

Programação Funcional

Funções de alta ordem

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

1. Funções de alta ordem

Exemplos de funções das bibliotecas do Haskell

1. A função `lines` recebe uma string e retorna um vetor de strings, o qual corresponde às linhas contidas na string original

```
ghci> :type lines
lines :: String -> [String]

ghci> lines "Hello\nWorld"
["Hello", "World"]
```

2. A função `unlines` é sua inversa: ela recebe um vetor de strings, e une todas elas em uma única string, adicionando o terminador de linha (`'\n '`) entre elas

```
ghci> :type unlines
unlines :: [String] -> String

ghci> unlines ["a", "b", "c"]
"a\nb\nc\n"
```

3. A função `last` retorna o último elemento da lista

```
ghci> last "ABC"
'C'
```

Exemplos de funções das bibliotecas do Haskell

4. A função complementar de `last` é a função `init`, que retorna todos, menos o último, elementos da lista

```
ghci> init "ABCDE"  
"ABCD"
```

5. A função `(++)` une duas listas em uma única lista

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

6. A função `concat` generaliza este comportamento, recebendo uma lista de listas e as concatenando em uma única lista

```
ghci> :type concat  
concat :: [[a]] -> [a]
```

```
ghci> concat ["um", "dois", "tres"]  
"umdoistres"
```

Exemplos de funções das bibliotecas do Haskell

7. A função `reverse` recebe uma lista `xs` e retorna uma nova lista, com todos os elementos de `xs` em ordem inversa

```
ghci> reverse [1..5]
[5, 4, 3, 2, 1]
```

8. As funções `and` e `or` aplicam as operações lógicas binárias (`&&`) e (`||`) em todos os elementos da lista, até que reste apenas um elemento

```
ghci> and [True, False, True]
False
```

```
ghci> or [True, False, True]
True
```

9. A função `splitAt` recebe um inteiro `i` e uma lista `xs`, e retorna um par de listas (`xs[1..i]`, `xs[(i+1)..i]`)

Funções infixadas

- ▶ Haskell utiliza, por padrão, a notação prefixada, de modo que, na aplicação da função `f` aos argumentos `x` e `y`, o nome da função precede os argumentos, que são separados por espaços em branco
$$z = f\ x\ y$$
- ▶ Se a função recebe dois ou mais argumentos, é possível que a notação infixada traga uma melhor compreensão e leitura
- ▶ Para utilizar a notação infixada, basta colocar o nome da função entre crases (```), tanto em uma definição quanto em uma chamada
- ▶ Ambas formas são intercambiáveis

```
import Data.Bits
```

```
bitwise_or :: Int -> Int -> Int
```

```
bitwise_or a b = a .|. b
```

```
main = print (x, y) where           -- saída: (3, 7)
```

```
    x = bitwise_or 1 2
```

```
    y = 3 `bitwise_or` 5
```

Exemplos de funções infixadas

1. A função `elem` recebe um elemento `x` e uma lista de elementos `xs` e retorna verdadeiro se `x` pertence a `xs`

```
ghci> :type elem
elem :: a -> [a] -> Bool
```

```
ghci> 'x' `elem` "Teste"
False
```

2. A negação de `elem` é a função `notElem`

```
ghci> 'x' `notElem` "Teste"
True
```

3. A função `isPrefixOf` do módulo `Data.List` recebe os mesmos parâmetros, e retorna verdadeiro se `x` é prefixo de `xs`
4. As funções `isInfixOf` e `isSuffixOf` do mesmo módulo tem comportamento semelhante, retornando verdadeiro se `x` é uma sublista de `xs` ou se `x` é sufixo de `xs`, respectivamente

Funções de alta ordem

- ▶ Uma função é dita de **alta ordem** se ela recebe uma ou mais funções como parâmetro ou retorna uma função
- ▶ Por exemplo, a função **break** recebe um predicado **P** e uma lista **xs**, e retorna uma par de listas (**ys**, **zs**), onde **xs = ys ++ zs** e **zs** tem início no primeiro elemento **x** de **xs** tal que a expressão '**P x**' é verdadeira

```
ghci> :type break
break :: (a -> Bool) -> [a] -> ([a], [a])
```

```
ghci> break even [1, 1, 2, 3, 5, 8]
([1, 1], [2, 3, 5, 8])
```

- ▶ A função **all** recebe um predicado **P** e uma lista **xs** e retorna verdadeiro se '**P x**' é verdadeira para todos **x** em **xs**
- ▶ A função **any** recebe os mesmos parâmetros, e retorna verdadeiro se '**P x**' é verdadeira para ao menos um elemento de **xs**

Exemplos de funções de alta ordem

1. A função `takeWhile` recebe um predicado `P` e uma lista `xs` e retorna uma lista `ys` cujos elementos são todos dentre os primeiros elementos `x` de `xs` tais que '`P x`' é verdadeira
2. Sua complementar é a função `dropWhile`, que recebe os mesmo parâmetros e retorna uma lista `ys` cujo primeiro elemento é o primeiro elemento `x` de `xs` para o qual a expressão '`P x`' é falsa

```
Prelude Data.Char> takeWhile isUpper "FGAmADF"  
"FGA"
```

```
Prelude Data.Char> dropWhile isUpper "FGAmADF"  
"maDF"
```

3. A função `span` retornam um par de listas com as duas partes resultantes da chamada de `takeWhile`

```
Prelude Data.Char> span isUpper "FGAmADF"  
("FGA", "maDF")
```

Referências

1. **SHALOM**, Elad. *A Review of Programming Paradigms Throughout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.
2. **SULLIVAN**, Bryan O.; **GOERZEN**, John; **STEWART**, Don. *Real World Haskell*, O'Reilly.