Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2017-2018

Homework 2

Due date: 8 April 2018, Sunday, 23:55

1 Objectives

This assignment aims to assist you to expand your knowledge on functional programming by the help of Haskell programming language. It also contains some object-oriented perspective which will make you understand better why we need different paradigms for different kinds of tasks.

2 Problem Definition

In this assignment, you will be simulating the Hunter-Prey problem from the first assignment, in a multi-agent approach. In other words, there will be more than one hunter and one prey in the environment. Each agent will be able to act independently with their given action sets. Their initial positions will be always different. However, two or more than agents with same type may reside in the same cell in a time step. Therefore, we will assume that there will not be any collision in such cases. Agents will have some features so that we can compare them and sort them to build up a hierarchy in order to decide which hunter will catch the prey when they all end up in the same cell. Details will be explained in the next section.

The simulation environment will be represented by a 2D grid (NxM) where each cell is either an empty cell, **a cell with a trap**, the cell containing an obstacle or the cell contains an agent. Figure 1. illustrates an example of a simulation environment.

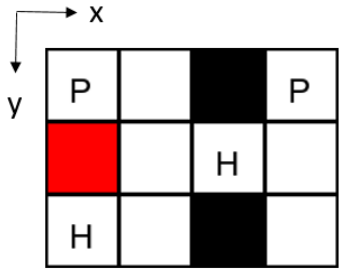

Figure 1: An example of simulation environment

In this representation;

- White cells show the empty cells.
- Red cells show the cell with a trap.
- 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 hunters are located at (0,2) and (2,1). The preys are located at (0,0) and (3,0). The cell at (0,1) contains a trap. 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 it has enough **energy** and the border is not reached and there is no obstacle in that direction. If it does not have enough energy it will rest in the same cell and gain some energy. If it has enough energy but 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 or reside in the same cell. When the prey pass to a cell with a trap, it will lose some energy.

The simulation will terminate when there is no prey left in the environment or the last actions in the given sequences are performed.

3 Specifications

3.1 Record Syntax for The Agents

Each prey will have;

- a unique integer as id,
- an integer representing the energy it has,
- and a list of Direction corresponding the actions it will take during the simulation.

We will use the following record syntax for this type:

```
data Prey = Prey {pID::Int, pEnergy::Int, pActions::[Direction]} deriving (Eq, Show
    , Read)
```

Each hunter will have;

- a unique integer as id,
- a Level type corresponding its experience in hunting,
- an integer representing the energy it has,
- an integer for the number of catches during the simulation,
- and a list of Direction corresponding the actions it will take during the simulation.

We will use the following record syntax for the Hunter type:

```
data Hunter = Hunter {hID::Int, hlevel::Level, hEnergy::Int, hNumberOfCatches::Int,
    hActions::[Direction]} deriving (Eq, Show, Read)
```

Here is the definition of Level type:

```
data Level = Newbie | Intermediate | Expert deriving (Enum, Eq, Ord, Show, Read)
```

Finally here is the definition of Direction type as in the first assignment:

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

3.2 Comparison of The Agents

You are expected to make instances of *Ord* class for *Hunter* and *Prey* according to following specifications.

The comparison of two preys will be as follows:

- The prey with more energy will be prior (**GT** in Haskell) than the other prey.
- If their energies are equal, then the prey with smaller id will be prior.

The comparison of two hunters will be as follows:

- The hunter with upper level will be prior then the hunter with lower level.
- If levels are equal, then the one with more energy will be prior.
- If energies are equal, then the hunter which caught more prey will be prior.
- If the number of catches are also equal, then the hunter with smaller id will be prior.

In order to satisfy these comparison criteria, you will be implementing your *compare* functions for Prey and Hunter types in the following syntax:

```
instance Ord Hunter where
    compare ...

instance Ord Prey where
    compare ...
```

3.3 Simulation Details

You are expected to write a Haskell function called "simulate" which takes a nested list of **Cell** representing the state of 2D environment and returns two lists as a tuple that represents the agents with their locations at the end of the simulation. First one will be the list of Hunter-Coordinate pair in *descending* order with respect to priority. Second one will be the list of Prey in *increasing* order in terms of priority.

```
simulate :: [[Cell]] -> ([(Hunter, Coordinate)], [(Prey, Coordinate)])
```

The definition of Coordinate type given below:

```
type Coordinate = (Int, Int)
```

In the nested list of Cell, five 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 Hunter will show the cell that contains a hunter.
- P Prey will show the cell that contains a prey.
- T will show the cells with a trap.

The Cell datatype is defined with some modification according to the new simulation environment:

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

Since you know all the types which are necessary for the simulation, we can continue with the rules of simulation:

1. The number of actions in given sequences will be equal for all of the agents.
2. Each step decreases the energy of an agent by 1. This will be applied even if the agent cannot pass another cell due to an obstacle or the border.
3. An agent with less energy than 10 cannot move and misses its current action. It rests in the same cell for a timestep and this will increase its energy by 1.
4. When a prey ends up in a cell with trap its energy will be reduced by 10.
5. A trap in the environment (if there exists) is always static. That is, the trap will not be removed after a prey falls into.
6. When a hunter catches a prey, its energy will increase by 20 if its total energy does not exceed 100. If such a case occur the energy of the hunter will be 100.
7. As mentioned before no collision will occur when two or more agents move to the same cell.
8. The hunter with the highest priority will begin to catch the prey that is the lowest in the ordering, if multiple hunters and preys reside in the same cell.
9. A hunter will be able to catch at most 2 preys in a time step. There is no limit for a hunter's number of catches during whole simulation.

4 Sample I/O

Inputs of this assignment will include sorting given agents and the simulation of the agents in given environment with given action sequences. Here are some examples for them with their outputs:

```
*Hw2> sort [Prey 1 100 [S,E], Prey 2 90 [S,N], Prey 3 100 [E,N]]
[Prey {pID = 2, pEnergy = 90, pActions = [S,N]},Prey {pID = 3, pEnergy = 100,
   pActions = [E,N]},Prey {pID = 1, pEnergy = 100, pActions = [S,E]}]
```

```
*Hw2> sort [Hunter 1 Expert 100 10 [N,W], Hunter 2 Expert 80 5 [E,S], Hunter 3
   Newbie 80 1 [N,N], Hunter 4 Expert 100 10 [S,W]]
[Hunter {hID = 3, hlevel = Newbie, hEnergy = 80, hNumberOfCatches = 1, hActions = [
   N,N]},Hunter {hID = 2, hlevel = Expert, hEnergy = 80, hNumberOfCatches = 5,
   hActions = [E,S]},Hunter {hID = 4, hlevel = Expert, hEnergy = 100,
   hNumberOfCatches = 10, hActions = [S,W]},Hunter {hID = 1, hlevel = Expert,
   hEnergy = 100, hNumberOfCatches = 10, hActions = [N,W]}]

*Hw2> simulate [[P (Prey 1 70 [S,E,W,S,N,E]), O, X, P (Prey 2 100 [N,N,W,S,E,E])],
   [T, O, H (Hunter 1 Expert 90 2 [E,S,N,E,S,W]), O],[H (Hunter 2 Newbie 70 0 [E,N,
   S,W,N,N]), O, X, O]]
([(Hunter {hID = 1, hlevel = Expert, hEnergy = 100, hNumberOfCatches = 3, hActions
   = [S,W]},(3,1)),(Hunter {hID = 2, hlevel = Newbie, hEnergy = 86,
   hNumberOfCatches = 1, hActions = [N,N]},(0,2))],[])

*Hw2> simulate [[P (Prey 1 70 [E,W,E,W,E,W]), O, X, P (Prey 2 100 [N,N,W,S,E,E])],
   [T, O, H (Hunter 1 Expert 90 2 [E,S,N,E,S,W]), O],[H (Hunter 2 Newbie 70 0 [E,N,
   S,W,E,N]), O, X, O]]
([(Hunter {hID = 1, hlevel = Expert, hEnergy = 98, hNumberOfCatches = 3, hActions =
    []},(3,2)),(Hunter {hID = 2, hlevel = Newbie, hEnergy = 64, hNumberOfCatches =
   0, hActions = []},(1,1))],[(Prey {pID = 1, pEnergy = 64, pActions = []},(0,0))])
```

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. **Grading:** You will have **20 points** for the correct implementation of *instances* and **80 points** for the correct implementation of *simulate* function.

6. **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.