# Cálculo Lambda

## Linguagens de Programação
## 2018.2019

*Teresa Gonçalves*
*tcg@uevora.pt*

Departamento de Informática, ECT-UÉ

# Sumário

**Funções e expressões funcionais**

**Expressões lambda**

**Redução, confluência e formas normais**

**Programação em cálculo-$\lambda$**

# Cálculo Lambda

## História

Alonzo Church, 1936

## O que é?

Sistema matemático

Permite ilustrar conceitos importantes de LP de forma simples e pura

## Tradicionalmente

Notação para definição de funções

Conjunto de regras de cálculo (**redução**)

Sistema de prova para equações entre expressões

# Cálculo Lambda

## Significado

### Lambda

Letra Grega λ

### Cálculo

Redução utilizada para **calcular** o resultado da aplicação duma função a um ou mais argumentos

Forma de avaliação simbólica de expressões

## Conceitos de LP

Parametrização

Através de expressões funcionais

Ligação (*binding*)

Através de declarações

Estratégias de avaliação

# Funções e expressões funcionais

# Cálculo-λ puro

## O que é uma função?

Uma regra para determinar um valor a partir de um ou mais argumentos

Exemplos

$$f(x) = x^2 + 3$$
$$g(x, y) = \sqrt{x^2 + y^2}$$

## 2 operadores

Abstração-λ (definição da função)

Aplicação (cálculo da função)

# Abstração Lambda

## O que é?

Operador que define uma função

**λx.M**

Se **M** for uma expressão, é a função obtida ao tratar **M** como função da variável **x**

## Exemplo

Função identidade

**λ**x.x

Equivalente a “I(x)=x”

obriga dar um nome à função

# Aplicação

## O que é?

Operador que aplica a função a um argumento

**$(\lambda x.M_1)\ M_2$**

Se $M_2$ for uma expressão, é aplicação da função ($\lambda x.M_1$) a $M_2$

## Exemplos

**$(\lambda x.x)\ M$**

Aplicação da função identidade à expressão M

O resultado desta expressão é M

**$\lambda f.\lambda g.\lambda x.f(g\ x)$**

Dadas duas funções **f** e **g**, esta função corresponde à composição de **f** e **g**
**$\lambda x.f(g\ x)$**

# Expressões lambda

# Sintaxe duma expressão lambda

## Assume-se

um conj. infinito *V* de **variáveis**, sendo *x*, *y*, *z*, ... variáveis arbitrárias

## Gramática BNF

<M> → x | <M> <M> | λx.<M>

*x* pode ser qualquer variável (elemento de V)

## Significado

x refere uma variável (determinada pelo contexto)

$M_1$ $M_2$ é a aplicação de $M_1$ ao argumento $M_2$

λx.M é a definição da função M de x

# Termo lambda

## É uma expressão de cálculo-λ

## Exemplos

| | |
|---|---|
| **λx.x** | Uma abstração-λ designada função identidade |
| **λx.f(g x)** | Outra abstração-λ |
| **(λx.x)** 5 | Uma aplicação |

# Convenções sintáticas

O âmbito de **λ,** numa abstração-λ, estende-se o mais para a direita possível

Exemplo

λx.x y = λx.(x y)

λx.x y ≠ (λx.x) y

A aplicação associa à esquerda

Exemplo

x y z = (x y) z

x y z ≠ x (y z)

# Ocorrências de variáveis

## Livre

Ocorrência fora do âmbito do operador de ligação

## Ligadora

Ocorrência onde a variável se torna ligada

## Ligada

Outras ocorrências não livres

## Exemplo

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

x: 1ª ocorrência → ocorrência ligadora
2ª ocorrência → ocorrência ligada

y: 1ª ocorrência → ocorrência ligadora
2ª ocorrência → ocorrência ligada
3ª ocorrência → ocorrência livre

# Ligação de uma variável

## Ocorrência livre

A variável não foi declarada na expressão

Não é possível avaliar a expressão sem colocá-la noutra (maior) que associe um valor à variável

Exemplo

Na expressão $x+3$, $x$ é livre

## Ocorrência ligada

É uma ocorrência não livre

**λ** é o operador de ligação

Porque liga uma variável num âmbito específico

Exemplo

Na expressão $\lambda x.M$, $x$ é ligada

$M$ é o âmbito da ligação $\lambda x$

# Variável ligada-λ

## Propriedade

É possível renomear uma variável ligada-λ *x* para *y* sem alterar o significado da expressão

Exemplo

λx.x define a mesma função que λy.y

## Expressões α-equivalentes

São expressões que diferem apenas nos nomes das variáveis ligadas

Escreve-se

$\lambda x.x =_{\alpha} \lambda y.y$

# Conj. variáveis livres

## VL(M)

conj. de variáveis livres na expressão M

definido por indução na estrutura das expressões

VL($x$) = {$x$}

VL($M\ N$) = VL($M$) ∪ VL($N$)

VL($\lambda x.M$) = VL($M$) – {$x$}, onde – significa diferença conj.

## Exemplos

VL($\lambda x.x$) = ∅

VL($\lambda f.\lambda x.$f(g x)) = {g}

# Sistema de prova do cálculo-λ

## Axiomas básicos

### equivalência-α

Equivalência de expressões com variáveis ligadas

### equivalência-β

Equivalência de expressões através do cálculo de uma aplicação por substituição

# Equivalência-α

$$\lambda x.M = \lambda y.[y/x]M$$

*[y/x]M* corresponde à substituição das ocorrências livres de *x* em *M* por *y*

*y* não pode aparecer em *M*

## Exemplo

$(\lambda x.(\lambda x.x)\ x)\ x =_{\alpha} (\lambda y.(\lambda z.z)\ y)\ x$

# Equivalência-β

$$(\lambda x.M)N = [N/x]M$$

$\lambda x.M$ é a função obtida ao tratar M como função de $x$

*[N/x] M :* o valor da aplicação em $N$ obtém-se, substituindo as ocorrências livres de $x$ em $M$ por $N$

## Exemplo

$(\lambda f.f\ x)\ (\lambda y.y) =_\beta (\lambda y.y)\ x =_\beta x$

# Substituição

## [N/x] M

Resultado da substituição de x por N em M (N substitui x em M)

## Regras

Definidas por indução na estrutura de M

$[N/x]\ x = N$

$[N/x]\ y = y \qquad\qquad y \neq x$

$[N/x]\ (M_1\ M_2) = [N/x]\ M_1\ \ [N/x]\ M_2$

$[N/x]\ (\lambda x.M) = \lambda x.M$

$[N/x]\ (\lambda y.M) = \lambda y.\ ([N/x]\ M) \qquad y \neq x,\ y \notin VL(N)$

# Colisão de variáveis

## Quando acontece?

Sempre que na substituição *[N/x] M*, as ligações-λ em M têm o mesmo nome que variáveis livres de N

## Como evitar?

Renomear as variáveis ligadas em (λ*x*.*M*) *N*

Todas as variáveis ligadas devem ser diferentes umas das outras e das variáveis livres

## Exemplo

$$(\lambda f.\lambda x.f(fx))(\lambda y.y + x)$$

# Redução, forma normal e confluência

# Redução-β

## O que é?

equivalência-β que permite avaliar uma função

Para indicar a direção, utiliza → em vez de =

$(\lambda x.M)\ N \rightarrow [N/x]\ M$

## *M → N*

*M* reduz-β para *N*

Quando um subtermo de M reduz-β para criar *N*

## Redex (reduction expression)

É um termo do tipo $(\lambda x.M)\ N$

# Forma normal

## O que é?

Uma expressão que não é possível reduzir mais

## Exemplos

$\lambda z.zz$

$$
\begin{aligned}
& (\lambda f.\lambda x.f(fx))(\lambda y.y+1)2 \\
\rightarrow\ & \lambda x.((\lambda y.y+1)((\lambda y.y+1)x))2 \\
\rightarrow\ & (\lambda x.(\lambda y.y+1)(x+1))2 \\
\rightarrow\ & (\lambda x.x+1+1)2 \\
\rightarrow\ & (2+1+1) \qquad \longleftarrow \text{ Forma normal}
\end{aligned}
$$

Se adicionarmos regra cálculo da adição

$$
\begin{aligned}
& 2+1+1 \\
\rightarrow\ & 3+1 \\
\rightarrow\ & 4
\end{aligned}
$$

# Reduções alternativas

**Podem existir várias reduções alternativas**

**Exemplo**

$$
\begin{aligned}
& \underline{(\lambda f.\lambda z.f(fz))(\lambda y.y+x)} \\
\rightarrow\ & \lambda z.\underline{(\lambda y.y+x)((\lambda y.y+x)z)} \\
\rightarrow\ & \lambda z.\underline{(\lambda y.y+x)(z+x)} \\
\rightarrow\ & \lambda z.\underline{(z+x)+x}
\end{aligned}
$$

$$
\begin{aligned}
& \underline{(\lambda f.\lambda z.f(fz))(\lambda y.y+x)} \\
\rightarrow\ & \lambda z.\underline{(\lambda y.y+x)((\lambda y.y+x)z)} \\
\rightarrow\ & \lambda z.\underline{((\lambda y.y+x)z)+x} \\
\rightarrow\ & \lambda z.\underline{(z+x)+x}
\end{aligned}
$$

# Reduções intermináveis

**Existem termos onde é possível reduzir sempre (sem nunca chegar à forma normal)**

**Exemplo**

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

# Confluência

## O que é?

Se existir, a forma normal é única

## O que significa?

Se

M puder ser reduzida para uma forma normal

Então

Existe uma única forma normal de M, independente/ da ordem de avaliação das sub-expressões

## A ordem de avaliação não afecta o resultado final da expressão!

# Estratégia de redução

## É uma regra para escolher redexes

## Estratégias

- Call-by-name (ou ordem normal de redução)
  - Escolhe o redex mais exterior, mais à esquerda
- Call-by-value
  - Escolhe o redex mais interior, mais à esquerda

## Exemplo

- Call-by-name

  $\underline{(\lambda x \lambda z.(\lambda x.x)\ z\ (yz))\ (\lambda x.x)}$

- Call-by-value

  $(\lambda x \lambda z.\underline{(\lambda x.x)\ z}\ (yz))\ (\lambda x.x)$

# Garantia de chegada à forma normal

## Call-by-name: sim

$$(\lambda x.\lambda y.x)\ z\ N \rightarrow (\lambda y.z)\ N \rightarrow z$$

## Call-by-value: não

$$\begin{aligned}
&(\lambda y.z)\ ((\lambda x.xx)(\lambda x.xx)) \\
\rightarrow\ &(\lambda y.z)\ ((\lambda x.xx)(\lambda x.xx)) \\
\rightarrow\ &(\lambda y.z)\ ((\lambda x.xx)(\lambda x.xx)) \\
\rightarrow\ &\ldots
\end{aligned}$$

# Rapidez de chegada à forma normal

## Call-by-value: mais rápido

$$
\begin{aligned}
& (\lambda x.xx)((\lambda y.y)(\lambda z.z)) \\
\rightarrow\ & (\lambda x.xx)(\lambda z.z) \\
\rightarrow\ & (\lambda z.z)(\lambda z.z) \\
\rightarrow\ & (\lambda z.z)
\end{aligned}
$$

## Call-by-name: mais lento

$$
\begin{aligned}
& (\lambda x.xx)((\lambda y.y)(\lambda z.z)) \\
\rightarrow\ & ((\lambda y.y)(\lambda z.z))\ ((\lambda y.y)(\lambda z.z)) \\
\rightarrow\ & (\lambda z.z)\ ((\lambda y.y)(\lambda z.z)) \\
\rightarrow\ & (\lambda y.y)(\lambda z.z) \\
\rightarrow\ & (\lambda z.z)
\end{aligned}
$$

# Computação em cálculo lambda

# Declaração let

## Construção que permite fazer declarações locais

*let x = M in N*

## *let* é "açucar sintático"

*let x = M in N* ⇔ (λ*x*.*N*) *M*

Não adiciona poder ao cálculo, mas torna-o mais "amigável"

## Exemplo

let *compose* = λ*f*.λ*g*.λ*x*.*f (g x) in compose h h*

$=_{\beta}$

λ*x*.*h (h x)* ← composição de h com h

# Múltiplas variáveis

## cálculo-λ

permite tratar uma expressão *M* como função de uma variável *x*

## Tratar *M* como função de duas variáveis *x* e *y*

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

Função de um único argumento que, quando aplicada, devolve uma segunda função que aceita um 2º argumento

## Exemplo

f(g,x)=g(x) argumento: par *(g,x)*

$f_{curry}=\lambda g.\lambda x.g\ x$ argumento: *g*

# *Currying*

## Técnica que simula funções de várias variáveis

*g* é uma versão *curried* de *f* se

$f(x_1,x_2,\ldots,x_k) = g\ x_1\ x_2 \ldots x_k$

A função *f* tem *k* argumentos que *g* aceita à vez

## Útil para avaliação parcial

quando um dos argumentos não está disponível

$(\lambda x.\lambda y.x*y)\ 2 \rightarrow \lambda y.2*y$

Uma função que multiplica o seu argumento *y* por 2

## Convenção

$\lambda x.\lambda y.M = \lambda xy.M$

# Exemplos

## Operador *

$\lambda x.\lambda y.x*y$

$(\lambda x.\lambda y.x*y)\ 2\ 3 \rightarrow (\lambda y.2*y)\ 3 \rightarrow\ 2*3$

## f(x,y)=x-y

$\lambda x.\lambda y.x-y$

$f(4, 3)$

$(\lambda x.\lambda y.x-y)\ 4\ 3$

# Abstração-λ em ML e C

## ML

fn *x* => *corpo_função;*

λ*x*.*corpo_função*

## C

int *f* (int *x*) {return *x*};

*chama_f*;

let *f* = (λ*x*.*x*) in *chama_f*

# Exemplo

```
int function f(x,y){
   return 2*x+y;
}
int main(){
   f(3,4);
}
```

**let f =** λxy.2*x+y in
  f 3 4

(λf. f 3 4) (λxy.2*x+y) →
(λxy.2*x+y) 3 4 →
(λy.2*3+y) 4 →
2*3+4

# Cálculo-λ aplicado

## O que é?

Uma extensão do cálculo-λ puro, que adiciona um conjunto de outras operações e tipos

## Exemplo

Adição dos operadores básicos de cálculo: +, -, *, /...

**LP = cálculo-λ aplicado**
**= cálculo-λ puro + tipos de dados adicionais**