CSEN906

Introduction to Artificial Intelligence

Winter Term 2017

21.11.2017

Project 2: ∃ a ∃ s[Escaped(R2D2,Result(a,s))]

Report

Yara Yehia	31-1022	T12
Esraa Salah	31-6542	T12
Hagar Yasser	31-3122	T10

Table of Contents

Problem Description

The goal of the logic-based agent is to activate the teleportal on a grid and escape through it. The teleportal is activated if and only if all rocks on the grid are positioned on pressure pads. The agent can push one rock at a time if it is not hindered by another rock or an obstacle. The agent starts on a blank cell and cannot move through rocks or obstacles. The logic agent starts out with a knowledge base (KB) of facts describing the grid; the grid is defined by its dimensions, the location(s) of the rocks, obstacles, pressure pads and teleportal. The KB also contains initially the agent position. Given a certain goal state (a set of logical sentences), the agent mission is to derive these sentences by incrementally applying given inference rules on its KB.

Implementation

The agent is mainly programmed in SWI Prolog along with the clpfd library. The grid instances, the initial states and the goal states are generated by a java program.

GenGrid Implementation

- Handled by the java program inside src/Main.java.
- Signature: public static void GenGrid(Grid g)
- Given a grid g as input, GenGrid(g) writes the initial state, as Prolog facts, along with the goal state in a *.pl* file.
- The method calls helper instance methods *writePrologFacts* and *writePrologQuery*.
- *Method writePrologFacts*() writes in a file "*initial-state-Sol<n>*" the facts of the initial state of the problem. It includes:
    - Dimensions of the grid.
    - Positions of obstacles, rocks, pressure pads.
    - Position of the teleportal.
    - Initial position of the agent.
- *Method writePrologQuery*() writes in a file "*query-Sol<n>*" a predicate "query(S):-"which is satisfied if the state (S) is the goal state. The goal state ensures that the agent is positioned on the teleportal and every pressure pad position contains a rock; these positions are known beforehand and hence we let the java program hard code them into the predicate body.
- The *main method* in Main.java, creates a grid object by parsing a specified file (check one of the files for the required grid format) in the folder *grid-tests*, then calls GenGrid with each grid as a parameter.

Terms & Helper Predicates

- Facts
    - dimensions(m,n).
        - The grid dimensions are *m* rows and *n* columns.
    - obstacle(R,C).
        - Cell in row *R* and column *C* contains an obstacle
    - obstacle(-1,-1)
        - To avoid error throwing in Prolog of non-existent predicate if the grid contains no obstacles.
- Predicates
    - valid(R,C)
        - Given that dimensions(m,n) holds:
          R is between 0 and m and C is between 0 and n.

- next_cell(A, R, C, RNext, CNext)
  Four definitions, one for each action.
    - Computes the corresponding next cell for a given action and a given current cell.
    - Works both ways using the power of clpfd.
- query_iter(S, D, R)
    - Simulates iterative deepening by calling the built-in predicate *call_with_depth_limit* and increasing the allowed depth recursively.

Successor State Axioms

- Fluent: *rock(R, C, result(A,S)):*
  I.e. cell (R,C) contains a rock in situation result(A,S).
  Two Definitions
    - It was not true and something made it true.
        - (R,C) cell should not contain an obstacle.
        - (R,C) should be a valid cell within the grid.
        - (R,C) did not contain a rock in the previous state (the agent cannot move 2 rocks in one push).
        - The agent was in a neighbour cell to the rock in the previous state and did an action in the rock direction.
    - It was true and nothing affected it.
        - (R,C) contained a rock in situation S.
        - If the agent was in an adjacent cell, then its action of pushing the rock was unsuccessful.
            - The rock is next to an edge.
            - The rock is next to an obstacle.
            - The rock is next to a rock in the agent direction.
- Fluent: *agent(R, C, result(A,S))*
  I.e. agent is on cell (R,C) in situation result(A,S).
  Two Definitions
    - It was not true and something made it true.
        - The aspired cell is valid.
        - Two cases:
            - Aspired cell is not an obstacle and not a rock.

        - Aspired cell is a movable rock to a valid position.
    - It was true and nothing affected it.
        - (R,C) contained agent in situation S.
        - Next cell in action direction is an obstacle.
          OR
        - Next cell in action direction is an unmovable rock (next to another rock or obstacle).

How to Query

1. Pick a grid from the grid tests folder to test, and insert its number in the file names in file logic-agent lines **11** and **12**. Replace <N> with file number.
   :- include('initial-state-Sol<N>").
   :- include('query-Sol<N>").
2. Cd to the project directory.
3. Run the SWI Prolog command prompt (*swipl* for mac).
4. Consult the file (logic-agent.pl): *consult('logic-agent.pl')*.
5. Type the query "*query_iter(S, 1, R).*".
6. *S* is the goal state and *R* is the result of calling call_with_depth_limit recursively.
7. *Warning:* For some unknown buggy reason in the clpfd library, on some versions of SWI Prolog, the query may throw an error "*clpd_current_propagator*" does not exist. In that case, querying using "call_with_depth_limit(query(S), 20, R)" may fix the issue and remove the error, and afterwards *query_iter(S, 1, R)* works normally. We think this is something related to memory issues since running the query several times in a row may suddenly yield an output.

Examples

```
?- query_iter(S, 1, R).
S = result(left, result(down, result(left, s0))) .

?- consult('logic-agent.pl').
true.

?- query_iter(S, 1, R).
S = result(right, result(up, s0)) .

?- consult('logic-agent.pl').
true.

?- query_iter(S, 1, R).
S = result(right, s0) .
```

1.
    a. Grid

P	O	O
R	T	O
A	O	O

?- query_iter(S, 1, R).
S = result(right, result(up, s0)) .

2.
    a. Grid

B	B	B
P	R	A
T	B	B

?- query_iter(S, 1, R).

S = result(left, result(down, result(left, s0))) .

3.

a. Grid

```
-------------
| B | B |
-------------
| A | T |
-------------
```

?- query_iter(S, 1, R).

S = result(right, s0) .