Simulação de Exploração de Minas Subterrâneas

BCC222 - Programação funcional

1 Introdução

Com o avanço da tecnologia, robôs são cada vez mais utilizado em atividades industriais que envolvem algum tipo de risco. Uma atividade que pode se beneficiar da utilização de robôs é a mineração. Já que ao invés de exigir a presença humana em minas subterrâneas, pode-se utilizar robôs para realizar a extração de minerais de interesse. O objetivo deste trabalho é a implementação de um programa que simule a execução de um robô em uma mina. Para isso, utilizaremos duas pequenas linguagens: A LDM, Linguagem de Descrição de Mina, utilizada para descrever minas a serem exploradas por robôs e a LCR, Linguagens de Comandos de Robô, utilizada para descrever quais ações deverão ser executadas por um robo em uma mina.

As próximas seções descreverão estas linguagens e os tipos de dados necessários para implementação deste trabalho.

2 Tipos de dados utilizados

2.1 Representação de robôs

Primeiramente, precisamos de uma maneira de descrever robôs. A configuração, em um dado instante, do robô durante o seu caminhamento na mina é formada pelas seguintes informações:

- Total de energia: Representa o quanto de energia disponível o robô possui.
- Posição Atual: Descreve em que ponto da mina o robô se encontra em um dado momento. A posição de um robô é descrita como um par ordenado de números inteiros.
- Material Coletado: Descreve a quantidade de material coletado pelo robô até um certo instante.

Desta forma, podemos representar um robô pelo seguinte tipo de dados:

```
type Fuel = Int
type Point = (Int,Int)
type Material = Int

data Robot = Robot {
               energy    :: Fuel,
               position  :: Point,
               collected :: Material
             } deriving (Eq, Ord)
```

Como exemplo deste tipo de dados, considere o seguinte robo de exemplo, que possui `100` unidades de energia, está na posição `(1,1)` e não coletou material algum:

```
sampleRobot :: Robot
sampleRobot = Robot {
               energy = 100,
               position = (1,1),
               collected = 0
             }
```

2.1.1 Exercício 1.

Implemente uma instância de `Show` para o tipo de dados `Robot` de maneira que a função `show` quando aplicada a `sampleRobot` produza a seguinte string:

```
"Energy:100\nPosition(1,1)\nCollected:0"
```

2.2 Descrição de minas

Uma mina é descrita por um modelo de seu mapa. Um mapa descreve um conjunto de pontos da mina. Cada ponto da mina é descrito por seu conteúdo, que pode ser formado pelos seguintes tipos de elementos:

- Posições vazias, que permitem que o robô caminhe livremente.
- Entrada da mina, que pode ser utilizada para entrar e sair da mina.
- Paredes, que são intransponíveis. Robôs não podem atravessá-las ou removê-las.
- Terra, que pode ser removida pelo robô, ao custo de 5 unidades de energia.
- Rochas, que podem ser removidas pelo robô, ao custo de 30 unidades de energia.
- Materiais, que podem estar diferentes em concentrações no interior da mina.

A descrição de elementos da mina é representada pelo seguinte tipo de dados Haskell:

```
data Element = Empty         -- espaço vazio
             | Entry         -- entrada da mina
             | Wall          -- parede
             | Earth         -- terra
             | Rock          -- rocha
             | Material Int  -- material, Int indica quantidade.
             deriving (Eq,Ord)
```

Para facilitar, vamos utilizar a seguinte convenção de mneumônicos para representar elementos de uma mina:

Elemento	Mneumônico
espaço vazio	
Entrada da mina	`E`
Parede	`%`
Terra	`.`
Rocha	`*`
50 de Material	`?`
100 de Material	`:`
150 de Material	`;`
Outras quantidades	`$`

Outras quantidades de material devem ser consideradas como sendo uma unidade deste material.

2.2.1 Exercício 2.

Utilizando a tabela de menumônicos anterior, apresente uma definição de `Show` para o tipo de dados `Element`.

2.2.2 Exercício 3.

Apresente uma definição de um parser para o tipo `Element`.

```
pElement :: Parser Char Element
```

A partir do tipo `Element`, podemos descrever uma mina, como sendo uma matriz $n \times m$ de elementos, em que n representa o número de linhas e m de colunas. O tipo `Mine` é utilizado para descrever minas:

```
type Line = [Element]

data Mine = Mine {
              lines    :: Int,
              columns  :: Int,
              elements :: [Line]
            } deriving (Eq, Ord)
```

2.2.3 Exercício 4

Consideramos que um valor do tipo `Mine` é válido se a matriz de elementos possui o número de linhas e cada linha possui o número de colunas especificado pelos campos `lines` e `columns`. Além diso, uma mina deve ter pelo menos uma entrada e esta deve estar nas bordas da mina. Implemente a função `validMine`, que retorna verdadeiro se uma mina é ou não válida.

```
validMine :: Mine -> Bool
```

3 A linguagem de descrição de mina

Especificações LDM nada mais são que uma descrição textual de uma mina. O exemplo a seguir, ilustra uma especificação de uma mina de 15×15.

```
%%%%%%%%%%%%%%%
%***.........%
%***... ...*..%
%***... ..***.%
%.?.... ...*..%
%..      .. ...%
%.... .... ...%
%.:.. .... ...%
%.. .       ..%
%..*. .. .....%
%.... .. .;;..%
%.*.. ...;;..*%
%............$%
%.........   .%
%%%%%%%%%%%%%L%
```

3.0.1 Exercício 5

Apresente o valor do tipo `Mine` correspondente adescrição em LDM da mina 15×15 acima.

```
exampleMine :: Mine
```

3.0.2 Exercício 6

Implemente um parser para o tipo `Mine`, que a partir de uma descrição em LDM, retorne um valor deste tipo.

```
pLine :: Parser Char Line

pMine :: Parser Char Mine
```

3.0.3 Exercício 7:

Implemente uma instância de `Show` para o tipo de dados `Mine`, de maneira que a string produzida pela função `show` seja exatamente a especificação em LDM da mina.

4 A Linguagem de Comandos de Robô

Robôs apenas executam comandos LCR. A LCR é também uma linguagem de mneumônicos e possui apenas as seguintes intruções:

- `L`: Se o robô encontra-se na posição (x, y), a instrução `L`. faz com que o robô se mova para a posição $(x - 1, y)$.
- `R`: Se o robô encontra-se na posição (x, y), a instrução `R` faz com que o robô se mova para a posição $(x + 1, y)$.
- `U`: Se o robô encontra-se na posição (x, y), a instrução `U` faz com que o robô se mova para a posição $(x, y + 1)$.
- `D`: Se o robô encontra-se na posição (x, y), a instrução `D` faz com que o robô se mova para a posição $(x, y - 1)$.
- `C`: Essa instrução faz com que o robô colete material, caso exista material na vizinhança da posição atual do robô. Se o robô encontra-se na posição (x, y), a vizinhança é formada pelos seguintes pontos: $(x + 1, y)$, $(x - 1, y)$,$(x, y + 1)$ e $(x, y - 1)$. Depois de coletar material, esta posição deve ser atualizada para vazio (valor `Empty`).
- `S`: Essa instrução faz com que o robô permaneça parado por uma unidade de tempo. O efeito desta instrução é recarregar o robô em 1 unidade de energia.

Cada instrução de movimento consome 1 unidade de energia do robô. A instrução de coleta de materiais consome 10 unidades de energia.

O seguinte tipo de dados, representa instruções LCR:

```
data Instr = L -- move para esquerda
           | R -- move para direita
           | U -- move para cima
           | D -- move para baixo
           | C -- coleta material
           | S -- para para recarga.
           deriving (Eq,Ord,Show,Enum)
```

4.0.1 Exercício 8

Implemente um parser para o tipo `Instr`.

```
pInstr :: Parser Char Instr
pInstr = undefined
```

Programas LCR consistem apenas de uma sequência de instruções. Considera-se que um programa executa com sucesso se o robô entra na mina e sai por uma das entradas desta.

4.0.2 Exercício 9

Um programa LCR consiste de uma string de mneumônicos sem espaços. Desta forma, programas podem ser vistos como uma lista de instruções. Implemente um parser para programas LCR.

```
pProgram :: Parser Char [Instr]
pProgram = undefined
```

4.1 Atualização da Mina

Note que ao executar uma instrução, a mina deve ser atualizada de maneira apropriada. Instruções executadas com sucesso transformam a posição atual no robô em uma posição vazia. Dizemos que instruções são executas com sucesso se:

- A instrução `S` é sempre executada com sucesso.
- Instruções de movimento são executadas se:
  - O robô possui energia suficiente para executá-las.
  - A posição de destino do movimento não é uma parede da mina.
- A instrução de coleta é executada com sucesso se o robô possui energia suficiente e a vizinhança da posição atual do robô possui materiais. A vizinhança de um ponto (x, y) é formada pelo seguinte conjunto de pontos $\{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$. A posição vizinha que possuir material coletado deve ser convertida para uma posição vazia.

Se uma instrução não pode ser executada com sucesso, o robô executa a instrução `S` e tenta executar a próxima instrução do programa.

Para permitir o fluxo de alterações de valores do tipo de dados `Mine`, utilizaremos uma mônada de estado para armazenar a configuração atual da execução do programa que é composta pelo valor atual do robô e o valor atual da mina.

```
type Conf = (Robot, Mine)

type ConfM a = State Conf a
```

4.1.1 Exercício 10

Utilizando a mônada `ConfM`, implemente as seguintes funções utilizadas para se obter componentes da configuração:

- A função `current` que retorna a posição atual do robô na mina.
- A função `mine` que retorna a configuração atual da mina.
- A função `enoughEnergy`, que retorna verdadeiro se o valor de energia atual do robô é maior que o inteiro fornecido como parâmetro.
- A função `incEnergy`, que incrementa por 1 o valor de energia atual do robô.

```
current :: ConfM Point
current = undefined

mine :: ConfM Mine
mine = undefined

enoughEnergy :: Int -> ConfM Bool
enoughEnergy = undefined

incEnergy :: ConfM ()
incEnergy = undefined
```

4.1.2 Exercício 11

Defina a função `valid :: Instr -> ConfM Bool` que determina se uma instrução é ou não válida de acordo com as regras anteriores.

```
valid :: Instr -> ConfM Bool
valid = undefined
```

4.1.3 Exercício 12

Implemente a função `updateMine :: Instr -> ConfM ()`, que a partir de uma instrução, atualiza a configuração da mina, caso esta seja válida.

```
> updateMine :: Instr -> ConfM ()
> updateMine = undefined
```

4.2 Execução de Instruções

De posse de funções para determinar quando instruções são válidas e para atualizar uma certa posição da mina, podemos definir a função que simula a execução de um robô em uma dada mina. Caso a instrução seja válida, atualiza-se o robô e a mina de maneira apropriada, caso contrário, a instrução executada deve ser `S`.

4.2.1 Exercício 13

Implemente a função `exec` que executa uma instrução LCM, caso esta seja válida, e atualiza a mina logo após a execução com sucesso desta.

```
> exec :: Instr -> ConfM ()
> exec = undefined
```

4.2.2 Exercício 14

Implemente a função `initRobot`, que a partir de um valor do tipo `Mine`, retorne uma configuração inicial do robô explorador. Esta configuração inicial deve atribuir um valor de 100 unidades de energia ao robô, como posição inicial deste, a entrada da mina e como valor inicial de material coletado, 0.

```
initRobot :: Mine -> Robot
initRobot = undefined
```

4.2.3 Exercício 15

Implemente a função `run`, que executa um programa LCM sobre uma dada mina, retornando a configuração final desta como resultado. Esta função deve receber como parâmetros o programa e a mina a ser explorada.

```
> run :: [Instr] -> Mine -> Mine
> run = undefined
```

5 Interface com o usuário

5.0.1 Exercício 16

Implemente uma função para ler arquivos ".ldm", contendo descrições de mina, retornando um valor de tipo `Mine` ou uma mensagem de erro indicando que não foi possível realizar a leitura deste arquivo.

```
readLDM :: String -> IO (Either String Mine)
readLDM = undefined
```

5.0.2 Exercício 17

Implemente uma função para ler arquivos "lcr", contendo descrições de comandos de robôs, retornando um valor do tipo `[Instr]` ou uma mensagem de erro indicando que não foi possível realizar a leitura deste arquivo.

```
readLCR :: String -> IO (Either String [Instr])
readLCR = undefined
```

Finalmente, a seguinte função chama as anteriores para executar os comandos de robô especificados por um arquivo lcr sobre a mina descrita por um arquivo lcm, imprimindo o resultado final da mina.

```
main :: IO ()
main = do
          args <- getArgs
          exec args

exec :: [String] -> IO ()
exec [fm , fr]
   = do
        pm <- readLDM fm
        pr <- readLCR fr
        let
           m = either error id pm
           r = either error id pr
           m' = run r m
        print m'
exec _ = putStrLn "Informe dois arquivos de entrada!"
```

6 Considerações Finais

- Este trabalho pode ser resolvido por grupos de até 3 alunos.
- Plágios não serão tolerados. Qualquer tentativa de plágio indentificada será punida com ZERO para todos os envolvidos. Lembre-se que é melhor entregar uma solução incompleta ou incorreta.
- Entrega deverá ser feita usando o Moodle até o dia 14/10/2022. Você deverá entregar somente um arquivo .zip contendo todo o projeto stack de sua solução.