Princípios de Programação

Trabalho para casa 4

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2018/2019

A linguagem Haskell tem o seu avaliador de expressões aritméticas. Se baterem `3 + 1` no `ghci`, o interpretador calcula o valor e apresenta o resultado. Neste trabalho vamos definir e avaliar as nossas próprias expressões aritméticas. As expressões numéricas podem ter diversas utilizações, pelo que vamos construir um módulo com algumas funções para construir e avaliar expressões aritméticas.
O módulo deverá chamar-se `Expressao` e deverá estar no ficheiro `Expressao.hs` (sem til!). Este módulo deverá exportar pelo menos os seguintes tipos de dados e funções:

1. O tipo de dados `Expr` mas não os seus construtores, de modo a que `Expr` seja um tipo de dados abstrato.

2. A função `constante :: Int -> Expr` que recebe um valor inteiro e devolve uma expressão que representa esse valor inteiro.

   ```
   ghci> :t constante 3
   constante 3 :: Expr
   ```

3. O operador `|+|`, com o tipo `(|+|) :: Expr -> Expr -> Expr`, que recebe duas expressões e devolve uma nova expressão que representa a soma das duas.

   ```
   ghci> :t constante 3 |+| constante 1
   constante 3 |+| constante 1 :: Expr
   ```

4. O operador `|*|`, com o tipo `(|*|) :: Expr -> Expr -> Expr`, que recebe duas expressões e devolve uma nova expressão que representa o produto das duas.

```
ghci> :t constante 3 |*| constante 1
constante 3 |*| constante 1 :: Expr
```

5. A função `avalia :: Expr -> Int`, que recebe uma expressão e realiza os cálculos necessários para devolver um inteiro.

   ```
   ghci> avalia $ constante 4 |+| (constante 3 |*| constante 2)
   10
   ```

   Note que até à chamada desta função, nenhuma operação é de facto realizada! As funções anteriores servem apenas para *construir* expressões.

6. Garanta que o tipo de dados `Expr` é instância de **`Show`** de modo a que transforme a expressão em notação matemática comum. Deve utilizar sempre parêntesis em redor de cada expressão não constante e espaços em redor de cada operador binário. Por exemplo:

   ```
   ghci> constante 4 |+| (constante 3 |*| constante 2)
   (4 + ( 3 * 2))
   ```

7. Garanta que o tipo de dados `Expr` é também instância de **`Eq`**. Duas expressões são iguais quando o valor da sua avaliação é também ele igual.

   ```
   ghci> constante 4 |+| constante 1 == constante 5
   True
   ```

De modo a facilitar o teste do módulo `Expressao` pode usar o ficheiro abaixo como cliente. Deverá submeter *dois* ficheiros: o `Expressao.hs` e o ficheiro abaixo. Este último deverá chamar-se `t4_fcXXXXX.hs`, onde `XXXXX` é o número de aluno.

```
import Expressao

expr1 :: Expr
expr1 = constante 3 |+| (constante 10 |*| constante 3)
expr2 :: Expr
expr2 = constante 3 |*| (constante 10 |+| constante 1)

main :: IO ()
main = do
  putStrLn $ show expr1 ++ "␣=␣" ++ show (avalia expr1)
  putStrLn $ show expr2 ++ "␣=␣" ++ show (avalia expr2)
  putStrLn $ show $ expr1 == expr2
  putStrLn $ show $ expr1 == constante 0
```

O resultado esperado da execução deste ficheiro é:

```
(3 + (10 * 3)) = 33
(3 * (10 + 1)) = 33
True
False
```

Notas

1. Os trabalhos serão avaliados automaticamente. Respeite o nome do módulo e das funções exportadas.
2. Não se esqueça de juntar uma assinatura para cada função que escrever.
3. Lembre-se que as boas práticas de programação Haskell apontam para a utilização de várias funções simples em lugar de uma função única mas complicada.
4. A precedência dos operadores não deve ser tomada em consideração. O uso de parêntesis é obrigatório em redor dos novos operadores.

Entrega. Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 21 de novembro de 2018.

Ética Os trabalhos de todos os alunos serão comparados por uma aplicação computacional. Lembre-se: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".