Logic and Functional Programming

# Prolog: Searching in an infinite space

Deadline: 25.05.2021.

## Objectives

- the usage of predicates and of logic programming to solve a simple AI problem: searching objects in an infinite space.
- for bonus: a more advanced search, in a space with some obstacles

## Problem description

A robot is placed in a room from an infinite space of square-shaped rooms, placed one-near-another on an infinite surface. Every room has four neighboring rooms into which the robot can move with a single operation.

Some rooms may contain boxes. The task of the robot is to pick them up and bring them back in the room where the robot was initially placed (the origin). The robot is asked to bring back a specified number of boxes, which is known to exist for sure in the environment.

The robot can not see from one room to another. Therefore, in order to find out if a room contains a box (or more), the robot must go into that room.

Every time, the robot can carry at most one box which, once it is picked up, he must carry to the origin.

To orient itself, the robot will use the cardinal points north-south-east-west, which are the directions where it can move.

## Requirements

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

The predicate `perform` is evaluated at every step to decide the next action of the robot. The decision must be made depending on the fact if there is a box at the current position, and the internal state of the robot. The signature of the predicate is:

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

The predicate receives in `ContainsBox` the value `true` if its current position contains at least one box, or `false` is there are no boxes at the current position.

The predicate will instantiate `Action` to the action that the robot decides to perform, which is one of:

`done`, `pickBox`, `deliverBox`, `move(north)`, `move(south)`, `move(east)`, `move(west)`, `moveWithBox(north)`, `moveWithBox(south)`, `moveWithBox(east)`, `moveWithBox(west)`

The robot can keep its internal state, which is passed between two evaluations of the predicate `perform`.

The initial state is specified by the program using the predicate

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

which specifies the number of boxes that the robot should collect and deliver, and instantiates `ProgramData` with the initial state of the robot. Afterwards, every call of the predicate `perform` will receive the previous state of the robot, and produces a new state for the robot (in accordance with the chosen action), which is used to instantiate the argument `ProgramDataUpdated`.

The internal state of the robot is accessed only through the predicates `initMyData` and `perform`. Therefore, the programmer is free to choose how to represent the internal state.

The actions of the robot must respect the following restrictions:

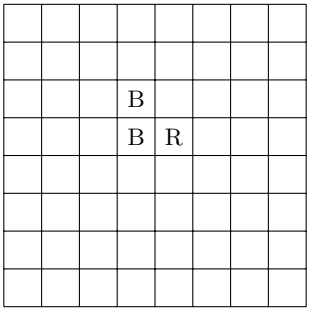
- action `move` can be performed only if the robot does not carry a box at that moment (if the robot performed `pickBox`, he should also have perfomed `deliverBox` afterwards).
- action `moveWithBox` can be performed only if the robot carries a box at that moment (the robot performed action `pickBox` but did not perform yet `deliverBox`).
- action `pickBox` can be performed only if the robot does not carry a box, and it is in a room which contains a box. After the robot picks a box, he carries it and that box is no more in that room.
- action `deliverBox` can be performed only if the robot carries a box and it is in the origin.
- action `done` can be performed only if the robot has 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` followed by `deliverBox` for each of them to be delivered.

There can be more boxes in one room. Every box requires separate delivery.

## 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
```

## Bonus: Obstacles

For bonus, it is required to allow the robot to act in an environment with unavailable rooms, with the following amendments:

- an unavailable room can not have neighbors which are unavailable rooms.
- there is no unavailable common neighbor of the common neighbors of an unavailable room.

For bonus, you should implement the predicate

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

The argument `AvailableDirections` will be a list with the directions to available rooms. The predicate `perform` for the problem without bonus is identical with `perform` for the problem with bonus, with the argument `AvailableDirections` instantiated with the list `[north,south,east,west]`.

For the bonus, you should implement the predicate `perform` with both four and five arguments.

## Programming guidelines

Download the text files:

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

  `initMyData/2`, `perform/4`; and `perform/5` for the bonus

  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.

  > To test your implementation, consult the program `t3t.pl` after you finished editing the file `t3t.pl`.

  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. Run

  ```
  ?- t3test(testid).
  ```

  where *testid* $\in \{\text{a3}, \text{b1}, \text{b2}, \text{c1}, \text{c2}, \text{d1}, \text{d2}, \text{d3}, \text{d4}, \text{d5}, \text{e1}, \text{e2}, \text{bonus1}\}$.

  – for testing all tests, with a final report, use `t3testallinone/0`.

  ```
  ?- t3testallinone.
  ```

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

When you are done, upload the file `t3.pl` with your implementation.