Introdução ao cálculo-λ

Pedro Vasconcelos

3 de Abril de 2020

O que é o cálculo-λ?

- É um modelo computação universal (equivalente à máquina de Turing)
- Ao contrário da MT, o cálculo-λ é um modelo para linguagens de programação:
  - âmbito de variáveis
  - ordem de computação
  - estruturas de dados
  - recursão
  - ...
- As linguagens funcionais são concebidas como implementações computacionais do cálculo-λ (linguagem *ISWIM* de Peter 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

1. Sintaxe
2. Reduções e normalização
3. Computação

Plano

1. Sintaxe
2. Reduções e normalização
3. 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

$(\lambda x\, M)$ é a abstração de x em M.

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

Exemplos:

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

$(\lambda x\, (\lambda y\, x))$ é a função para cada x dá *uma outra função* que para cada y dá x

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

Convenções 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:

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

Ocorrências de variáveis livres e ligadas

As ocorrências de x em $(\lambda x\, M)$ dizem-se ligadas pela abstracção.

Uma ocorrência que não é ligada diz-se livre.

$$(\lambda z\,(\lambda x\,(y\;x))) \quad x \text{ ligada}, \;\; y \text{ livre}$$

Uma variável pode ocorrer livre e ligada no mesmo termo.

$$((\lambda x\; x)\,(\lambda y\; x)) \quad x \text{ ligada}, \;\; x \text{ livre}$$

Ocorrências livres e ligadas de variáveis

$BV(M)$ é o conjunto de variáveis com ocorrências ligadas em M (*bound variables*);

$FV(M)$ é o conjunto das variáveis com ocorrências livres em M (*free variables*).

$$BV(\lambda z\,(\lambda x\,(y\,x))) = \{x,\, z\}$$
$$FV(\lambda z\,(\lambda x\,(y\,x))) = \{y\}$$
$$BV((\lambda x\,x)\,(\lambda y\,x)) = \{x,\, y\}$$
$$FV((\lambda x\,x)\,(\lambda y\,x)) = \{x\}$$

Definidas por recursão sobre o termo (ver a bibliografia).

Substituição

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

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

Nota: apenas substitui ocorrências livres de x.

Mudança de variáveis ligadas

Os nomes de variáveis ligadas não são relevantes.
Exemplo: os dois programas seguintes são equivalentes.

```
int f(int x,int y) {
  return x+y;
}
```

```
int f(int a,int b) {
  return a+b;
}
```

A conversão-α formaliza esta noção de equivalência.

Conversão-α

$$(\lambda x\, M) \to_\alpha (\lambda y\, M[y/x]) \qquad \text{se } y \notin BV(M) \cup FV(M)$$

Exemplos:

$$\lambda x.\, xy \to_\alpha \lambda z.\, xy[z/x] \equiv \lambda z.\, zy$$

$$\lambda x.\, xy \not\to_\alpha \lambda y.\, xy[y/x] \equiv \lambda y.\, yy \qquad \text{porque } y \in FV(xy)$$

Equivalência-α

Consideramos $M \simeq N$ se $M \to_\alpha N$.

Exemplo:

$$\lambda x.\, xy \simeq \lambda z.\, zy$$

Mais geralmente:

$$M \simeq N \quad \text{se} \quad M \equiv M_0 \to_\alpha M_1 \to_\alpha M_2 \to_\alpha \cdots \to_\alpha M_k \equiv N$$

com $k \geq 0$.

Plano

1. Sintaxe
2. Reduções e normalização
3. Computação

Conversão-β

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

Exemplo:

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

Corresponde à invocação de uma função:

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

Captura de variáveis

A condição $BV(M) \cap FV(N) = \emptyset$ é necessária para evitar captura de variáveis:

$$((\lambda x \overbrace{(\lambda y\, x)}^{M})\ \overbrace{y}^{N}) \rightarrow_\beta (\lambda y\, x)[y/x] \qquad y \in BV(M) \cap FV(N) \neq \emptyset$$
$$\equiv (\lambda y\, y)$$

Usamos conversões-α para evitar a captura da variável y:

$$((\lambda x\, (\lambda y\, x))\, y) \rightarrow_\alpha ((\lambda x \overbrace{(\lambda z\, x)}^{M})\ \overbrace{y}^{N})$$
$$\rightarrow_\beta (\lambda z\, x)[x/y] \qquad BV(M) \cap FV(N) = \emptyset$$
$$\equiv (\lambda z\, y)$$

Conversão-η

$$(\lambda x\,(M\,x)) \to_\eta M$$

- Simplifica uma abstracção redudante:

$$((\lambda x\,(M\,x))\,N) \to_\beta (M\,N) \quad \text{logo} \quad (\lambda x\,(M\,x)) \simeq M$$

- Apenas necessária para garantir a unicidade da forma normal (mais à frente)
- Não é necessária para a implementação de linguagens funcionais

Currying

Não necessitamos de abstrações de duas ou mais variáveis:

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

Substituimos os argumentos um de cada vez:

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

Esta codificação chama-se "*currying*" como referência ao nome do lógico Haskell Curry (embora tenha sido inventada anteriormente por Moses Schönfinkel).

Reduções

Escrevemos $M \to N$ se M reduz num passo a N usando conversões β ou η.

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

Igualdade

Escremos $M = N$ se M se pode converter em N por zero ou mais reduções ou expansões; ou seja, a relação $(\rightarrow \cup \rightarrow^{-1})^*$.

Exemplo:

$$a((\lambda y.\, by)c) = (\lambda x.\, ax)(bc)$$

porque

$$a((\lambda y.\, by)c) \rightarrow a(bc) \leftarrow (\lambda x.\, ax)(bc)$$

Intuição: se $M = N$ então M e N são termos com o mesmo "resultado".

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:

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

Logo: $(\lambda x.\, a\, x)\,((\lambda y.\, by)\, c)$ 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:

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

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

P: Será que chegamos sempre à mesma forma normal?

R: Sim (Teorema de Church-Rosser)

Confluência

Teorema (Church-Rosser)

Se $M = N$ então existe L tal que $M \twoheadrightarrow L$ e $N \twoheadrightarrow L$.

A demonstração baseia-se na seguinte propriedade:

Diamond property

Se $\begin{tikzcd} & M \arrow[dl, twoheadrightarrow] \arrow[dr, twoheadrightarrow] & \\ M_1 & & M_2 \end{tikzcd}$ então existe L tal que $\begin{tikzcd} M_1 \arrow[dr, twoheadrightarrow] & & M_2 \arrow[dl, twoheadrightarrow] \\ & L & \end{tikzcd}$.

Mais informação: ver a bibliografia.

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

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

Redução em ordem aplicativa

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

Ordem normal: computação redudante

Supondo **mult** um termo tal que

$$\mathbf{mult}\; N\; M \to N \times M$$

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

Definindo

$$\mathbf{sqr} \equiv \lambda x.\, \mathbf{mult}\; x\; x$$

vamos reduzir

$$\mathbf{sqr}\,(\mathbf{sqr}\; N)$$

Ordem normal: computação redudante

Redução em ordem aplicativa:

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

Redução em ordem normal:

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

Plano

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.

Esta omissão não é fundamental: 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:

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

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 então:

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

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

Codificar números naturais

Usando numerais de Church:

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

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

Operações aritméticas

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

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: subtracção, multiplicação, exponenciação, 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.

Declarações

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

Exemplo:

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

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:

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

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

Solução: 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:

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

Note que g ocorre ligada no corpo da função.

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:

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

Logo

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

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