Introduction to Artificial Intelligence
Project 2: Help R2-D2 Escape!
Report

Ahmed Soliman
Ahmad Elsagheer

November 21, 2017

1 Introduction

A grid of size $R \times C$ that consists of R rows and C columns is given. The grid contains a robot, a teleport, obstacles, rocks, pressure pads and empty cells. In one move, the robot can do one of the following actions:

- move to an adjacent empty cell. Two cells are adjacent if they share a side.
- push a rock from one cell to an adjacent cell. The robot must be in a cell adjacent to the cell containing the rock and the new cell of the rock must be either an empty cell or a free pressure pad.

2 Grid Representation

The problem grid is generated randomly using a java file `GenGrid.java` that takes the grid dimensions from the user and generates a grid in an output file that should be loaded in the agent's program. Examples of the output files can be found in the folder `examples`.

R2D2 initial world can be visualized as two-dimensional grid using the below symbols. Rows are numbered from top to down and columns are numbered from left to right using a 1-based indexing. The grid file contains the visualization commented at the beginning of the file.

Cell Description	Char	Cell Description	Char
Teleport	T	Robot	R
Obstacle	#	Empty cell	.
Pressure pad	P	Rock	O

The grid is represented in the output file using the following facts. `Location` is a structure of the format `location(X, Y)` where `X` denotes the row and `Y` denotes the column.

- `grid_size(R, C)`: where `R` and `C` are the number of rows and columns of the grid, resepectively.
- `r2d2_init_location(Location)`: where `Location` is the initial position of the robot in the grid.
- `teleport(Location)`: where `Location` is the teleport position.
- `rock_location(Location)`: where `Location` is the rock position. Each rock has its own fact.
- `pressure_pad(Location)`: where `Location` is the pad position. Each pad has its own fact.
- `obstacle(Location)`: where `Location` is the obstacle position. Each obstacle has its own fact.

2.1 Actions and Predicate Terms

Actions are represented in the agent's program as facts `action(name, DX, DY)` where `name` is (`north`, `east`, `south` or `west`) and `DX` and `DY` are the values that should be added to each coordinate value of the current position to get the new position.

The following predicate terms are used in the agent's program:

- `goal_test(S)`: true if the robot is at the teleport location in situation `S`.
- `in_grid(C)`: true if cell `C` is inside the grid.
- `free_cell(C, S)`: true if cell `C` is empty at situation `S`.
- `valid_move(C1, C2, A, S)`: true if the robot can move from `C1` to `C2` if it does action `A` in situation `S`.
- `movable_rock(C1, C2, C3, A, S)`: true if at situation `S` there is a robot is at `C1`, a rock at `C2`, a free cell at `C3` and the rock will move from `C2` to `C3` if the robot does action `A`.
- `robot_away(Cell, A, S)`: true if the robot at situation `result(A, S)` is not at `Cell`.
- `active_teleport(Cell, S)`: true if the teleport at `Cell` is active in situation `S`.
- `check_pads(P, S)`: true if each pad cell in list `P` has a rock in situation `S`.

2.2 Successor-State Axioms

Robot Axiom `robot(L, result(A, S))` is true if and only if the robot was at location `L'` in situation `S` and can make a valid move to location `L`. This is checked using the predicate `valid_move(L', L, A, S)`.

Rock Axiom `rock(L, result(A, S))` is true if and only if the rock was:

- at location `L` in situation `S` and the action `A` didn't affect the location of the rock (i.e. the robot moved to a location different from `L`). This is checked using the predicate `robot_away(L, A, S)`.
- at location `L'` in situation `S` and the action `A` caused the rock to be moved from `L'` to `L` (i.e. the new robot position is `L'`). This is checked using the predicate `movable_rock(L'', L', L, A, S)` where `L''` is the robot location in situation `S`.

3 How it works

Before running a query, put the grid file in the same directory as `rd2d.pl` file, then update the grid file name in `rd2d.pl` line 6 (default grid file name is `facts`). In this file, you can change the depth limit as well.

To run the query, run the file `run.pl` using `prolog` and call the predicate `search([])`. The output will be a sequence of moves of the format (`action, count`) where `action` is (`north`, `east`, `south` or `west`) and `count` is the number of times this action should be applied. This is a compact form instead of printing the situations in the nested result format.

3.1 Examples

In `examples` folder, there are two examples. These examples are described below.

Example 1

```
.R..
.O#.
TP..
```

Output: [(south,1),(west,1),(east,1),(west,1),(east,1),(north,1),(west,1),(south,2)]

Example 2

```
ROP
.##
.T.
```

Output: [(east,1),(west,1),(south,2),(north,1),(south,1),(north,1),(south,1),(east,1)]