*.

We make the following simplifying assumptions.

a) Your goal is to have one color in each bottle, and the bottle should be completely filled with the color.

b) There are only 3 bottles.

c) There are only 2 colors.

d) There are only 2 layers in each bottle.

e) You can only pour one layer at a time.

f) The only action is pour(i, j), pour from bottle i into bottle j.

g) Bottle indexing starts from 1.

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

a) You will find a Prolog file, `KB.pl`, on the CMS 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 `Watersort.pl` in which you will write your implementation. `Watersort.pl` must be in the same directory in which `KB.pl` lies. Import `KB.pl` at the beginning of `Watersort.pl` using

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

b) Come up with fluents (predicate symbols whose denotations change across situations) to describe the state of the world. For each fluent, write a successor-state axiom in `Watersort.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 can be followed to have one color in each bottle and the bottle should be filled with the same color. 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 back-chain 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 iterative deepening search is complete.

Here is 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)**:

```
bottle1(b,r).
bottle2(b,r).
bottle3(e,e).
```

- **Query1**: goal(S).
- **Output1**: S = result(pour(2, 1), result(pour(2, 3), result(pour(1, 3), s0))).
- **Query2**: goal(result(pour(1, 2), result(pour(1, 3), result(pour(2, 3), s0)))).
- **Output2**: true.
- **Query3:** goal(result(pour(2, 1), result(pour(1, 3), result(pour(2, 3), s0)))).
- **Output3:** true .
- **Query4**: goal(result(pour(2, 1), result(pour(1, 2), result(pour(2, 1), result(pour(1, 2), result(pour(2, 1), result(pour(1, 3), result(pour(2, 3), s0)))))))).
- **Output4**: false.
- **Query5**: goal(result(pour(1, 2), result(pour(2, 3), s0))).
- **Output5:** false.
- **Query6**: goal(result(pour(1, 3),result(2, 3), s0)).
- **Output6:** 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 `Watersort.pl`

   b) Project Report, including the following:

   - Fluents used in the project, 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 November 21st at 23:59.

   Submission form. You should submit a .zip file containing `KB.pl` and `Watersort.pl` using the following form.

   `https://docs.google.com/forms/d/e/1FAIpQLScWebOrg_aa-KqkEb5wrbcStwxLaV5lag2jMtRBSh352F-0ag/viewform`.

   Brainstorming Session. In tutorials.