Lambda-calculus: fundamentos de Programação Funcional

Mário Florido

17 de outubro de 2022

Lambda-calculus

- Modelo computação universal (i.e., equipotente à máquina de Turing)
- Ao contrário da MT, o cálculo-λ é também um modelo para linguagens de programacão:
  - âmbito de variáveis
  - ordem de computaçãoo
  - estruturas de dados
  - recursão
  - ...
- As linguagens funcionais são implementações computacionais do cálculo-λ (Landin, 1964).

Bibliografia

1. *Foundations of Functional Programming*, notas de um curso de Lawrence C. Paulson, Computer Laboratory, University of Cambrige.
2. *Lambda Calculi: a guide for computer scientists*, Chris Hankin, Graduate Texts in Computer Science, Oxford University Press.

Plano

Sintaxe

Reduções

Computação

Plano

Sintaxe

Reduções

Computação

Termos do cálculo-λ

$x, y, z, \ldots$	uma variável é um termo;
$(\lambda x\, M)$	é um termo se x é variável e M é um termo;
$(M\, N)$	é um termo se M e N são termos.

exemplos de termos	**não são termos**
y	$()$
$(\lambda x\, y)$	$x \lambda y$
$((\lambda x\, y)\, (\lambda x\, (\lambda x\, y)))$	$x(y)$
$(\lambda y\, (\lambda x\, (y\, (y\, x))))$	$(\lambda x\, (\lambda y\, y)$

Interpretação do cálculo-λ

$(\lambda x\, M)$ função com parâmetro x e corpo M.

$(M\, N)$ aplicação de M ao argumento N.

Exemplos:

$(\lambda x\, x)$ *função identidade*: a x faz corresponder x

$(\lambda x\, (\lambda y\, x))$

- Não há distinção entre dados e programas;
- Não há constantes (e.g. números).
- Tudo são λ-termos!

Convenção de parêntesis

Abreviaturas:

$$\begin{array}{rcl} \lambda x_1 x_2 \ldots x_n . M & \equiv & (\lambda x_1 (\lambda x_2 \ldots (\lambda x_n M) \ldots)) \\ (M_1 M_2 \ldots M_n) & \equiv & (\ldots (M_1 M_2) \ldots M_n) \end{array}$$

Exemplos:

$$\begin{array}{rcl} \lambda xy . x & \equiv & (\lambda x (\lambda y x)) \\ \lambda xyz . xz(yz) & \equiv & (\lambda x (\lambda y (\lambda z ((x z) (y z))))) \end{array}$$

Substituição

$M[N/x]$ o termo resultante da substituição de x em M por N.

$$(\lambda x\, y)[(z\, z)/y] \equiv (\lambda x\, (z\, z))$$
$$(\lambda x\, y)[(z\, z)/x] \equiv (\lambda x\, y)$$

Plano

Sintaxe

Reduções

Computação

Conversão-β

$$((\lambda x\, M)\, N) \rightarrow_\beta M[N/x] \qquad \text{se } BV(M) \cap FV(N) = \emptyset$$

Exemplo:

$$((\lambda x\, \underbrace{(x\, x)}_{M})\, \underbrace{(y\, z)}_{N}) \rightarrow_\beta (x\, x)[(y\, z)/x] \equiv ((y\, z)\, (y\, z))$$

Conversão-β

$$((\lambda x\, M)\, N) \rightarrow_\beta M[N/x] \qquad \text{se } BV(M) \cap FV(N) = \emptyset$$

Exemplo:

$$((\lambda x\, \underbrace{(x\, x)}_{M})\, \underbrace{(y\, z)}_{N}) \rightarrow_\beta (x\, x)[(y\, z)/x] \equiv ((y\, z)\, (y\, z))$$

Corresponde à invocaçãoo de uma função:

- x o parâmetro;
- M o corpo da função;
- N o argumento.

Currying

Não necessitamos de abstracções de duas ou mais variáveis:

$$\lambda xy.\, M \quad \equiv \quad (\lambda x\, (\lambda y\, M))$$

Substituimos os argumentos um de cada vez:

$$\begin{aligned}((\lambda xy.\, M)\, P\, Q) &\equiv (((\lambda x\, (\lambda y\, M))\, P)\, Q) \\ &\to_\beta ((\lambda y\, M)[P/x]\, Q) \\ &\to_\beta M[P/x][Q/y]\end{aligned}$$

Esta codificação chama-se "*currying*" em homenagem a Haskell Curry.

Reduções

Escrevemos $M \to N$ se M reduz num passo a N usando conversão β.

Escrevemos $M \twoheadrightarrow N$ para a redução em múltiplos passos ($\to^*$).

Forma normal

Se não existir N tal que $M \to N$, então M está em formal normal.

M admite forma normal N se $M \twoheadrightarrow N$ e N está em forma normal.

Exemplo:

$$(\lambda x.\, a\, x)\, ((\lambda y.\, b y)\, c) \to a((\lambda y.\, b y)\, c) \to a(bc) \not\to$$

Logo: $(\lambda x.\, a\, x)\, ((\lambda y.\, b y)\, c)$ admite forma normal $a(bc)$.

Analogia: resultado de uma computação.

Termos sem forma normal

Nem todos os termos admitem forma normal:

$$\begin{aligned}\Omega &\equiv ((\lambda x.\, x\, x)\, (\lambda x.\, x\, x)) \\ &\to_\beta (x\, x)[(\lambda x.\, x\, x)/x] \\ &\equiv ((\lambda x.\, x\, x)\, (\lambda x.\, x\, x)) \equiv \Omega\end{aligned}$$

Logo:

$$\Omega \to \Omega \to \Omega \to \cdots$$

Analogia: uma computação que não termina.

Confluência

Podemos efectuar reduções por ordens diferentes.

Exemplo:

$$\underline{(\lambda x.\, a\, x)\,((\lambda y.\, b y)\, c)} \to a(\underline{(\lambda y.\, b y)\, c}) \to a(bc) \not\to$$

$$(\lambda x.\, a\, x)\,(\underline{(\lambda y.\, b y)\, c}) \to \underline{(\lambda x.\, a\, x)\,(bc)} \to a(bc) \not\to$$

P: Será que chegamos sempre à mesma forma normal?

Confluência

Podemos efectuar reduções por ordens diferentes.

Exemplo:

$$\underline{(\lambda x.\, a\, x)\,((\lambda y.\, by)\, c)} \to a(\underline{(\lambda y.\, by)\, c}) \to a(bc) \not\to$$

$$(\lambda x.\, a\, x)\,(\underline{(\lambda y.\, by)\, c}) \to \underline{(\lambda x.\, a\, x)\,(bc)} \to a(bc) \not\to$$

P: Será que chegamos sempre à mesma forma normal?

R: Sim (Teorema de Church-Rosser)

Estratégias de redução

Como reduzir $(M\,N) \twoheadrightarrow P$?

ordem normal: reduzir M e substituir N sem reduzir.

1. $M \twoheadrightarrow (\lambda x\, M')$
2. $M'[N/x] \twoheadrightarrow P$

ordem aplicativa: reduzir M e N antes de fazer a substituição da variável.

1. $M \twoheadrightarrow (\lambda x\, M')$
2. $N \twoheadrightarrow N'$
3. $M'[N'/x] \twoheadrightarrow P$

Alguns factos sobre reduções

1. Se a redução por ambas as estratégias termina, então chegam à mesma forma normal
2. Se um termo admite forma normal, esta pode sempre obtida pela redução em ordem normal
3. A redução em ordem aplicativa pode não terminar mesmo quando existe forma normal
4. A redução em ordem normal pode reduzir o mesmo termo várias vezes

Ordem aplicativa: não terminação

Seja $\Omega \equiv ((\lambda x.\, x\, x)\, (\lambda x.\, x\, x))$; vamos reduzir

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

Redução em ordem normal

$$\underline{((\lambda x.\, y)\, \Omega)} \rightarrow_\beta y \quad \text{forma normal}$$

Redução em ordem aplicativa

$$((\lambda x.\, y)\, \underline{\Omega}) \rightarrow_\beta ((\lambda x.\, y)\, \underline{\Omega}) \rightarrow_\beta \cdots \quad \text{não termina}$$

Ordem normal: computação redudante

Supondo **mult** um termo tal que

$$\textbf{mult}\ N\ M \to N \times M$$

para codificações N, M de números naturais (veremos como mais à frente).

Definindo

$$\textbf{sqr} \equiv \lambda x.\, \textbf{mult}\ x\ x$$

vamos reduzir

$$\textbf{sqr}\,(\textbf{sqr}\ N)$$

Ordem normal: computação redudante

Redução em ordem aplicativa:

$$\textbf{sqr}\,(\textbf{sqr}\,N) \to \textbf{sqr}\,(\textbf{mult}\,N\,N) \to \textbf{sqr}\,N^2 \to \textbf{mul}\,N^2\,N^2$$

Redução em ordem normal:

$$\begin{aligned}\textbf{sqr}\,(\textbf{sqr}\,N) &\to \textbf{mult}\,(\textbf{sqr}\,N)\,(\textbf{sqr}\,N) \\ &\to \textbf{mult}\,\underbrace{(\textbf{mult}\,N\,N)\,(\textbf{mult}\,N\,N)}_{\text{duplicação}}\end{aligned}$$

Plano

Sintaxe

Reduções

Computação

Computação usando cálculo-λ

O cálculo-λ é um modelo de computação universal: qualquer função recursiva (computável por uma máquina de Turing) pode ser codificada no cálculo-λ.

Computação usando cálculo-λ

Estruturas de dados como boleanos, inteiros, listas, etc. não são primitivas do cálculo-λ.

Estas estruturas podem ser definidas usando apenas o cálculo puro.

Contudo: implementações de linguagens funcionais usam representações optimizadas por razões de eficiência.

Valores boleanos

Definimos:

$$\begin{aligned} \textbf{true} &\equiv \lambda xy.\, x \\ \textbf{false} &\equiv \lambda xy.\, y \\ \textbf{if} &\equiv \lambda pxy.\, pxy \end{aligned}$$

Então:

$$\begin{aligned} \textbf{if true}\ M\ N &\twoheadrightarrow M \\ \textbf{if false}\ M\ N &\twoheadrightarrow N \end{aligned}$$

Exercício: verificar as reduções acima.

Pares ordenados

Um *constructor* e dois *selectores*:

$$\begin{aligned} \mathbf{pair} &\equiv \lambda xyf.\, fxy \\ \mathbf{fst} &\equiv \lambda p.\, p\,\mathbf{true} \\ \mathbf{snd} &\equiv \lambda p.\, p\,\mathbf{false} \end{aligned}$$

Temos:

$$\begin{aligned} \mathbf{fst}\,(\mathbf{pair}\,M\,N) &\twoheadrightarrow \mathbf{fst}\,(\lambda f.\, f\,M\,N) \\ &\to (\lambda f.\, f\,M\,N)\,\mathbf{true} \\ &\to \mathbf{true}\,M\,N \\ &\twoheadrightarrow M \end{aligned}$$

Analogamente: $\mathbf{snd}\,(\mathbf{pair}\,M\,N) \twoheadrightarrow N$.

Codificar números naturais

Usando numerais de Church:

$$\underline{0} \equiv \lambda fx.\, x$$
$$\underline{1} \equiv \lambda fx.\, f\, x$$
$$\underline{2} \equiv \lambda fx.\, f\,(f\, x)$$
$$\vdots$$
$$\underline{n} \equiv \lambda fx.\, \underbrace{f\,(\ldots(f}_{n \text{ vezes}}\, x)\ldots)$$

Intuição: $\underline{n}$ itera uma função n vezes.

Aritmética

$$\begin{aligned}
\mathbf{succ} &\equiv \lambda nfx.\, f\,(n\,f\,x) \\
\mathbf{iszero} &\equiv \lambda n.\, n\,(\lambda x.\, \mathbf{false})\,\mathbf{true} \\
\mathbf{add} &\equiv \lambda mnfx.\, m\,f\,(n\,f\,x)
\end{aligned}$$

Verificar:

$$\begin{aligned}
\mathbf{succ}\,\underline{n} &\twoheadrightarrow \underline{n+1} \\
\mathbf{iszero}\,\underline{0} &\twoheadrightarrow \mathbf{true} \\
\mathbf{iszero}\,(\underline{n+1}) &\twoheadrightarrow \mathbf{false} \\
\mathbf{add}\,\underline{n}\,\underline{m} &\twoheadrightarrow \underline{n+m}
\end{aligned}$$

Analogamente: subtração, multiplicação, expoente, etc.

Listas

$$[x_1,\ x_2, \ldots,\ x_n] \quad \simeq \quad \mathbf{cons}\, x_1\, (\mathbf{cons}\, x_2\, (\ldots (\mathbf{cons}\, x_n\, \mathbf{nil}) \ldots))$$

Dois constructores, teste da lista vazia e dois selectores:

$$\begin{aligned}
\mathbf{nil} &\equiv \lambda z.\, z \\
\mathbf{cons} &\equiv \lambda xy.\, \mathbf{pair}\, \mathbf{false}\, (\mathbf{pair}\, x\, y) \\
\mathbf{null} &\equiv \mathbf{fst} \\
\mathbf{hd} &\equiv \lambda z.\, \mathbf{fst}\, (\mathbf{snd}\, z) \\
\mathbf{tl} &\equiv \lambda z.\, \mathbf{snd}\, (\mathbf{snd}\, z)
\end{aligned}$$

Listas

Verificar:

$$\mathbf{null}\ \mathbf{nil} \twoheadrightarrow \mathbf{true} \tag{1}$$

$$\mathbf{null}\ (\mathbf{cons}\ M\ N) \twoheadrightarrow \mathbf{false} \tag{2}$$

$$\mathbf{hd}\ (\mathbf{cons}\ M\ N) \twoheadrightarrow M \tag{3}$$

$$\mathbf{tl}\ (\mathbf{cons}\ M\ N) \twoheadrightarrow N \tag{4}$$

NB: (2), (3), (4) resultam das propriedades de pares, mas (1) não.

Variáveis locais

$$\textbf{let } x = M \textbf{ in } N$$

Exemplo:

$$\begin{aligned} \textbf{let } f &= \lambda x.\, \textbf{add } x\ x \\ \textbf{in } &\lambda x.\, f\ (f\ x) \end{aligned}$$

Tradução para o cálculo-λ

Definimos:

$$\textbf{let } x = M \textbf{ in } N \equiv (\lambda x.\, N)\, M$$

Então:

$$\textbf{let } x = M \textbf{ in } N \twoheadrightarrow N[M/x]$$

Declarações imbricadas

Não necessitamos de sintaxe extra:

$$\begin{aligned} & \textbf{let } \{x = M;\ y = N\} \textbf{ in } P \\ \equiv\ & \textbf{let } x = M \textbf{ in } (\textbf{let } y = N \textbf{ in } P) \end{aligned}$$

Declarações recursivas

Tentativa:

$$\textbf{let } f = \lambda x.\ \textbf{if } (\textbf{iszero } x)\ \underline{1}\ (\textbf{mult } x\ (f\ (\textbf{sub } x\ \underline{1})))$$
$$\textbf{in } f\ \underline{5}$$

Declarações recursivas

Tentativa:

$$\begin{array}{l}\textbf{let } f = \lambda x.\ \textbf{if } (\textbf{iszero } x)\ \underline{1}\ (\textbf{mult } x\ (f\ (\textbf{sub } x\ \underline{1})))\\ \textbf{in } f\ \underline{5}\end{array}$$

Tradução:

$$(\lambda f.\, f\ \underline{5})\ (\lambda x.\, \textbf{if } (\textbf{iszero } x)\ \underline{1}\ (\textbf{mult } x\ (f\ (\textbf{sub } x\ \underline{1}))))$$

Não define uma função recursiva porque f ocorre livre no corpo da definição.

Declarações recursivas: combinadores ponto-fixo

Soluçãoo: usar um combinador ponto-fixo i.e. um termo **Y** tal que

$$\mathbf{Y}\,F = F\,(\mathbf{Y}\,F) \quad \text{para qualquer termo } F$$

Definimos o factorial recursivo como:

$$\begin{array}{l}\textbf{let } f = \mathbf{Y}\,(\lambda g\,x.\,\textbf{if } (\textbf{iszero } x)\ \underline{1}\ (\textbf{mult } x\ (g\ (\textbf{sub } x\ \underline{1}))))\\ \textbf{in } f\ \underline{5}\end{array}$$

Note que g ocorre ligada no corpo da funçãoo.

Definições recursivas: combinador ponto-fixo

Seja:

$$\mathbf{Y}\ F = F\ (\mathbf{Y}\ F) \quad \text{para qualquer } M$$
$$\text{fact} \equiv \mathbf{Y}\ (\lambda g\, x.\, \mathbf{if}\ (\mathbf{iszero}\ x)\ \underline{1}\ (\mathbf{mult}\ x\ (g\ (\mathbf{sub}\ x\ \underline{1}))))$$

Calculemos:

$$\begin{aligned}
\text{fact}\ \underline{5} &\equiv \mathbf{Y}\ (\lambda g\, x.\, \ldots)\ \underline{5} \\
&= (\lambda g\, x.\, \ldots)\ \underbrace{(\mathbf{Y}\ (\lambda g\, x.\, \ldots))}_{\text{fact}}\ \underline{5} \\
&\twoheadrightarrow \mathbf{if}\ (\mathbf{iszero}\ \underline{5})\ \underline{1}\ (\mathbf{mult}\ \underline{5}\ (\text{fact}\ (\mathbf{sub}\ \underline{5}\ \underline{1}))) \\
&\twoheadrightarrow \mathbf{if\ false}\ \underline{1}\ (\mathbf{mult}\ \underline{5}\ (\text{fact}\ \underline{4})) \\
&\twoheadrightarrow \mathbf{mult}\ \underline{5}\ (\text{fact}\ \underline{4}) \\
&\twoheadrightarrow \mathbf{mult}\ \underline{5}\ (\mathbf{mult}\ \underline{4}\ (\ldots\ (\mathbf{mult}\ \underline{1}\ \underline{1})\ldots)) \equiv \underline{120}
\end{aligned}$$

Combinadores ponto-fixo

Y pode ser definido no cálculo-λ puro (Haskell B. Curry):

$$\mathbf{Y} \equiv \lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x))$$

Verificação:

$$\begin{aligned}\mathbf{Y}\ F &\rightarrow (\lambda x.\,F(xx))\,(\lambda x.\,F(xx)) \\ &\rightarrow F((\lambda x.\,F(xx))\,(\lambda x.\,F(xx))) \\ &\leftarrow F(\mathbf{Y}\,F)\end{aligned}$$

Logo

$$\mathbf{Y}\ F = F(\mathbf{Y}\,F)$$

Há uma infinidade de outros combinadores ponto-fixo (ver a bibliografia).