Programação Funcional

Fundamentos

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

Operadores Aritméticos

- Em Haskell, expressões utilizando os operadores aritméticos binários podem ser escritas tanto em forma prefixada quanto na forma infixada:

```
ghci> (+) 6 3        -- Forma prefixada da expressão 6 + 3
```

- Além da adição, a subtração, a multiplicação, a divisão e a exponenciação também estão disponíveis

```
ghci> (-) 6 3        -- 6 - 3 = 3
ghci> (*) 6 3        -- 6 * 3 = 18
ghci> (/) 6 3        -- 6 / 3 = 2.0
```

- O operador ^ representa a divisão em ponto flutuante, não inteira
- Os resultados não geram *overflow*, sendo a aritmética estendida implementada nativamente

```
ghci> 2 ^ 70         -- 2 ^ 70 = 1180591620717411303424
ghci> 2 ** 70        -- pow(2, 70) = 1.1805916207174113e21
```

- Números negativos devem vir entre parêntesis, para evitar ambiguidades

```
ghci> 6 - (-3)       -- 6 + 3 = 9
```

Operadores lógicos e relacionais

- Em Haskell os valores booleanos são **`True`** e **`False`**
- Os operadores lógicos são: **e** (`&&`), **ou** (`||`) e **não** (`not`)
- Os operadores relacionais são: **igual** (`==`), **diferente** (`/=`), **menor** (`<`), **menor ou igual** (`<=`), **maior** (`>`) e **maior ou igual** (`>=`)
- No ghci, a precedência dos operadores pode ser consultada por meio do comando '`:info`'
- O valor 1 significa a menor precedência possível; 9 é a maior precedência possível

```
ghci> :info (+)     -- infixl 6 +
ghci> :info (*)     -- infixl 7 *
ghci> :info (^)     -- infixl 8 *
```

Listas

- As listas são tipos primitivos em Haskell
- Elas são declaradas por meio de colchetes, e os seus elementos são separados por vírgulas

```
ghci> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

- A lista vazia é representada por `[]`
- Todos os elementos de uma lista devem ser do mesmo tipo
- Haskell permite uma notação que permite a enumeração dos elementos da lista

```
ghci> [1..5]                -- [1, 2, 3, 4, 5]
ghci> [2, 5..18]            -- [2, 5, 8, 11, 14, 17]
```

- Listas podem ser concatenadas por meio do operador `++`

```
ghci> [5..6] ++ [1..4]      -- [5, 6, 1, 2, 3, 4]
```

- Um elemento pode ser adicionado ao início de uma lista por meio do operador cons `(:)`

```
ghci> 1 : [2..4]            -- [1, 2, 3, 4]
```

Caracteres e strings

- Um caractere é delimitado por aspas simples

```
ghci> 'a'
```

- Uma string (de caracteres) é delimitada por aspas duplas

```
ghci> "Exemplo de string"
```

- Efetivamente, uma string é uma lista de caracteres

```
ghci> "ABC" == ['A', 'B', 'C']                    -- True
```

- Vale a igualdade: `"" == []`
- Observe também que `"A"` e `'A'` tem tipos distintos
- Como as strings são listas, a notação de enumeração pode ser utilizada:

```
ghci> ['a'..'z'] == "abcdefghijklmnopqrstuvwxyz"    -- True
```

Tipos de dados em Haskell

- O **tipo** de um dado é uma abstração sobre a cadeia de *bytes* que armazena o valor da variável ou constante
- Haskell é uma linguagem com tipagem de dados **forte** e **estática**, onde os tipos das expressões pode ser inferidos **automaticamente**
- Em um sistema de tipagem estática, os tipos dos dados e das expressões devem ser conhecidos em tempo de compilação
- Em um sistema forte, as regras identificação, conversão e validação dos tipos são estritas e aplicadas em tempo de compilação
- Em Haskeel, se uma expressão violar as regras de tipagem ela será considerada mal formada, e levará a um erro de tipo
- Também não há promoção de tipos ou conversões implícitas de tipos dentro de uma expressão

Tipos de dados em Haskell

- Conversões entre tipos envolvem cópias, o que pode impactar na performance dos programas
- A combinação de tipagem forte e estática faz com que os erros de tipos em Haskell jamais aconteçam em tempo de execução
- O fato de ter um sistema forte e estático torna Haskell uma linguagem segura; a inferência de tipos a torna uma linguagem concisa
- A convenção em Haskell é que tipos de dados iniciem em letras maísculas, e as variáveis iniciem em letra minúscula
- No ghci, o tipo de uma expressão pode ser determinado por meio do comando `:type`
- A assinatura de um tipo é

```
expression :: Type
```

Scripts

- Programas em Haskell também podem ser escritos em arquivos, chamados **scripts**
- Estes *scripts* podem ser interpretados pelo programa runghc, ou compilados pelo ghc
- O *script* abaixo reproduz parcialmente o comportamento do comando wc do Linux, que conta o número de palavras da entrada:

```
-- Para rodar use o comando
--      $ runghc wc.hs
-- ou compile com o comando
--      $ ghc wc.hs
main = interact wc
    where wc input = ((show . length . words) input) ++ "\n"
```

- Comentários iniciam com dois traços (--)

Referências

1. **BARENDREGT**, Henk; **BARENDSEN**, Erik. *Introduction to Lambda Calculus*, March 2000.
2. **SHALOM**, Elad. *A Review of Programming Paradigms Througout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.
3. **SULLIVAN**, Bryan O.; **GOERZEN**, John; **STEWART**, Don. *Real World Haskell*, O'Relly.