

Polimorfismo: Permite que uma função trabalhe com diferentes tipos de dados. Ex.: Cálculo Lâmbda.

Cálculo Lâmbda

Sintaxe: o " λx " simboliza função

→ Lâmbda abstração:

$\lambda x. \text{expressão}$

o corpo da função vem após o ponto

→ <expressão> → x

| <expressão> <expressão>
| $\lambda x. <\text{expressão}>$

(variável)
(aplicação)
(abstracção)

Ex.: A função $x \rightarrow x+1$ representa uma função que recebe um argumento "x" e retorna "x+1". Escreve-se então:

$\lambda x. x+1$

→ Escopo:

→ Neste caso, o escopo da função é o próprio corpo da função

* Ver numerais de Church até currificação

Aula 19/10

* Variáveis livres e ligadas

$(\lambda z)(\lambda z.zz)(\lambda z.zy)$

variável ligada variável livre

* Currificação: Passar um argumento por vez p/ função (função currificada)

$\text{Soma } xy = x+y$

A ideia é ter $\rightarrow \text{Soma } 2y = 2+y$

$f_c = \lambda g. \lambda x. g x$

→ Como passar uma função p/ função currificada?

1) Defina a função normal

Ex.: $\text{SomaTupla}(x,y) = x+y$

2) Transformar p/ curtiada c/ a função 'curry'

$\text{SomaCurried} = \text{curry SomaTupla}$

* Divergência:

→ Funções que não chegam a forma normal (mais reduzida)

FUNÇÕES DE ORDEM SUPERIOR

funcão pode ser argumento p/
outra função

funções genéricas sobre listas:

→ mapeamento :

↳ transforma cada objeto de um conjunto em outros elementos de outro conjunto



→ filtragem: Condicão que é aplicada em uma lista de modo que filtre os elementos



→ Redução

Ex.: menor valor, soma de elementos



FUNÇÕES DE ORDEM SUPERIOR

→ mapear:

→ Uma função p multiplicar uma lista por 2, 3 e 4:

$$\text{mult2 } x = 2 * x$$

$$\text{mult3 } x = 3 * x \quad (\Rightarrow)$$

$$\text{mult4 } x = 4 * x$$

dobrar = mapear mult2 lista

triplo = mapear mult3 lista

quadruplo = mapear mult4 lista

definição da função mapear:

$$\text{mapear } [] = []$$

$$\text{mapear } f (x:xs) = f x : \text{mapear } f xs$$

} no terminal:

ghci> mapear

qualquer nº
(2 *) [1,2,3,4,5]
↓

[2,4,6,8,10]

Haskell tem uma função parenta: função map

→ filtrar:

Definição:

'p' é condição

$$\text{filtrar } [] = []$$

$$\text{filtrar } p (x:xs) | p x = x : \text{filtrar } p xs$$

otherwise = filtrar p xs

} no terminal:

ghci> filtrar (>1) [1,2,3,4,5,1,6,1]

↓

[2,3,4,5,6]

função pronta em Haskell: filter

Ex.: Filtrar os nº pares de uma lista:

$$\text{par } x = \text{mod } x \ 2 == 0$$

$$\text{filtrar_} [] = []$$

$$\text{filtrar } p(x:xs) \mid p\ x = x : \text{filtrar } p\ xs$$

| otherwise = filtrar p xs

no terminal:

filtrar par [1,2,1,3]

↓
[2]

ou

$$\text{filtrar } (\lambda x \rightarrow \text{mod } x \ 2 == 0) [1,2,1,3]$$

↓

[2]

→ Reduzir:

Definição:

$$\text{reduzir } f[x] = x$$

$$\text{reduzir } f(x:xs) = f\ x \ (\text{reduzir } f\ xs)$$

Ex.:

$$\text{reduzir soma } [1,2,3] = \text{soma } 1 \ (\text{reduzir soma } [2,3])$$

$$\text{soma } 2 \ (\text{reduzir soma } [3])$$

$$\downarrow$$

$$\text{soma } 2 \ 3 = 5$$

Outra forma sóp' conter no erro (lista vazia)

$$\text{reduzir } f \text{ aux } t] = \text{aux}$$

$$\text{reduzir } f \text{ aux } (a:b) = f\ a \ (\text{reduzir } a\ f \text{ aux } b)$$

$$\text{Ex.: reduzir soma } 0 [1,2,3] = \text{soma } 1 \ (\text{reduzir soma } 0 [2,3])$$

$$\text{reduzir soma } 2 \ (\text{reduzir soma } 0 [3]) \leftarrow (\text{soma } 1 \ 5) = 6$$

$$\hookrightarrow \text{soma } 3 \ (\text{reduzir } 0 \ [\]) \underbrace{\qquad\qquad\qquad}_{(\text{soma } 3 \ 0) = 3}$$

0

Revisão: Redução beta

α -equivalentes!

$$(\lambda z.z)(\lambda z.zz)(\lambda z.zy)$$

iig iig iig livre

$$(\lambda z.z)(\lambda y.yz)(\lambda x.xa)$$

$$(\lambda x.xa)(\lambda x.xa)$$

$$(\lambda x.xa)a$$

aa

$$(\lambda z.zz)(\lambda z.zy)$$

$$(\lambda z.zy)(\lambda z.zy)$$

$$(\lambda z.zy)y$$

$$yy$$

$(\lambda x \cdot \lambda y \cdot xy) (\lambda a \cdot a) b$ livre
 * variável livre: b
 * PI beta redução:
 → começa substituindo $(\lambda a \cdot a)$ em λx
 (é mais externo)

$(\lambda y \cdot (\lambda a \cdot a) yy) b$
 $(\lambda a \cdot a) bb$
 bb

$(\lambda x \cdot (\lambda y \cdot (\lambda y \cdot y) y) z)$ livre
livre } redução α

$\cong (\lambda x \cdot (\lambda a \cdot (\lambda a) a) y) z$

$(\lambda a \cdot (\lambda a) a)$
 zy

redução β

$((\lambda x \cdot xx) (\lambda y \cdot y)) (\lambda y \cdot y)$
 $(\lambda y \cdot y) (\lambda y \cdot y) (\lambda y \cdot y)$
 $(\lambda y \cdot y) (\lambda y \cdot y)$
 $(\lambda y \cdot y)$

$((\lambda x \cdot \lambda y \cdot (xy)) (\lambda y \cdot y)) w$
 $\cong ((\lambda x \cdot \lambda a \cdot (xa)) (\lambda y \cdot y)) w$
 $(\lambda a \cdot (\lambda y \cdot y) a) w$

$(\lambda y \cdot y) w$

$(\lambda x \cdot x x) (\lambda y \cdot y x) z$

$(\lambda y \cdot y x) (\lambda y \cdot y x) z$

$(\lambda y \cdot y x) x z$
 $x x z$

Aula 07/11 fúnçao composta em Haskell: $f \circ g = f(g(x))$

Exercício 2) slide:

→ neste caso também é possível utilizar "\$"

Separar p xs = (filter p xs, filter (not p) xs)

associatividade à direita

$((f \circ g) h) j \Rightarrow (f(g(hj)))$

módulos

Exemplo de módulos que já estavam sendo usados

$\rightarrow \text{import Data.Char}$
 $\rightarrow \text{import Data.List}$

ok Parta no moodle

→ Ex.: Cube.area

✓ ↗ função que
Nome de eu quero
módulo