CSEN B303: Concepts of Programming Languages, Spring Term 2024
Haskell Project: Gold Digger Robot

Due: 6^{th} *June 2024*

1 Overview

In this project you are going to implement the AI module for a gold digger robot in Haskell. The environment in which the robot operates is an 4×4 grid of cells. Initially, a cell on the grid is either empty, contains the robot, or contains gold. The robot can move in all four directions and is able to dig up gold only if it is in the same cell as the gold. Your program will take as input the initial position of the robot and the positions of all of the gold. The objective of the AI module is to compute a sequence of actions that the robot can follow in order to go to all the gold and dig for it. Below is an example grid.

	0	1	2	3
0				
1			G	
2			G	
3	R			

The robot R starts at (3,0) while the gold is at (2,2) and (1,2). One possible generated sequence of actions by your program is:
`["up","right","right","dig","up","dig"]`

2 Type Definitions

Your implementation must contain the following type definitions.

2.1 Cell

```
type Cell = (Int,Int)
```

A `Cell` represents a position on the grid with the first coordinate in the pair representing a row number, and the second coordinate representing a column number.

2.2 MyState

```
data MyState = Null | S Cell [Cell] String MyState
```

The `MyState` structure represents the state of the robot at any given time. It is either `Null` or the data constructor `S` followed by a cell representing the robot's position, a list of cells representing the positions of the gold to be dug up, a string representing the last action performed to reach this state, and the parent state. The parent state is the last state the robot was in before doing the last performed action. The initial state of the robot in the above grid is accordingly represented as: `S (3,0) [(2,2),(1,2)] "" Null`

3 Functions

Your implementation must also contain the following functions. It is recommended to implement the functions in the order they are written below. You are not allowed to change the given type definitions of the functions, but you can add any other helper functions you need.

3.1 Robot Actions

3.1.1 up

The function takes as input a state and returns the state resulting from moving up from the input state. If up will result in going out of the boundaries of the grid, `Null` should be returned.

Type: `up :: MyState -> MyState`

Example(s)

```
up (S (3,0) [(2,2),(1,2)] "" Null)
= (S (2,0) [(2,2),(1,2)] "up" (S (3,0) [(2,2),(1,2)] "" Null))
up (S (0,0) [(2,2),(1,2)] "" Null)
= Null
```

3.1.2 down

The function takes as input a state and returns the state resulting from moving down from the input state. If down will result in going out of the boundaries of the grid, `Null` should be returned.

Type: `down :: MyState -> MyState`

Example(s)

```
down (S (3,0) [(2,2),(1,2)] "" Null)
= Null
down (S (2,0) [(2,2),(1,2)] "up" (S (3,0) [(2,2),(1,2)] "" Null))
= (S (3,0) [(2,2),(1,2)] "down" (S (2,0) [(2,2),(1,2)] "up" (S (3,0) [(2,2),(1,2)] ""
    Null)))
```

3.1.3 left

The function takes as input a state and returns the state resulting from moving left from the input state. If left will result in going out of the boundaries of the grid, `Null` should be returned.

Type: `left :: MyState -> MyState`

Example(s)

```
left (S (3,0) [(2,2),(1,2)] "" Null)
= Null
```

3.1.4 right

The function takes as input a state and returns the state resulting from moving right from the input state. If right will result in going out of the boundaries of the grid, `Null` should be returned.

Type: `right :: MyState -> MyState`

Example(s)

```
right (S (3,0) [(2,2),(1,2)] "" Null)
= S (3,1) [(2,2),(1,2)] "right" (S (3,0) [(2,2),(1,2)] "" Null)
```

3.1.5 dig

The function takes as input a state and returns the state resulting from digging up gold from the input state. Digging should not change the position of the robot, but removes the dug up from the list of gold to be dug up. If the robot is not in the same position as any of the gold, `Null` should be returned.

Type: `dig :: MyState -> MyState`

Example(s)

```
dig (S (3,1) [(2,2),(3,1)] "right" (S (3,0) [(2,2),(3,1)] "" Null))
= S (3,1) [(2,2)] "dig" (S (3,1) [(2,2),(3,1)] "right" (S (3,0) [(2,2),(3,1)] "" Null)
    )
```

3.2 Solution Generation

3.2.1 isGoal

The function takes as input a state, returns `True` if the input state has no more gold to dig up (the list of gold positions is empty), and `False` otherwise.

Type: `isGoal :: MyState -> Bool`

Example(s)

```
isGoal (S (3,1) [] "dig" (S (3,1) [(3,1)] "right" (S (3,0) [(3,1)] "" Null)))
= True
isGoal (S (3,1) [(3,1)] "right" (S (3,0) [(3,1)] "" Null))
= False
```

3.2.2 nextMyStates

The function takes as input a state and returns the set of states resulting from applying `up`, `down`, `left`, `right`, and `dig` from the input state. The output set of states should not contain any `Null` states.

Type: `nextMyStates :: MyState -> [MyState]`

Example(s)

```
nextMyStates (S (3,0) [(2,2),(1,2)] "" Null)
 = [(S (2,0) [(2,2),(1,2)] "up" (S (3,0) [(2,2),(1,2)] "" Null)),S (3,1) [(2,2),(1,2)]
     "right" (S (3,0) [(2,2),(1,2)] "" Null)]
```

3.2.3 search

The function takes as input a list of states. It checks if the head of the input list is a goal state, if it is a goal, it returns the head. Otherwise, it gets the next states from the state at head of the input list, and calls itself recursively with the result of concatenating the tail of the input list with the resulting next states (take care that the order of the concatenation here is important, the next states must be placed by the end of the list).

Type: `search :: [MyState] -> MyState`

3.2.4 constructSolution

The function takes as input a state and returns a set of strings representing actions that the robot can follow to reach the input state from the initial state. The possible strings in the output list of strings are only `"up"`, `"down"`, `"left"`, `"right"`, and `"dig"`.

Type: `constructSolution :: MyState -> [String]`

Example(s)

```
constructSolution (S (3,1) [] "dig" (S (3,1) [(3,1)] "right" (S (3,0) [(3,1)] "" Null)
    ))
= ["right","dig"]
```

3.2.5 solve

The function takes as input a cell representing the starting position of the robot, a set of cells representing the positions of the gold, and returns a set of strings representing actions that the robot can follow to reach a goal state from the initial state.

Type: `solve :: Cell -> [Cell] -> [String]`

Example(s)

```
solve (3,0) [(2,2),(1,2)]
= ["up","right","right","dig","up","dig"]
```

4 Guidelines

- This project should be done by the **same** teams as the Prolog Project. Please note that in case of desired changes, you must swap with a team member from another team.
- In case of any requested changes, please use the following link to submit each swap request separately by ***Tuesday 21st of May at 11:59 PM*** https://forms.gle/X5C8BLJ7pwgFNcRd6. **All** members of both team must be informed before the form submission. There will be **no changes** allowed after the deadline.
- You can consult the manual and search online. However, all work done in this project must be done by the team members and the team members only. **All team members should work on this project equally** and no work should be done by anyone outside the team. Do not discuss approaches for completing the project with colleagues. This also means that copying code from online resources or ChatGPT is not allowed. Team evaluations *might* be conducted at the end in order to verify that. All files **will** be checked for plagiarism.
- Each team should submit a single `.hs` file, containing the team's full project implementation. The submitted file **must** abide by the following rules:
  - The file should be named in the following format "**Team_TeamNumber.hs**", for example: "**Team_7.hs**".
  - You should include a clear documentation per implemented function. You should write each function's documentation above the function's implementation.
- It is the team's full responsibility to successfully submit a valid `.hs` file before the project's deadline using the following link: https://forms.gle/46JbQcgS4VEDBe5fA