Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2015-2016

Programming Assignment 5

Due date: 30 May 2016, Monday, 23:55

1 Introduction

In this assignment, you are going to help a robot in its navigation problem in a 2D grid world domain. The robot wants to navigate in the grid and it is given a sequence of actions to reach a designated cell from another cell. However, the given information is malformed, that is, some actions are missing and some actions are harmful. Your task is to correct the given sequence of actions so that the robot can navigate between the given cells.

2 Specifications

2.1 Grid Domain

1. The robot is placed in a 2D $N \times N$ grid world domain where it can employ four actions as *north*, *south*, *east* and *west*. The grid is surrounded by walls so that the robot can't leave the grid.

2. The grid cells are indexed with X and Y coordinates where X coordinate increases from left to right and Y coordinate increases from top to bottom. The top left corner of the grid has (0,0) coordinates.

3. Each action moves the robot to one step to the intended direction. That is, the robot moves by one cell in each step.

4. Actions are deterministic, that is, an action always leads the robot to the neighbour cell in the intended direction unless there is a wall. The robot stays at the same cell when it bumps to a wall.

An example 8×8 domain is given in Figure 1.

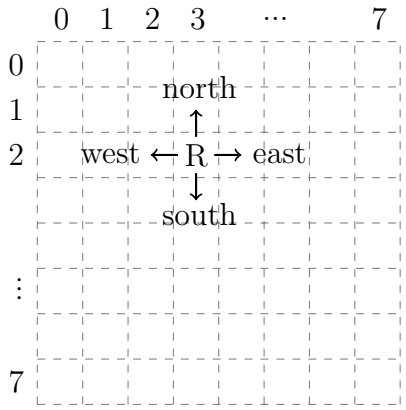

Figure 1: An example 8×8 grid domain where the robot is placed at position (3, 2).

2.2 Task

You are going to implement a predicate called **complete_path/5** taking 5 arguments.

```
complete_path(N, START_POSITION, END_POSITION, PATH, COMPLETE_PATH).
```

A query including this predicate can be used to complete the given path (if such a path exists) by:

- deleting harmful actions leading robot in the wall
- replacing missing actions given as "unknown" by one of the actions

Also, it can be used to verify that the given complete path is truly a possible completed version of the given path as there can be multiple paths between two grid cells. It is guaranteed that this predicate will be queried by giving at least the first four arguments.

1. The argument **N**, is the size of the grid which is guaranteed to be greater than or equal to 1.
2. **START_POSITION** argument is the cell position that the path starts while **END_POSITION** is the one where the path ends. A cell position in the grid is represented with the predicate **position/2** as x and y coordinates of the cell is given with related arguments: `position(X, Y)`. These given positions are guaranteed to be inside the grid.
3. A path is a list of actions that must be taken in each step. For example, a path from the position (0, 0) to the position (3,3) can be [*east*, *east*, *south*, *east*, *south*, *south*].
4. A single action leading the robot to a wall is considered **harmful** and such actions must be removed from the path during the correction.
5. A missing action is given with the keyword **"unknown"** in the given path.
6. The argument **PATH** represents the given path to the robot. This path may have missing or harmful actions. In the same example above from (0, 0) to (3, 3), a malformed path may be [*north*, *east*, *east*, *south*, **unknown**, *south*, *south*]. The first action is an harmful one since it makes the robot to bump a wall and the fifth action is missing.
7. The argument **COMPLETE_PATH** represents the completed version of the given path. This argument should not have any missing or harmful actions. The completed version of the above example is again [*east*, *east*, *south*, *east*, *south*, *south*].

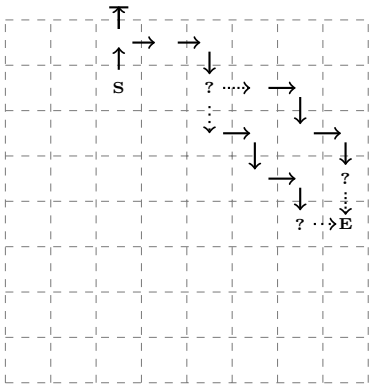

Figure 2: An uncomplete path in 8×8 grid domain. Two possible paths are given with dotted arrows where the starting and ending positions are given with **S** and **E**, respectively.

8. The resulted complete path may not be the shortest path between the given positions. However, it should not have any missing or harmful actions.

9. The predicate should not add more actions to the given path. It should only correct the path.

10. The predicate can only change an unknown action with an action that is not harmful. If the only possible actions replacing "unknown" keyword are harmful ones, then the predicate should fail.

11. If there is no possible complete path between the given positions, the predicate should fail. Such a case can happen when the number of steps are not enough to reach the given destination or the path leads the robot to another position.

12. There may be different paths between given positions. Therefore, the predicate should be able to find all possible complete paths. **HINT:** You do not have to use a heuristic to find possible complete paths. Let Prolog try all solutions exhaustively.

2.3 An Example

In the example drawn in Figure 2, the robot is given a path of [*north*, *north*, *east*, *east*, *south*, **unknown**, *east*, *south*, *east*, *south*, **unknown**]. The starting cell position is (2, 1) and the ending cell position is (7, 4). To complete the given path, firstly, the second *north* action must be removed as it leads the robot to a wall and it is a harmful action. Next, the action to be taken at the position (4, 1) and the last action are missing. The robot can try four different actions at (4, 1) but only two of them can complete the given path correctly. If this missing action at (4, 1) is replaced with *east* action, then the last action must be replaced with *south* and vice versa. These possible complete paths are shown in the Figure 2. The query including this case and the result is given in the following run.

```
?- complete_path(8, position(2, 1), position(7, 4), [north, north, east, east, south, unknown, east, south, ↩
    east, south, unknown], CP).
CP = [north, east, east, south, east, east, south, east, south, south] ;
CP = [north, east, east, south, south, east, south, east, south, east] ;
false.
```

2.4 Example Runs

```
?- complete_path(5, position(0, 0), position(1, 0), [east], CP).
CP = [east].

?- complete_path(5, position(0, 0), position(1, 0), [east], [east]).
true.

?- complete_path(5, position(0, 0), position(1, 0), [east, east], CP).
false.
```

```
?- complete_path(5, position(0, 0), position(1, 0), [], CP).
false.

?- complete_path(5, position(0, 0), position(1, 0), [north], CP).
false.

?- complete_path(5, position(0, 0), position(1, 0), [north, east], CP).
CP = [east].

?- complete_path(5, position(0, 0), position(1, 0), [unknown, east], CP).
false.

?- complete_path(5, position(0, 0), position(1, 0), [unknown, unknown, east], CP).
CP = [east, west, east] ;
CP = [south, north, east] ;
false.

?- complete_path(1, position(0, 0), position(1, 0), [east], CP).
false.

?- complete_path(1, position(0, 0), position(1, 0), [unknown], CP).
false.

?- complete_path(1, position(0, 0), position(1, 0), [], CP).
false.

?- complete_path(1, position(0, 0), position(0, 0), [north], CP).
CP = [].

?- complete_path(1, position(0, 0), position(0, 0), [], CP).
CP = [].

?- complete_path(1, position(0, 0), position(0, 0), [unknown], CP).
false.

?- complete_path(5, position(0, 0), position(3, 3), [east, south, unknown, east, south, east], CP).
CP = [east, south, south, east, south, east].

?- complete_path(5, position(3, 2), position(4, 3), [unknown, east, east, south], CP).
CP = [east, south] ;
CP = [west, east, east, south] ;
false.

?- complete_path(5, position(0, 0), position(3, 3), [east, south, unknown, unknown, south, east], CP).
CP = [east, south, east, south, south, east] ;
CP = [east, south, south, east, south, east] ;
false.

?- complete_path(5, position(0, 0), position(3, 3), [unknown, east, south, unknown, unknown, south, east], CP).
false.

?- complete_path(5, position(0, 0), position(3, 3), [north, east, south, unknown, unknown, south, east], CP).
CP = [east, south, east, south, south, east] ;
CP = [east, south, south, east, south, east] ;
false.

?- complete_path(5, position(0, 0), position(3, 3), [east, east, north, north, north, south, east, south, south↩
    ], CP).
CP = [east, east, south, east, south, south].

?- complete_path(5, position(4, 1), position(0, 2), [unknown, north, north, west, south, west, unknown, south, ↩
   unknown], CP).
CP = [north, west, south, west, west, south, west] ;
CP = [south, north, north, west, south, west, west, south, west] ;
false.
```

3 Regulations

1. **Programming Language:** You should write your code using SWI Prolog and test them on inek machines.

2. **Late Submission:** See the syllabus for the details.

3. **Cheating:** All the work should be done individually. **We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.

4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

4 Submission

Submission will be done via COW. Submit a single file called "hw5.pl" through the COW system. The file must start with the following line.

```
:- module(hw5, [complete_path/5]).
```