Generalização

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

1 de Junho de 2017

Funções de Alta Ordem

Maps e Filters

Lambdas

Funções folds

Funções de Alta Ordem

- Já sabemos que em Haskell uma função pode receber funções como parâmetros e retorna outras funções.
- Esse tipo de função é chamada de Função de Alta Ordem.
- Também entendemos que a *currificação* de funções permite retornar funções parciais.

```
aplicarDuasVezes :: (a -> a) -> a -> a
aplicarDuasVezes f x = f ( f x )
```

Função Misteriosa

```
misterio :: (a -> b -> c) -> [a] -> [b] -> [c]
misterio _ [] _ = []
misterio _ _ [] = []
misterio f (x:xs) (y:ys) = f x y : misterio f xs ys
```

O que essa função faz ?

Maps e *Filters*

Aplicar uma função a cada elemento de uma lista.

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

A função *map* torna a escrita de compreensão de listas mais elegante.

Verificar se cada elemento da lista atende ou não uma condição.

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter p (x:xs)
    | p x       = x : filter p xs
    | otherwise = filter p xs
```

Exemplo de *filter*

```
maiorDivisor :: (Integral a) => a
maiorDivisor = head (filter p [100000,99999..])
  where p x = x `mod` 3829 == 0
```

A função *takehile* é um *filter* que para quando encontra um elemento que retorna False.

```
takeWhile (/=' ') "primeira segunda terceira quarta"
sum (takeWhile (<10000) (filter odd (map (^2) [1..])))
sum (takeWhile (<10000) [n^2 | n <- [1..], odd (n^2)])
```

Sequências de Collatz

- Dado um número natural, se for par, divida por 2. Se é ímpar, multiplicar por 3 e adicionar 1.
- Aplicar as mesmas regras para o resultado.
- Supõe-se que não importa o número inicial, um dia a lista termina em 1.

Pergunta: para todos os números iniciais entre 1 e 100, quantas sequências de Collatz existem com comprimento menor do que 15?

Lambdas

- Lambdas são funções anônimas.
- São criadas para servirem de parâmetro para funções de alta ordem.
- No lugar do nome, usamos a barra \, seguida pelos parâmetros.
- Neste caso o símbolo $->$ serve para iniciar o corpo da função.
- É boa prática envolver a lambda em parênteses.

```
resultado = length (filter
    (\xs -> length xs > 15)
        (map collatz [1..100]))
```

Funções *fold*

- Funções *fold* recebe uma função binária, um valor inicial e uma lista.
- A função *fold* automaticamente destrincha o padrão (x:xs) e aplica a função binária no valor inicial e em x.
- A lista é percorrida e o valor inicial é chamado de acumulador.

```
-- Soma
soma :: (Num a) => [a] -> a
soma xs = foldl (\acc x -> acc + x) 0 xs
-- Função de Alta Ordem para Soma
soma :: (Num a) => [a] -> a
soma = foldl (+) 0
-- Lambdas e folds, juntas!!!
pertence :: (Eq a) => a -> [a] -> Bool
pertence y ys =
   foldl (\acc x -> if x == y then True else acc)
      False ys
```

Mais Exemplos de fold

```haskell
maximo :: (Ord a) => [a] -> a
maximo = foldr1 (\x acc -> if x > acc then x else acc)

reversa :: [a] -> [a]
reversa = foldl (\acc x -> x : acc) []

produto :: (Num a) => [a] -> a
produto = foldr1 (*)

filtrar :: (a -> Bool) -> [a] -> [a]
filtrar p =
          foldr
            (\x acc -> if p x then x : acc else acc) []

cabeca :: [a] -> a
cabeca = foldr1 (\x _ -> x)

ultimo :: [a] -> a
ultimo = foldl1 (\_ x -> x)
```

Funções *scan*

As funções *scanr* e *scanl* são como *foldr* e *foldl*, mas retornam os valores intermediários dos acumuladores em uma lista.

```
scanl (+) 0 [3,5,2,1]
[0,3,8,10,11]
scanr (+) 0 [3,5,2,1]
[11,8,3,1,0]
scanl1
   (\acc x -> if x > acc then x else acc)
      [3,4,5,3,7,9,2,1]
[3,4,5,5,7,9,9,9]
scanl (flip (:)) [] [3,2,1]
[[],[3],[2,3],[1,2,3]]
```

Exercício

Usando as funções *takewhile*, *scanl1* e *map*, crie uma função *somaRaizes* que diga quantos elementos são necessários a partir de 1 para que a soma das raízes seja maior que 1000?
Na boa e velha matemática:

$$somaQuadrados(n) = \sqrt{1} + \sqrt{2} + \sqrt{3} + ... + \sqrt{n}$$
$$somaQuadrados(n) > 1000, n = ?$$