Mini-project

Prolog: searching in an infinite space

October 2016

Goals

- Use first-order logic predicates with the Logic Programming paradigm to solve a simple AI problem: searching objects in an infinite space.

Deadline:

November 15, 2016.

Description of the problem

A robot is placed in one of an infinite number of square rooms, placed one near each other on an infinite surface. Every room has four neighbouring rooms to which the robot can move by performing only one operation.

Some rooms contain boxes, which the robot can pick up by performing one operation. The task of the robot is to pick up all boxes, to carry and deliver them in the room where he was initially placed (the origin). The robot is informed to deliver a specified number of boxes to the origin, which we know for sure to exist in the search space of the robot (the robot is not asked to deliver more boxes than there exist).

The robot can not see from one room into another, thus in order to learn if one or more boxes exist in a room, the robot must move into that room. At any time, the robot can carry at most one box. Once the robot picks up a box, he must carry it back to the origin.

To orient itself, the robot will use the Cardinal points North-South-East-West, which are also the directions where it can move.

Requirements

Implement the predicates `initMyData` and `perform` from the file `t3.pl`.

The predicate `perform` is evaluated at every step to decide the next action of the robot. The decision should be made depending on

- whether there exists a box at the current position, and
- the internal state of the robot.

The signature of the predicate `perform` is

```
perform(+ProgramData, +ContainsBox, -Action, -ProgramDataUpdated)
```

where the arguments prefixed with `+` indicate input arguments, and the arguments prefixed with `-` indicate output arguments:

- The argument `ProgramData` is a term which represents the current state of the robot. Here, the robot stores all information he needs to know in order to decide what to do next.
  - ▶ You are free to decide the structure and content of the state of the robot. Note that the robot should keep track of its current position on the surface, and how many boxes he has to find and deliver.
- ▶ The argument `ContainsBox` is either `true` or `false`, and it tells the robot if there is at least one box in the room where he is (its current position).

Based on the values of `ProgramData` and `ContainsBox`, the robot must compute the values of the following two arguments:

- ▶ `Action`, which indicate the action decided by the robot, which can be one of the following:

  `done`: the robot decided to stop.

  `pickBox`: the robot decided to pick up a box.

  `deliverBox`: the robot decided to drop the box he has in the room where he is.

  `move(north)`, `move(east)`, `move(south)`, `move(west)`: the robot decided to move without box in the specified direction.

  `moveWithBox(north)`, `moveWithBox(east)`, `moveWithBox(south)`, `moveWithBox(west)`: the robot decided to carry a box in the specified direction.

- ▶ `ProgramDataUpdated`: The state of the robot after `Action` is performed. This updated state is passed to the next call of the predicate `perform`.

The initial state of the robot will be `ProgramData`, whose value will be determined by the call of the predicate

```
initMyData(+NBoxesToPick, -ProgramData)
```

which receives as input the number of boxes which the robot should pick up. The internal state of the robot is accessed only by the predicates `initMyData` and `perform`, thus you are free to choose one that is suitable to solve this problem.

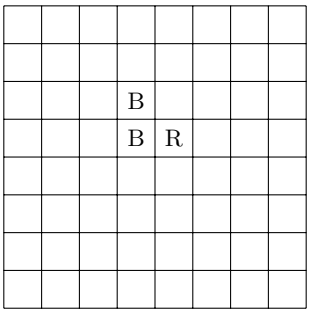
There are a few restrictions on the actions the robot can make:

- ▷ the `move` action can be made only if the robot does not carry any box at that moment (if he performed `pickBox`, he should have performed also `deliverBox`);
- ▷ the `moveWithBox` action can be made only if the robot carries a box at that moment (he performed `pickBox` but did not perform `deliverBox` since then);
- ▷ the action `pickBox` can be made only if the robot does not carry yet a box, and it is placed in a room containing a box. After the robot picks the box, that box is no more in that room;
- ▷ the action `deliverBox` can be made only if the robot carries a box, and is located in the origin;
- ▷ the action done can be made only if the robot delivered the required number of boxes, it is in the origin, and does not carry any box.

If there are boxes in the origin, the robot must perform `pickBox` and `deliverBox` in order to deliver each of them.

Example

Assume the following configuration of rooms with boxes B, and the initial position of the robot R depicted below:



A correct sequence of actions performed by the robot could be:

```
move(east)
move(north)
move(west)
move(west)
pickBox                    (here ContainsBox is true)
moveWithBox(east)
moveWithBox(south)
deliverBox
move(north)
move(west)
```

```
move(south)
pickBox                    (here ContainsBox is true)
moveWithBox(east)
deliverBox
done
```

Programming guidelines

Download the archive `robot.zip` which contains two text files:

- `t3.pl`: This is the file where you should add tour own code. You should implement the predicates

  `initMyData/2` and `perform/4`

  but you can also define and use as many auxiliary predicates as you want.

- `t3t.pl`: This file contains a Prolog program to test the correctness of your implementation.

  Every definition of the predicate `t3test/4` contains a test, caracterised by an identifier, the number of boxes to be delivered, the positions of the boxes as list of pairs of coordinates, and a limit imposed on the length of the solution.

  - for testing a certain test, use the predicate `t3test/1`, with the identifier of the test.
  - for testing all the tests, with a final report, use `t3testallinone/0`.

  Initially, consult the file `t3t.pl` in Prolog, which will load the file `t3.pl`.

When you are done, send the file `t3.pl` with your implementation to me:

`mircea.marin@e-uvt.ro`

Do not forget to mention your name when you submit your labwork.