Minicurso 1

Uma introdução ao cálculo lambda e a linguagens de programação funcionais

Rodrigo Machado
rma@inf.ufrgs.br

UFRGS UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

.inf INSTITUTO DE INFORMÁTICA UFRGS

Conteúdo

Ideias iniciais

Cálculo lambda
- Sintaxe
- Operação de substituição
- Equivalência alfa
- Redução beta
- Propriedades da redução beta

Programação em cálculo lambda
- Booleanos
- Números naturais
- Pares ordenados
- Listas
- Definições locais
- Funções recursivas
- Funcões de alta ordem

Revisão

Referências

Conteúdo

Ideias iniciais

Cálculo lambda
- Sintaxe
- Operação de substituição
- Equivalência alfa
- Redução beta
- Propriedades da redução beta

Programação em cálculo lambda
- Booleanos
- Números naturais
- Pares ordenados
- Listas
- Definições locais
- Funções recursivas
- Funcões de alta ordem

Revisão

Referências

Antes de iniciarmos ...

Considere os seguintes grupos de linguagens de programação:

Grupo 1 : OCAML, F#, LISP, Scheme/Racket, Haskell
Grupo 2 : Javascript, Python, Ruby, C#
Grupo 3 : C, C++, Java, Pascal/Delphi

Quais grupos contém linguagens de programação nas quais vocês já programaram?

Você já ouviu falar antes de Máquinas de Turing, Funções Recursivas Parciais ou Cálculo Lambda?

Antes de iniciarmos (2) ...

Suponha uma linguagem de programação com números e valores booleanos. Considere as seguintes construções:

1. execução condicional (`if-then` e `if-then-else`)
2. laços de repetição (`while` e `for`)
3. definição de variáveis e operador de atribuição
4. definição e aplicação de funções

Se fosse lhe pedido para você escolher três itens acima e jogar fora o item restante, quais itens seriam escolhidos?

Seria possível escrever todos os programas desejados somente com os itens escolhidos?

E se fosse possível escolher *somente um* dos itens acima?

o que é uma linguagem Turing-completa?

Origem: notação lambda

Considere a seguinte definição matemática:

$$f(x) = x^2 + 7$$

A igualdade acima está *definindo uma função* f, que consome um número e devolve um número.

$$f(3) = 3^2 + 7 = 16$$

A igualdade acima está *aplicando* a definição de f a um valor específico.

O propósito original da **notação lambda** foi resolver a seguinte ambiguidade:

$$f(e) = e^2 + 7$$

Afinal, se trata da aplicação de f sobre a constante $e = 2.716\ldots$ ou da definição de f?

Origem: notação lambda (2)

A **notação lambda** (Church, 1932) permite diferenciar claramente a *definição de uma função* de sua respectiva *aplicação*.

$$f = \lambda x.x^2 + 7$$

Acima temos a definição da função. O λx indica que x deve ser interpretado como um *parâmetro formal*, isto é, um nome temporário para o valor a ser recebido pela função.

$$f(3) = (\lambda x.x^2 + 7)(3) = 3^2 + 7$$

Acima temos a aplicação da função f, definida anteriormente, ao valor 3.

Note que o **significado** da aplicação é a **substituição** do *parâmetro formal* x pelo *valor concreto* 3.

Origem: cálculo lambda

Ao formalizar as ideias fundamentais de *definição* e *aplicação* de funções, se chegou ao **cálculo lambda**. A versão *pura* do cálculo não continha nada além de funções (sem números, sem operações).

Ainda na década de 1930, Church e seus alunos (Kleene, Rosser) mostraram que a versão pura do cálculo era tão expressiva quanto outros modelos de computação propostos (em particular, Máquinas de Turing e Funções Recursivas Parciais).

Atualmente, *Cálculo Lambda* diz respeito a uma grande família de formalismos construídos sobre os mesmos conceitos fundamentais.

Motivação: por que cálculo lambda?

O cálculo lambda original tem aproximadamente 80 anos! Não existiam nem computadores naquela época! *Por que estudar algo tão antigo em 2013?*

Algumas razões:

- cálculo lambda foi e continua sendo *influente*: ele serviu de inspiração para *linguagens de programação* como (Lisp e Haskell) e para o *estilo de programação funcional* nas demais linguagens.
- cálculo lambda (com tipos) é a *base* para o estudo formal de linguagens de programação e sistemas de tipos (mesmo as orientadas a objetos e imperativas).
- cálculo lambda (com tipos) *possui uma conexão* importante com Lógica Matemática, sendo a base de assistentes de prova como Coq e Agda.
- cálculo lambda é um cálculo *minimalista* e *elegante*.
- para saber utilizar de forma eficaz as **novas linguagens de programação**...

Motivação: funções anônimas hoje em dia

Linguagens de programação e suporte a funções anônimas:

```
C          : não
Pascal     : não
C++        : sim (a partir da versão C++11)
Javascript : sim
Python     : sim
Ruby       : sim
C#         : sim (a partir de versão 3.0)
Java       : sim (a partir de versão 8)
```

E é claro, todas as linguagens de programação funcionais/mistas:

`LISP, Scheme, Clojure, Scala, Ocaml, F#, Haskell, ...`

Este minicurso

Objetivos

1. Apresentar as ideias fundamentais de cálculo lambda: termos-lambda, substituição, alfa-equivalência, redução beta
2. Argumentar sobre propriedades do cálculo: confluência de redução beta, universalidade
3. Mostrar como codificar diversos tipos de dados e estruturas de controle como termos-lambda

Abordagem

- conceitual (alguns detalhes formais serão omitidos)
- foco em programação: vamos definir a função *fatorial* em cálculo lambda puro.

Conteúdo

Ideias iniciais

Cálculo lambda
- Sintaxe
- Operação de substituição
- Equivalência alfa
- Redução beta
- Propriedades da redução beta

Programação em cálculo lambda
- Booleanos
- Números naturais
- Pares ordenados
- Listas
- Definições locais
- Funções recursivas
- Funcões de alta ordem

Revisão

Referências

Sintaxe: pré-termos

Começaremos definindo um conjunto Λ^- de **pré-termos**.

Considere um conjunto infinito (mas contável) de *nomes* (a.k.a parâmetros, identificadores, variáveis) o qual chamaremos Var.

Vamos denotar os elementos de Var por $x, y, z, \ldots$

Definição: o conjunto Λ^- é o **menor** conjunto tal que:

1. se $x \in \mathrm{Var}$ então $x \in \Lambda^-$ (variáveis)
2. se $M \in \Lambda^-$ e $N \in \Lambda^-$ então $@(M, N) \in \Lambda^-$ (aplicação)
3. se $x \in \mathrm{Var}$ e $M \in \Lambda^-$ então $\lambda x.M \in \Lambda^-$ (abstração lambda)

Notação: vamos utilizar simplesmente $M\ N$ para representar $@(M, N)$.

Sintaxe: exemplos

Exemplos de elementos de Λ^-:

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

Nota: precisamos de regras claras para evitar ambiguidade na escrita de um pré-termo.

Exemplo: $\lambda x.x\ y$ representa $\lambda x.(x\ y)$ ou $(\lambda x.x)\ y$?

Sintaxe: regras de notação

1. O λ engloba tudo à direita (o máximo possível).

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

2. O operador @ é associativo à esquerda.

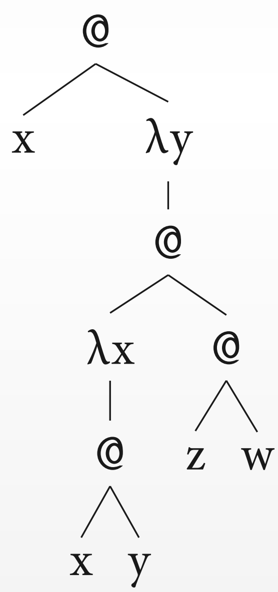
$$x\ y\ z = (x\ y)\ z \neq x\ (y\ z)$$

3. Notação simplificada para λ's seguidos.

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

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

$$\Updownarrow$$



Sintaxe: termos com nomes especiais

Alguns pré-termos são famosos e possuem nomes especiais.

Combinadores (lógica combinatorial)

$$\begin{aligned}
\mathbf{I} &= \lambda x.x \\
\mathbf{K} &= \lambda x\ y.x \\
\mathbf{S} &= \lambda x\ y\ z.(x\ z\ (y\ z))
\end{aligned}$$

Auto-aplicação e ponto fixo

$$\begin{aligned}
\omega &= \lambda x.x\ x \\
\Omega &= \omega\ \omega \\
\mathbf{Y} &= \lambda f.\ (\lambda x.\ f(x\ x))(\lambda x.\ f(x\ x))
\end{aligned}$$

A utilidade desses termos será vista posteriormente.

Variáveis livres e ligadas

Definição: dentro do pré-termo $\lambda x.M$, dizemos que M é o **escopo** de λx.

Definição: uma ocorrência de x dentro do escopo de λx é dita **ligada**. Caso contrário, x ocorre **livre**.

Exemplo:

a)	$\lambda x.x\ y$	x é ligada, y é livre
b)	$\lambda x.z\ \lambda z.z\ x$	z é livre, z e x são ligadas
c)	$\lambda x.x\ \lambda x.x\ y$	x e x são ligadas, y é livre

Nota:

1. Um mesmo nome por ter ocorrências ligadas e livres no mesmo pré-termo (b).
2. Uma ocorrência de x é sempre ligada ao λx mais interno (c).

Variáveis livres e ligadas (2)

Ocorrências livres e ligadas de variáveis têm significados distintos:

- variáveis livres referem-se a nomes externos (globais).
- variáveis ligadas referem-se a parâmetros formais (locais).

Nomes de *variáveis livres* são importantes:

$$\text{expressões distintas} \begin{cases} \sin(\pi) - 42 + \pi^2 \\ \sin(e) - 42 + e^2 \end{cases}$$

Nomes de *parâmetros formais* não são importantes:

$$\text{mesma definição} \begin{cases} \lambda x.\, \sin(x) - 42 + x^2 \\ \lambda e.\, \sin(e) - 42 + e^2 \end{cases}$$

Variáveis livres e ligadas (3)

Definição: a função FV computa as variáveis que ocorrem livres em um pré-termo.

$$
\begin{aligned}
FV(x) &= \{x\} \\
FV(\lambda x.M) &= FV(M) - \{x\} \\
FV(M\ N) &= FV(M) \cup FV(N)
\end{aligned}
$$

Definição: um pré-termo M onde $FV(M) = \{\}$ é **fechado**, também chamado **combinador**. Caso contrário, ele é **aberto**.

Operação de substituição

A **operação de substituição**

$$M[x := N]$$

substitui todas as **ocorrências livres** de x em M por N.

Exemplo:

$$\begin{array}{lllll}
1) & (\lambda x.xy)[x := w] & = & \lambda x.xy \\
2) & (\lambda x.xy)[y := w] & = & \lambda x.xw \\
3) & (\lambda x.xy)[z := w] & = & \lambda x.xy \\
4) & (z\lambda z.z)[z := w] & = & w\lambda z.z \\
5) & (zz)[z := \lambda z.zz] & = & (\lambda z.zz)(\lambda z.zz) \\
6) & (\lambda x.xy)[y := x] & = & (\lambda x.xx)?
\end{array}$$

Pergunta: alguém nota algo estranho com a substituição 6?

Captura de variáveis livres

Problema do Ex. 6: **captura de variáveis livres**

$$\overbrace{(\lambda x.xy)}^{M}\ [y := \overbrace{x}^{N}\,]$$

- Considere M: na posição onde está y, o nome x é ligado
- Considere N: o nome x ocorre livre
- Ao substituir y por N, podemos colocar um x livre em uma posição onde o mesmo nome x está ligado.
- Mas nomes livres e ligados possuem interpretações distintas, e isso *altera* o significado do termo.
- Vamos querer evitar esse problema ao definirmos substituição.

Operação de substituição: definição formal

Definição: **operação de substituição** evitando captura de variáveis livres

$$x[y := P] = \begin{cases} P & \text{se } x = y \\ x & \text{caso contrário} \end{cases}$$

$$(\lambda x.M)[y := P] = \begin{cases} \lambda x.M & \text{se } x = y \\ \lambda x.(M[y := P]) & \text{se } x \neq y \text{ e } x \notin FV(P) \end{cases}$$

$$(M\ N)[y := P] = M[y := P]\ \ N[y := P]$$

Pergunta: segundo esta definição, qual o resultado de $(\lambda x.xy)[y := x]$?

Resposta: indefinido!

Equivalência alfa

Definição: dois pré-termos M e N são α-equivalentes ($M =_\alpha N$) sss eles diferem somente na escolha dos nomes de variáveis ligadas.

Exemplo:

$$\lambda x.xy =_\alpha \lambda z.zy \qquad \lambda x.xy \neq_\alpha \lambda y.yy$$
$$\lambda x.xx =_\alpha \lambda z.zz \qquad \lambda y.zy \neq_\alpha \lambda y.xy$$

O conceito de α-equivalência captura a intuição que a escolha dos nomes dos parâmetros formais realmente não é importante.

Equivalência alfa: definição formal

Definição: $=_\alpha$ é a menor relação de equivalência sobre Λ^- tal que

$$\frac{y \notin FV(M)}{\lambda x.M =_\alpha \lambda y.(M[x := y])} \ (\alpha) \qquad \frac{M =_\alpha M'}{N\ M =_\alpha N\ M'}$$

$$\frac{M =_\alpha M'}{M\ N =_\alpha M'\ N} \qquad \frac{M =_\alpha M'}{\lambda x.M =_\alpha \lambda x.M'}$$

Termos lambda

Definição: um **termo lambda** é uma *classe de equivalência* de pré-termos α-equivalentes.

Notação: vamos denotar por $\{M\}_\alpha$ o termo lambda contendo o pré-termo M.

Exemplo:

$$\{\lambda x.x\}_\alpha = \{\ \lambda x.x,\ \lambda y.y,\ \lambda z.z, \ldots\}$$
$$\{\lambda x.x\ y\}_\alpha = \{\ \lambda a.a\ y,\ \lambda b.b\ y, \ldots\ \}$$

Definição: Λ é conjunto de todos os termos lambda:

$$\Lambda = \frac{\Lambda^-}{=_\alpha}$$

Termos lamba: operação de substituição

Operações definidas em Λ^- podem ser estendidas para Λ.

Lembre:

$$(\lambda x.xy)[y := x] \Rightarrow \text{indefinido}$$

Porém, como

$$\lambda x.xy =_\alpha \lambda a.ay$$

e

$$(\lambda a.ay)[y := x] = \lambda a.ax$$

podemos estender $M[x := N]$ para termos, obtendo

$$\{\lambda x.xy\}_\alpha[y := \{x\}_\alpha] = \{\lambda a.ax\}_\alpha$$

Efeito: temos em Λ uma operação de substituição que é *total* e *bem-definida* (evita captura de variáveis livres).

Termos vs pré-termos

No que segue, vamos falar de termos e não mais de pré-termos.

Contudo, vamos escrever termos usando M ao invés de $\{\mathrm{M}\}_\alpha$, e assumir que a operação de substituição é total e evita a captura de variáveis livres.

Essa convenção é também chamada *convenção de Barendregt*, devido à sua utilização no livro deste autor.

A principal justificativa é reduzir a poluição excessiva da notação.