PROGRAMAÇÃO FUNCIONAL: Funções de Ordem Superior

Professor Rafael Kingeski

Departamento de Ciência da Computação
Centro de Ciências Tecnológicas - CCT
UDESC - Joinville.

Índice

Índice

Funções Genéricas sobre Listas

- Mapeamento - Função aplicada sobre cada elemento da lista de modo que retorne outra lista modificada.
- Filtragem - Condição aplicada em uma lista de modo que retorne os valores filtrados nessa condição.
- Redução - Função aplicada em todos os valores de modo que retorne apenas um único valor.

Índice

Funções de Ordem Superior

Um dos principais diferenciais da programação funcional é a ideia de que funções podem se tornar argumentos de outras funções. Essa ideia de combinar funções generalizadoras resulta em funções de ordem superior.

Mapear

Uma função para calcular o dobro de uma lista pode ser implementada por:

```
vezes2 [] = []
vezes2 (x:xs) = 2*x : vezes2 xs
```

Uma função para calcular o triplo de uma lista pode ser implementada por:

```
vezes3 [] = []
vezes3 (x:xs) = 3*x : vezes3 xs
```

Uma função para calcular o quádruplo de uma lista pode ser implementada por:

```
vezes4 [] = []
vezes4 (x:xs) = 4*x : vezes4 xs
```

Mapear

Podemos definir uma função que aplica uma transformação a cada elemento de uma lista de uma forma genérica, assim podemos aplicar qualquer função em uma lista sem precisar implementa-la para cada caso.

Definimos então as funções multiplicação por 2, 3 e 4:

```
mult2 x = 2*x
mult3 x = 3*x
mult4 x = 4*x
```

Mapear


```
dobrol = mapear mult2 lista
triplol = mapear mult3 lista
quadrl = mapear mult4 lista
```

Definimos então a função mapear:

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

Uma outra forma de aplicar a função mapear é utilizando compreensão de listas:

```
mapear2 = [(f x) | x <- lista]
```

Filtrar

Se desejamos encontrar um valor maior que 1 em uma lista podemos implementar a função:

```
maior1 [] = []
maior1 (x:xs) | x > 1 = x : maior1 xs
              |otherwise = maior1 xs
```

Filtrar

Podemos definir a função filtrar como:

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

Reduzir

Se definimos a função reduzir como uma função que recebe uma lista e retorna apenas um elemento, podemos pensar em que funções podem ser aplicadas em uma lista que retorne apenas um único valor.

Um exemplo seria a soma de todos os valores de uma lista.

Podemos implementar a função reduzir da seguinte maneira:

```
reduzir :: (a -> a -> a) -> [a] -> a
reduzir f [x] = x
reduzir f (x:xs) = f x (reduzir f xs)
```

Reduzir

A função Reduzir definida anteriormente possui um erro se aplicada a uma lista vazia. Para contornar este erro podemos reescrever uma nova função reduzir da seguinte maneira:

```
reduzird :: (a -> b -> b) -> b -> [a] -> b
reduzird f aux [] = aux
reduzird f aux (a:b) = f a (reduzird f aux b)
```

Reduzir

Agora podemos utilizar funções anonimas aplicadas na função reduzir:

```
> reduzird ( \ x y -> y+1) 0 [5,12,2,3]
```

Reduzir

A função Reduzir definida anteriormente calculava o resultado com a assosciatividade da variável auxiliar à direita da função. Podemos reescrever uma nova função reduzir à esquerda da seguinte maneira:

```
reduzire :: (a -> b -> a) -> a -> [b] -> a
reduzire f aux [] = aux
reduzire f aux (a:b) = reduzire f (f aux a) b
```

Exercícios

1. Construa uma função que dada uma lista de strings, transforme todas as letras para maisuculas/minusculas. Usando a mesma função, use uma lista de inteiros e transforme os números de 0 até 9 para seu equivalente em string.

```
>transforma all2min "LpGUdeSc"
>"lpgudesc"
>transforma all2mai "LpGUdeSc"
>"LPGUDESC"
>transforma num2string [1,2,3]
>["um","dois","três"]
```

Exercícios

Utilizando a função aplicarsobre (mapear) e reduzir, para uma lista de inteiros, calcule inicialmente o quadrado da lista. Em seguida divida pelo valor de sua posição na lista. Finalmente some os valores de toda a lista.

$$\frac{x_1^2}{1} + \frac{x_2^2}{2} + \frac{x_3^2}{3} + \ldots + \frac{x_n^2}{n}$$

Exercícios

Utilizando as funções implementadas anteriormente calcule o valor:

$$e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

Referências


C. C. de Sá; M. F. da Silva
Haskell: Uma Abordagem Prática
(Novatec, 2006)