# PROGRAMAÇÃO FUNCIONAL: Haskell
# Cálculo Lambda

Professor Rafael Kingeski

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

# Índice

# Índice

# História do Cálculo Lambda



- Criada por Alonzo Church em 1936;
- Visava dar uma resposta ao Entscheidungsproblem levantado por David Hilbert em 1900;
- O problema de decisão proposto por Hilbert é conhecido como um marco para a história inicial da computação;
- $\lambda$-cálculo foi de extrema importância para a matemática e serviu como base para a linguagem LISP, primeira linguagem funcional;

# Índice

# Cálculo Lambda



Cálculo lambda ($\lambda$-calculo) é um modelo matemático capaz de ilustrar de forma simples alguns importantes conceitos presentes em linguagens de programação, como por exemplo ligação, escopo, ordem de avaliação, computabilidade, sistemas de tipos, etc.

Dada uma função matemática para calcular o cubo de um número:

$$f(x) = x^3 \tag{1}$$

Podemos ainda chamar esta função de cubo(x):

$$cubo(x) = x^3 \tag{2}$$

Utilizando a notação proposta por Church, chama de $\lambda$-expressão ou expressão lambda.
O processo de reescrever uma função abstraindo o seu nome é chamado de abstração, representa um nome indefinido genérico.
Utilizando o simbolo auxiliar $\lambda$ reescrevendo a função cubo como uma $\lambda$-abstração:

$$\lambda x.x^3 \tag{3}$$

Uma expressão lambda (em sua forma pura) é definida pela seguinte sintaxe:

`<expressão>` $\rightarrow$ `x` (variável)
| `<expressão> <expressão>` (aplicação)
| $\lambda x$.`<expressão>` (abstração-$\lambda$ ou função)

# Escopo



O escopo de uma variável é o contexto no qual a variável pode ser usada. Em uma expressão $\lambda x.M$ a expressão M é o escopo da variável x.

$$(\lambda a.\lambda b.\lambda c.a(abc))(\lambda a.\lambda b.ab)$$

Para aplicar valores em uma abstração lambda devemos seguir a ordem de precedência:

$$\lambda x.(\lambda y.(x + y^2))\ 2\ 3 \tag{4}$$

$$(\lambda y.(2 + y^2))\ 3 \tag{5}$$

$$2 + 3^2 \tag{6}$$

$$11 \tag{7}$$

$$\lambda x.\lambda y.\lambda z.\lambda w.(((x+y)+z)+w))\ 3\ 4\ 5\ 6 \tag{8}$$

$$\lambda y.\lambda z.\lambda w.(((3+y)+z)+w))\ 4\ 5\ 6 \tag{9}$$

$$\lambda z.\lambda w.(((3+4)+z)+w))\ 5\ 6 \tag{10}$$

$$\lambda w.(((3+4)+5)+w))\ 6 \tag{11}$$

$$(((3+4)+5)+6) \tag{12}$$

$$((7+5)+6) \tag{13}$$

$$(12+6) \tag{14}$$

$$(18) \tag{15}$$

**Equivalência $\alpha$**

Duas expressões lambda são $\alpha$-equivalentes caso difiram apenas nos nomes das variáveis que ocorrem ligadas a abstrações $\lambda$, funções $\alpha$-equivalentes representam a mesma computação.
Exemplo:

$$\lambda x \lambda y.x \equiv_\alpha \lambda a \lambda b.a \tag{16}$$

$$\lambda f \lambda x.fx \equiv_\alpha \lambda a \lambda b.ab \tag{17}$$

Uma variável livre é uma variável que não está ligada a uma abstração $\lambda$. O conjunto de variáveis livres de uma expressão M é representado por FV(M), que é definido como:

$$FV(x) = \{x\}$$
$$FV(MN) = FV(M) \cup FV(N)$$
$$FV(\lambda x.M) = FV(M) - \{x\}$$

# Redução $\beta$



A substituição de todas as ocorrências livres de uma variável x por uma expressão M é representada por $[M/x]$. O axioma central do cálculo-$\lambda$ envolve a substituição e é chamado de redução $\beta$. A redução $\beta$ representa a aplicação de uma abstração $\lambda$ em um argumento:

$$(\lambda x.N)M =_{\beta} [M/x]N \tag{18}$$

Exemplo:

$$(\lambda f.fx)(\lambda y.y) \equiv (\lambda y.y)x \equiv x \tag{19}$$

estas expressões são $\beta$-equivalentes.

Exemplo:

$$(\lambda x.xx)(\lambda a.a)(\lambda b.\lambda c.c) \tag{20}$$

$$(\lambda a.a)(\lambda a.a)(\lambda b.\lambda c.c) \tag{21}$$

$$(\lambda a.a)(\lambda b.\lambda c.c) \tag{22}$$

$$(\lambda b.\lambda c.c) \tag{23}$$

# Redução $\beta$



Exemplo:

$$(\lambda x.x)(\lambda a.ab)(\lambda c.\lambda d.c) \tag{24}$$

$$(\lambda a.ab)(\lambda c.\lambda d.c) \tag{25}$$

$$(\lambda c.\lambda d.c)b \tag{26}$$

$$\lambda d.b \tag{27}$$

# Ordem de Avaliação



As regras de avaliação não especificam a ordem exata que uma expressão deve ser reduzida, uma possível ordem de avaliação é reduzir completamente o argumento antes de substituí-lo no corpo da função, essa avaliação e chamada avaliação por valor (call-by-value ou eager evaluation). Uma outra alternativa é avaliar o argumento apenas se necessário, essa ordem de avaliação e chamada de avaliação preguiçosa (lazy evaluation, call-by-need ou normal order).

Avaliação por Valor:

$$(\lambda w \lambda y \lambda x.y(wyx))((\lambda a \lambda b \lambda c.b(abc))(\lambda s \lambda z.z)$$

$$(\lambda w \lambda y \lambda x.y(wyx))((\lambda b \lambda c.b((\lambda s \lambda z.z)bc))$$

$$(\lambda w \lambda y \lambda x.y(wyx))((\lambda b \lambda c.b((\lambda z.z)c))$$

$$(\lambda w \lambda y \lambda x.y(wyx))(\lambda b \lambda c.b(c))$$

$$\lambda y \lambda x.y((\lambda b \lambda c.b(c))yx)$$

$$\lambda y \lambda x.y((\lambda c.y(c))x)$$

$$\lambda y \lambda x.y(y(x))$$

Avaliação Preguiçosa:

$$(\lambda w \lambda y \lambda x.y(wyx))((\lambda a \lambda b \lambda c.b(abc))(\lambda s \lambda z.z)$$

$$\lambda y \lambda x.y(((\lambda a \lambda b \lambda c.b(abc))(\lambda s \lambda z.z))yx)$$

$$\lambda y \lambda x.y((\lambda b \lambda c.b((\lambda s \lambda z.z)bc))yx)$$

$$\lambda y \lambda x.y((\lambda b \lambda c.b((\lambda z.z)c))yx)$$

$$\lambda y \lambda x.y((\lambda b \lambda c.b(c))yx)$$

$$\lambda y \lambda x.y((\lambda c.y(c))x)$$

$$\lambda y \lambda x.y(y(x))$$

# Teorema de Church-Rosser



Se $P \rightarrow_\beta R$ e $P \rightarrow_\beta Q$ então existe um S tal que: $R \rightarrow_\beta S$ e $Q \rightarrow_\beta S$

Corolário: Nenhuma expressão pode ser convertida em duas formas normais distintas. Isto significa que não existem duas formas normais para uma expressão que não sejam $\alpha$-convertíveis entre si.

Nenhuma sequencia de redução poderá levar a um resultado incorreto, o máximo que poderá acontecer é a não-terminação.

A **Ordem Normal de Redução** especifica que o redex mais à esquerda mais externo deverá ser reduzido primeiro.

# Numerais de Church



Pode-se expressar os números naturais por uma função sucessora do valor 0. Desta forma o número 1 pode ser expresso como sucessor de zero, o número 2 como sucessor do número 1 e assim sucessivamente.

$$0 \equiv \lambda s \lambda z.z \tag{28}$$

$$1 \equiv \lambda s \lambda z.s(z) \tag{29}$$

$$2 \equiv \lambda s \lambda z.s(s(z)) \tag{30}$$

$$3 \equiv \lambda s \lambda z.s(s(s(z))) \tag{31}$$

$$4 \equiv \lambda s \lambda z.s(s(s(s(z)))) \tag{32}$$

$$SUC = (\lambda w \lambda y \lambda x.y(wyx)) \tag{33}$$

$$ADD = \lambda x \lambda y \lambda w \lambda u.xw(ywu) \tag{34}$$

Exemplo:

$$(\lambda w \lambda y \lambda x.y(wyx))(\lambda s \lambda z.s(s(z))) \tag{35}$$

$$(\lambda y \lambda x.y((\lambda s \lambda z.s(s(z)))yx) \tag{36}$$

$$(\lambda y \lambda x.y(\lambda z.y(y(z))x)) \tag{37}$$

$$(\lambda y \lambda x.y(y(y(x)) \equiv_\alpha \lambda s \lambda z.s(s(s(z))) \tag{38}$$

Representa-se os valores booleanos, verdade e falsidade por:

$$V = \lambda a \lambda b.a \tag{39}$$

$$F = \lambda a \lambda b.b \tag{40}$$

E os operadores lógicos:

$$E = \lambda x \lambda y.xy(\lambda u \lambda v.v) \tag{41}$$

$$OU = \lambda x \lambda y.x(\lambda u \lambda v.u)y \tag{42}$$

$$NAO = \lambda x.x(\lambda u \lambda v.v)(\lambda w \lambda z.w) \tag{43}$$

# Booleanos



Função IF:

$$\lambda c \lambda x \lambda y.cxy \tag{44}$$

Exemplo:
If verdadeiro 1 2

$$(\lambda c \lambda x \lambda y.cxy)(\lambda a \lambda b.a)\ 1\ 2 \tag{45}$$

$$(\lambda x \lambda y.(\lambda a \lambda b.a)xy)\ 1\ 2 \tag{46}$$

$$(\lambda y.(\lambda a \lambda b.a)1y)\ 2 \tag{47}$$

$$(\lambda a \lambda b.a)\ 1\ 2 \tag{48}$$

$$1$$

UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

# Currificação



Podemos representar uma função que recebe dois argumentos como $\lambda x.(\lambda y.M)$, sendo M uma expressão lambda, possivelmente envolvendo x e y. Aplicando um único argumento a essa função temos como retorno uma função que aceita o segundo argumento y. Por exemplo, a função matemática $f(g,x)$ tem dois argumentos, mas poderia ser representada em $\lambda$-cálculo como:

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

A diferença entre $f$ e $f_c$ é que a função $f$ deve receber um par de argumentos (g, x) enquanto $f_c$ pode receber um único argumento $g$, retornando nesse caso $\lambda x.gx$. Depois de passados todos os argumentos para a funçao $f_c$ o resultado é exatamente o mesmo da função $f$. Funções como $f_c$ passaram a ser conhecidas como funções Currifcadas depois que o matemático Haskell Curry estudou suas propriedades.

Nem todas as Expressões-$\lambda$ vão reduzir a uma expressão na forma normal. Não porque que elas já estejam em sua forma normal, mas por que elas divergem.
Exemplo:

$$(\lambda x.xx)(\lambda x.xx)$$

$$[(\lambda x.xx)/x]\,.xx$$

$$(\lambda x.xx)(\lambda x.xx)$$

# Exercícios

$$(\lambda abc.cba)zz(\lambda wv.w)$$

# Exercícios



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

# Exercícios



$$(\lambda y.y)(\lambda x.xx)(\lambda z.zq)$$

# Cidadão de Primeira Classe



No contexto de linguagens de programação cidadãos ou valores de primeira classe são valores que podem ser passados como argumentos e retornados por funções. Em Haskell, funções são valores de primeira classe.

Função que aplica duas vezes outra função:

aplica2 f x = f (f x)

Função incremento:

inc x = x + 1

Se podemos aplicar uma função a outra então podemos fazer:

aplica2 inc 10

E o resultado será 12.

# Notação Lambda



```
aplica2 f x = f (f x)
aplica2 (\ x -> x + 1) 10
(\ x -> x + 1) ((\ x -> x + 1) 10)
(\ x -> x + 1 ) 10 + 1
10 + 1 + 1
12
```

# Currificação



```
maiorc x y = if x>y then x else y
maiorn (x,y) = if x>y then x else y
curry' = \ f x y -> f (x,y)
```

UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

# Referências


- C. C. de Sá; M. F. da Silva
  *Haskell: Uma Abordagem Prática*
  (Novatec, 2006)
- Jorge Muniz Barreto
  Material de Disciplina - Programação Funcional
  http://www.inf.ufsc.br/ j.barreto/PF/material.htm
- Cristiano Damiani Vasconcellos
  Material de Disciplina - Programação Funcional


UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA