

Programação Funcional

Generalizações

Vander Alves

Funções de alta ordem

- Funções como argumentos ou como resultado de outras funções.
- Permite
 - definições polimórficas
 - funções aplicadas sobre uma coleção de tipos
 - padrões de recursão usados por várias funções.

Exemplos

```
double :: [Int] -> [Int]
```

```
double [] = []
```

```
double (a:x) = (2*a) : double x
```

```
sqrList :: [Int] -> [Int]
```

```
sqrList [] = []
```

```
sqrList (a:x) = (a*a) : sqrList x
```

Funções de mapeamento (*mapping*)

Exemplos

```
times2 :: Int -> Int
```

```
times2 n = 2 * n
```

```
sqr :: Int -> Int
```

```
sqr n = n * n
```

Funções de transformação dos elementos

A função de mapeamento

- Recebe como argumentos
 - a transformação a ser aplicada a cada elemento da lista
 - uma função
 - a lista de entrada

map

```
map :: (t -> u) -> [t] -> [u]
```

```
map f [] = []
```

```
map f (a:as) = f a : map f as
```

```
doubleList xs = map times2 xs
```

```
sqrList xs = map sqr xs
```

```
snds :: [(t,u)] -> [u]
```

```
snds xs = map snd xs
```

```
map length ["abc", "defg"] = ?
```

Outra definição para map

```
map f l = [f a | a <- l]
```

Por que funções de alta ordem

- Facilita entendimento das funções
- Facilita modificações (mudança na função de transformação)
- Aumenta reuso de definições/código
 - modularidade
 - usar a função `map` para remarcar o valor de uma lista de preços

Exemplo: análise de vendas

```
total :: (Int->Int) -> Int -> Int
```

```
total f 0 = f 0
```

```
total f n = total f (n-1) + f n
```

```
totalSales n = total sales n
```

```
sumSquares :: Int -> Int
```

```
sumSquares n = total sq n
```

Outros exemplos

```
maxFun :: (Int -> Int) -> Int -> Int
```

```
maxFun f 0 = f 0
```

```
maxFun f n = maxi (maxFun f (n-1)) (f n)
```

```
zeroInRange :: (Int -> Int) -> Int -> Bool
```

```
zeroInRange f 0 = (f 0 == 0)
```

```
zeroInRange f n = zeroInRange f (n-1)  
                  || (f n == 0)
```

Exercício

- Use a função `maxFun` para implementar a função que retorna o maior número de vendas de uma semana de 0 a n semanas

`maxSales :: Int -> Int`

- Dada uma função, verificar se ela é crescente em um intervalo de 0 a n

`isCrescent :: (Int -> Int) -> Int -> Bool`

Exemplo: *folding*

`sumList :: [Int] -> Int`

`sumList [] = 0`

`sumList (a:as) = a + sumList as`

`e1 + e2 + ... + em`

`fold :: (t -> t -> t) -> [t] -> t`

`fold f [a] = a`

`fold f (a:as) = f a (fold f as)`

`sumList l = fold (+) 1`

Exemplo: *folding*

```
and :: [Bool] -> Bool
```

```
and xs = fold (&&) xs
```

```
concat :: [[t]] -> [t]
```

```
concat xs = fold (++) xs
```

```
maximum :: [Int] -> Int
```

```
maximum xs = fold maxi xs
```

Exemplo: *folding*

```
fold (||) [False, True, True]
```

```
fold (++) ["Bom", " ", "Dia"]
```

```
fold min [6]
```

```
fold (*) [1..6]
```

foldr

```
foldr :: (t -> u -> u) -> u -> [t] -> u
```

```
foldr f s [] = s
```

```
foldr f s (a:as)  
    = f a (foldr f s as)
```

```
concat :: [[t]] -> [t]
```

```
concat xs = foldr (++) [] xs
```

```
and :: [Bool] -> Bool
```

```
and bs = foldr (&&) True bs
```

Exemplo: filtrando

```
digits, letters :: String -> String

filter :: (t -> Bool) -> [t] -> [t]
  filter p [] = []
  filter p (a:as) | p a = a : filter p as
                  | otherwise = filter p as

digits st = filter isDigit st

letters st = filter isLetter st

evens xs = filter isEven xs
  where isEven n = (n `mod` 2 == 0)
```


outra definição para `filter`

```
filter p l = [a | a <- l, p a]
```

Exercícios

- Defina as seguintes funções sobre listas
 - eleva os itens ao quadrado
 - *mapping*
 - retorna a soma dos quadrados dos itens
 - *folding*
 - manter na lista todos os itens maiores que zero.
 - *filtering*

Polimorfismo

- Função possui um tipo genérico
- Mesma definição usada para vários tipos
- Reuso de código
- Uso de variáveis de tipos

```
zip :: [t] -> [u] -> [(t,u)]  
zip (a:as) (b:bs) = (a,b):zip as bs  
zip _ _ = []
```

Polimorfismo

```
length [] = 0
```

```
length (a:as) = 1 + length as
```

```
rev [] = []
```

```
rev (a:as) = rev as ++ [a]
```

```
id x = x
```

- Funções com várias instâncias de tipo

Polimorfismo

```
rep 0 ch = []
```

```
rep n ch = ch : rep (n-1) ch
```

- Haskell: inferência de tipos

```
:type rep
```

```
Int -> a -> [a]
```

Exemplo: Biblioteca

```
books :: Database -> Person -> [Book]
books db per = map snd (filter isPer db)
  where isPer (p,b) = (p == per)

returnLoan :: Database -> Person
              -> Book -> Database
returnLoan db p b = filter notPB db
  where notPB pr = (pr /= (p,b))
```