Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2017-2018

Homework 1

Due date: 25 March 2018, Sunday, 23:55

1 Objectives

This assignment aims to assist you to discover the functional programming world by the help of Haskell programming language.

2 Problem Definition

In this assignment, you will be simulating one of the classical Artificial Intelligence problems: the *hunter-prey* problem. In this problem, the agent called as *hunter* pursues and the other agent called as *prey*, and the prey tries to avoid encounters with the hunter. Although the problem itself contains autonomous entities, we will assume that we know the initial status of the agents in the environment and the actions which the agents are going to take during the simulation are given in first-hand.

The simulation environment will be represented by a 2D grid (NxM) where each cell is either an empty cell, the cell containing an obstacle or the cell contains an agent. You can see an example of a representation of this environment in the Figure 1.

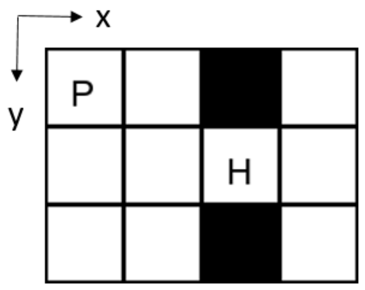

Figure 1: An example of simulation environment

In this representation;

- White cells show the empty cells.
- Black cells show the cells those contain an obstacle.
- The cells with letters H and P contain the hunter and the prey respectively.
- The coordinate of the top left corner is (0,0). Therefore, we can say that the hunter is located at (2,1) and the prey is located at (0,0). The cells at (2,0), and (2,2) contain an obstacle.

Each agent can move in the given four direction (North, East, South, West) if the border is not reached and there is no obstacle in that direction. If an action that is in the direction of border or an obstacle is given, the agent will stay in the same cell for the next time step as well. We will assume that no collision will occur if the agents move towards each other.

Assume that the *West* action is taken by the hunter, and the *East* action is taken by the prey in the given environment in Figure 1. Then the environment becomes as Figure 2, and the positions of the hunter and the prey will be (1,1), and (1,0) respectively.

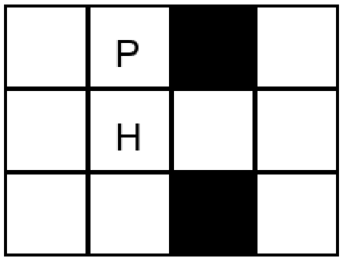

Figure 2: New environment after one time step

If we take the simulation one step further with the actions *North* for the hunter, and *South* for the prey, we get the environment as Figure 3:

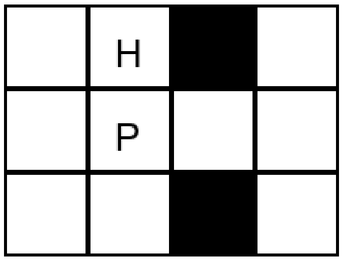

Figure 3: New environment after two time steps

The simulation will terminate if the hunter and the prey ends up in the same cell or the last actions in the given sequence are performed.

3 Specifications

You are expected to write a Haskell function called "simulate" which takes a nested list of **Cell** representing the state of 2D environment and a list of (**Direction**, **Direction**) tuples corresponding to the actions taken by the hunter and the prey respectively and returns a **Result** which reports the result of the simulation.

Type declaration of this function is given below.

```
simulate :: [[Cell]] -> [(Direction, Direction)] -> Result
```

In the nested list of Cell, four types of Cell will appear corresponding to the each type of cells:

- O will show the empty cells.
- X will show the cells with an obstacle.
- H will show the cell that contains the hunter.
- P will show the cell that contains the prey.

Here is the definition of Cell datatype:

```
data Cell = O | X | H | P deriving (Read,Show,Eq)
```

Here is the representation of the environment from Figure 1 in the nested list form:

```
[[P,O,X,O],[O,O,H,O],[O,O,X,O]]
```

You can find the Directions corresponding to the each action and the definition of this datatype below:

- N for North
- E for East
- S for South
- W for West

```
data Direction = N | S | E | W deriving (Read,Show,Eq)
```

An example of the action sequence (as the example in Section 2.) in the list of (Direction, Direction) form will be as the following:

```
[(W,E),(N,S)]
```

The function returns two types of Result:

- Fail, if the hunter fails to catch the prey in the given action sequence.
- Caught $<$ *the coordinate of the final cell* $>$", if the hunter catches the prey.

Here is the definition of Result datatype:

```
data Result = Fail | Caught (Int,Int) deriving (Read,Show,Eq)
```

The *simulate* function must be defined in the Hw1.hs file which you can find in homework files. After the implementation of this function, you can load your module to ghci by typing to terminal:

```
$ ghci Hw1.hs
```

For moodle system, running the file will load the file automatically to the ghci.

Then you can call the function *simulate* as:

```
*Hw1> simulate [[P,O,X,O],[O,O,H,O],[O,O,X,O]] [(W,E),(N,S)]
```

Then you should get the following result:

```
Fail
```

4 Sample I/O

```
*Hw1> simulate [[P,O,O],[O,O,O],[O,O,H]] [(W,E),(N,S)]
Caught (1,1)

*Hw1> simulate [[P,O,O],[O,O,O],[O,O,H]] [(W,E),(N,E)]
Fail

*Hw1> simulate [[O,O,X],[O,O,X],[O,H,O],[O,X,O],[O,X,P]] [(E,N),(S,S),(S,W),(E,N)]
Caught (2,4)
```

5 Regulations

1. **Programming Language:** You must code your program in Haskell. Your submission will be tested with `ghc/ghci` on moodle system. You are expected to make sure your code loads successfully in `ghci` interpreter on the moodle system.

2. **Late Submission:** You have been given 10 late days in total and at most 3 of them can be used for an assignment. After 3 days you get 0, no excuse will be accepted.

3. **Cheating: 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.

6 Submission

Submission will be done via moodle system. You can either download the template file, make necessary additions and upload the file to the system or edit using the editor on the moodle and save your changes.