Paradigmas de Programação

Fabrício Olivetti de França

31 de Julho de 2018

Quebra-cabeça da Zebra

Um famoso quebra-cabeça de grade conhecido como Jogo da Zebra ou Quebra-cabeça de Einstein começa com as seguintes proposições:

1. Existem cinco casas.
2. O inglês mora na cada vermelha.
3. O Sueco tem um cachorro.
4. O dinamarquês bebe chá.
5. A casa verde está imediatamente a esquerda da casa branca.
6. Eles bebem café na casa verde.
7. O homem que fuma Pall Mall tem um pássaro.
8. Na casa amarela eles fumam Dunhill.
9. Na casa do meio eles bebem leite.
10. O norueguês mora na primeira casa.
11. O homem que fuma Blend mora na casa ao lado da casa com gatos.
12. Em uma casa ao lado da casa com um cavalo, eles fumam Dunhill.
13. O homem que fuma Blue Master bebe cerveja.
14. O Alemão fuma Prince.
15. O norueguês mora na casa ao lado da casa azul.
16. Eles bebem água na casa ao lado da casa de quem fuma Blend.

O objetivo é descobrir quem é o dono da Zebra.

Vamos resolver essa questão utilizando os conceitos de Monads aprendidos na aula passada.

1. Crie um novo projeto com o `stack` chamado *zebra* e inclua o seguinte código no arquivo *Main.hs*:

```
module Main where

import Control.Monad
import Control.Applicative
import Data.List
```

```haskell
-- gera todas as permutações de um tipo da classe Bounded
perms :: (Bounded a, Enum a) => [[a]]
perms = permutations [minBound..maxBound]
```

2. Crie os tipos *Nacao, Cor, Pet, Bebida, Cigarro* derivando as classes *Bounded, Enum, Eq, Show*.

3. Defina `nacoes, cores, pets, bebidas, cigarros` como as permutações dos valores de cada tipo. Para isso verifique no ghci como `perms` se comporta ao definir o tipo `a`, para explorar, carregue o código no *ghci* e digite:

```haskell
> perms :: [[Nacao]]
```

4. Todas as dicas do problema ou envolvem um tipo (ex.: o Norueguês mora na primeira casa) ou dois (ex.: o Dinamarques tem um cachorro). Vamos contar quantas dicas tem de cada relação:

	Nacao	Cor	Pet	Bebida	Cigarro
Nacao	1	2	1	1	1
Cor		1		1	1
Pet			0		3
Bebida				1	2
Cigarro					0

Percebemos que a relação com maior número de dicas é Pet-Cigarro. Defina uma variável `solucoes` utilizando o *do-notation* para receber um `pet` da lista de permutações de `pets` e um cigarro da lista de permutações de `cigarro` e retorne a tupla `(pet, cigarro)`. Na função `main` faça:

```haskell
main :: IO ()
main = do print (take 3 solucoes)
```

5. Começando pela dica 7, temos que a mesma casa que fuma PallMall tem um Pássaro, vamos criar uma função que verifica se dois elementos estão na mesma posição nas listas de permutação correspondente:

```haskell
mesmaCasa :: (Eq a, Eq b) => a -> [a] -> b -> [b] -> Bool
mesmaCasa x' [] y' ys = False
mesmaCasa x' xs y' [] = False
mesmacasa x' (x:xs) y' (y:ys) = ???
```

A ideia é que se `x'==x` e `y'==y` então estão na mesma casa, caso contrário devemos verificar o restante da lista. Note que existem duas situações que retornamos falso e não precisamos olhar o restante da lista. Teste no *ghci* com algum exemplo fabricado.

6. Para filtrar os resultados com esse predicado, utilizaremos a função `guard`:

```haskell
resultado = do x1 <- lista1
```

```
    x2 <- lista2
    -- se o retorno for verdadeiro,
    -- devolve (x1,x2),
    -- senão passa para o próximo item
    guard $ predicado x1 x2
    return (x1,x2)
```

Filtre os resultados para `mesmaCasa PallMall cigarro Passaro pet` e verifique o resultado.

7. As dicas 11 e 12 verificam se um está na casa ao lado do outro, se `x` está ao lado de `y`, ou `x` está a esquerda de `y` ou vice-versa. Como teremos algumas dicas que verificam se um está a esquerda do outro, vamos começar definindo:

```
aEsquerdaDe :: (Eq a, Eq b) => a -> [a] -> b -> [b] -> Bool
```

Veja que para verificar se `x` está a esquerda de `y`, podemos verificar se eles estão na mesma casa ao deslocar `ys` para esquerda.

Agora podemos definir `aoLadoDe` utilizando `aEsquerdaDe`:

```
aoLadoDe :: (Eq a, Eq b) => a -> [a] -> b -> [b] -> Bool
aoLadoDe x xs y ys = (aEsquerdaDe x xs y ys) || (aEsquerdaDe y ys x xs)
```

Complemente o `guard` para incluir os predicados 11 e 12 e verifique o resultado até então.

8. Capture as permutações de `nacao` e verifique os predicados 3 e 14. Implemente a função `primeiraCasa` que verifica se um item está na primeira posição da permutação e verifique o predicado 10.
9. Capture as permutações de `cor` e verifique os predicados 2, 5, 8 e 15.
10. Capture as permutações de `bebida` e verifique os predicados 4, 6, 13, 16. Implemente uma função `casaDoMeio` para o predicado 9.
11. Altere a função `main` para:

```
main :: IO ()
main = do print (head [zip5 n c p b cig | (n,c,p,b,cig) <- solucoes])
```

E responda quem tem a Zebra.