Composição e Point-Free Style

Programação Funcional

Prof. Maycon Amaro

Composição de Função

Na matemática, uma função composta é uma função que é aplicada ao resultado de outra função.

$$g(x) = x + 1$$
$$f(x) = 2 * x$$

$$h = f \circ g$$
$$h(x) = f(g(x)) = 2 * (x + 1)$$

$$j = g \circ f$$
$$j(x) = g(f(x)) = 2x + 1$$

Ponto de Função

No ramo da matemática conhecido como *topologia*, um **ponto** é um valor do espaço sobre o qual uma função está definida.

Ao escrever $f(x) = \ldots$, o x descreve o ponto da função, ou seja, um valor.

```
f :: Int -> Int
f x = x + 1
-- x é o ponto da função f
```

Ponto fixo de Função

Um ponto fixo é um valor para a função que resulta em si mesmo. 0 é um ponto fixo da função de dobro.

$$f(x) = 2x$$
$$f(0) = 0$$

Disso deriva o nome *operador de ponto fixo* no λ-cálculo, que é o termo que permite recursão.

Função "Livre de Ponto"

Quando usamos composição de função na matemática, não há porque escrever os parâmetros, i.e., os pontos.

$$h = f \circ g$$

Isso inspira uma forma de escrever funções na computação, conhecida como **point-free style**.

Point-free style

Sabemos que `even` está no prelúdio e verifica se um número é par. Podemos criar um *alias* para ela:

```
ehPar :: Int -> Bool
ehPar x = even x
```

Como são a mesma função, não há necessidade de fazer menção ao parâmetro:

```
ehPar :: Int -> Bool
ehPar = even
```

Aplicação Parcial

A função sucessor para números utiliza o operador de soma:

```
sucessor :: Int -> Int
sucessor x = x + 1
```

Colocando o operador em sua forma pré-fixada:

```
sucessor :: Int -> Int
sucessor x = (+) x 1
```

Aplicação Parcial

Como a soma é uma operação comutativa, podemos inverter a ordem dos parâmetros:

```
sucessor :: Int -> Int
sucessor x = (+) 1 x
```

A função de sucessor nada mais é que a aplicação parcial da soma com o número 1. Não há porque mencionar o parâmetro.

```
sucessor :: Int -> Int
sucessor = (+) 1
```

E se não for comutativo?

A função metade é uma aplicação parcial de `div`:

```
metade :: Int -> Int
metade x = div x 2
```

`div` não é comutativo, não podemos trocar a ordem.

Devemos abandonar o *point-free style* aqui?

A função `flip`

Uma função de ordem superior do prelúdio que inverte a ordem dos parâmetros, sem alterar o comportamento da função:

```
flip :: (a -> b -> c) -> b -> a -> c
flip f b a = f a b
```

Com uso dela, podemos fazer com que o denominador venha primeiro ao usar `div`:

```
metade :: Int -> Int
metade x = flip div 2 x
```

E com isso podemos usar *point-free style*:

```
metade :: Int -> Int
metade = flip div 2
```

Compondo Funções

O operador `.` em Haskell serve para compor funções. É apenas uma coincidência ele se chamar ponto. *Point-free style* não se refere à ele, mas aos **pontos de função**.

```haskell
f :: Int -> Int
f = (*) 2

g :: Int -> Int
g = (+) 1

h :: Int -> Int
h = f . g

j :: Int -> Int
j = g . f
```

Point-free style

Escrever funções em *point-free style* é considerado uma excelente prática.

```
somaElementos :: [Int] -> Int
somaElementos = foldr1 (+)
```

Essa versão é mais compacta, elegante e abstraída que a versão *point-wise* já vista anteriormente.

Além disso, oferece oportunidades de otimização de código aos compiladores. O compilador padrão de F# é um exemplo.

Exemplos

Um número que não é par, é ímpar:

```
odd :: Int -> Bool
odd = not . even
```

`odd` está disponível no prelúdio.

Exemplos

A primeira coluna de uma matriz é o primeiro elemento de cada linha.

```
fstColumn :: [[Int]] -> [Int]
fstColumn = map head
```

Exemplos

O último elemento de uma lista é o primeiro de trás pra frente.

```
last :: [Int] -> Int
last = head . reverse
```

`last` está disponível no prelúdio, e é genérica.

Em Outras Linguagens

JavaScript

```
const even = (x) => x % 2 == 0
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
console.log(nums.filter(even))
//[2, 4, 6, 8, 10]
```

Unix Scripting

Esse comando mostra o modelo da placa de vídeo em sistemas Unix. O operador | funciona como composição de programas, e é chamado de *pipeline*.

```
glxinfo | grep Device
```