Aviso Prévio

Bibliografia

Introdução

Notações e Teoria . . .

Reduções e Igualdade

Propriedades

Teoria da Computação
Introdução ao Cálculo Lambda

Simão Melo de Sousa

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

1. Aviso Prévio

- A redacção deste documento baseou-se fortemente na bibliografia indicada. Parece-nos então óbvio que a leitura e a aprendizagem directa pelas obras originais é recomendada, e mesmo essencial à compreensão profunda das noções aqui apresentadas;
- O português não é a língua materna do autor e o presente documento encontra-se em fase de elaboração pelo que se agradece e até se incentiva qualquer sugestão ou correcção.

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

2. Bibliografia

Consultar [1, 2, 3, 4, 5, 6, 7]

3. Introdução

Desenvolvemos no primeiro capítulo desta disciplina uma primeira argumentação sublinhando a relevância do cálculo λ. Vamos, nesta introdução, relembrar o papel que o cálculo λ tem na informática (ou mais precisamente na ciência da computação) e na matemática.

O cálculo λ e a matemática

- Crise dos fundamentos: na sua tentativa de recriar os fundamentos da matemática o lógico Alonzo Church elaborou uma primeira versão (que, infelizmente, também acabou por ser paradoxal) do cálculo λ. No entanto extensões do cálculo λ original resolveram esta questão. Historicamente o cálculo λ foi um formalismo em que questões centrais da presente disciplina (formalização da noção de algoritmo, computabilidade de funções, decidibilidade de problemas) foram exploradas.

- Matemática construtiva: uma escola importante de matemáticos, a escola da matemática construtiva (o matemático pioneiro desta escola tem por nome J.L.L. Brower) baseou a sua disciplina na recusa das demonstrações envolvendo o raciocínio por absurdo: uma demonstração deve ser uma *construção* de uma argumentação verificável de que a propriedade é verdade. Como tal, basear uma demonstração sobre o argumento "é verdade porque não pode ser de outra forma" (é informalmente uma tradução do raciocínio por absurdo) é banido. O cálculo λ tem um papel central nesta matemática: pode servir de fundamento a matemática construtiva.

- Demonstração por computador e Isomorfismo de Curry-Howard: uma visão alternativa da relação cálculo λ/matemática construtiva é a seguinte. Se uma implementação do cálculo λ é também uma implementação da matemática construtiva então programar em cálculo λ é de uma certa forma "fazer" matemática construtiva. A definição precisa desta ponte entre matemática construtiva e programação é designada por Isomorfismo de Curry-Howard. Assim o cálculo λ é uma forte contribuição a área da demonstração por computador ou genericamente a área da teoria da Demonstração.

O cálculo λ e a Ciência da Computação

- A função como cidadão de primeira classe: na teoria dos conjuntos, como o vimos nos capítulos anteriores, uma função, digamos f, é uma relação particular, logo é vista como o conjunto f de todos os pares (x, y) tais que $f(x) = y$. Neste contexto é relativamente fácil comparar funções: basta comparar o conjunto de pares associado (tal igualdade é designada por igualdade por extensão). O problema é que esta perspectiva não é de todo satisfatória para os informáticos (e para muitos matemáticos também):

   1. sejam f_A e f_B as funções que implementam um determinado cálculo mas utilizando algoritmos A e B diferentes (digamos que A é exponencial e B polinomial). Na teoria dos conjuntos $f_A = f_B$.
   2. o que nos interessa não é só que y é o resultado de $f(x)$ mas é como y é obtido/computado de x por f.

O cálculo λ foi desenvolvida como uma alternativa a teoria dos conjuntos na qual a noção de função e de cálculo têm um papel central.

- Concepção de linguagens de programação: já referimos no primeiro capítulo desta disciplina que o cálculo λ:

  1. esteve na origem do paradigma funcional. Por exemplo nos anos 1950, John McCarthy extendeu o cálculo λ para construir a primeira linguagem da família LISP. O Peter Landin construiu, nos anos 1960 e também como uma extensão do cálculo λ, a linguagem ISWIM, o antecessor das linguagens funcionais tipificadas como o OCaml;
  2. constituiu e continua a constituir um protótipo ideal para concepção de mecanismos avançados para linguagens de programação. Referiremos por exemplo as máquinas virtuais, os sistemas de tipos, os analisadores estáticos de programas, o polimorfismo, os tipos dependentes, etc...

- Semântica das linguagens de programação: nos anos 1960 Peter Landin demonstrou que o cálculo λ é um formalismo adequado para servir de modelo matemático quando se estudas linguagens de programação (imperativas). Nesta mesma vertente o Dana Scott e o Strachey contribuíram para este relacionamento. Em particular ao desenvolver a teoria dos domínios (nos quais se inserem os CPOs) e ao exibir uma relação forte entre essa teoria e o cálculo λ.

- Teoria da computação: historicamente, os primeiros resultados da teoria da computação foram desenvolvidos utilizando dois formalismos: as máquinas de Turing e o cálculo λ. Iremos neste capítulo e nos próximos desenvolver este ponto.

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

Página Pessoal
Página de Rosto
Página 8 de 100
Retroceder
Ecrã Todo
Fechar
Sair

4. Notações e Teoria Básica

Seja $\mathcal{V}$ um conjunto numerável (infinito) de variáveis. Utilizaremos preferencialmente as letras x, y, z ou versões numeradas (como x_i ou y_5, por exemplo) para designá-las.

Utilizaremos o símbolo $\equiv$ para designar a igualdade sintáctica: $a \equiv b$ se a <u>é</u> a palavra b.

Definição 4.1 (Termos λ) *A classe Λ dos termos λ é o subconjunto do monóide livre gerado a partir do alfabeto $\mathcal{V} \cup \{' \; ', '(', ')', '\lambda'\}$ definido indutivamente por:*

$$Var\frac{x \in \mathcal{V}}{x \in \Lambda}$$

$$App\frac{M \in \Lambda \qquad N \in \Lambda}{(M\ \ N) \in \Lambda} \qquad Abs\frac{M \in \Lambda}{(\lambda x.M) \in \Lambda}$$

Quando não houver riscos de confusão, designaremos por termos os termos λ.

Por convenção notaremos termos como $(\lambda x_1.(\lambda x_2.(\cdots(\lambda x_n.M))))$ por $\lambda x_1 \cdots x_n.M$ ou ainda por $\lambda \vec{x}.M$. Da mesma forma, preferiremos a notação $MN_1 \ldots N_n$ ou $M\vec{N}$ à notação original $(\cdots(MN_1)\cdots N_n)$.

O termo (MN) representa a aplicação da função representada por M ao valor representado por N. Por seu turno, o termo $(\lambda x.M)$ representa a função (aqui anónima) unária que do parâmetro x devolve o valor M. De facto, em OCaml $(\lambda x.M)$ escreve-se *directamente* (function x > M).

4.1. Variáveis

Se analisarmos os termos $\lambda x.(xy)$, $\lambda z.(zy)$ e $\lambda y.(yy)$ vemos que um termo define um divisão do conjunto das variáveis. Por exemplo, intuitivamente identificamos os dois primeiros termos. Chamar a variável "parâmetro" x ou z é pouco relevante. O que o é, é a mudança de nome implicar que todas as ocorrências de x "debaixo" do λ [1] serem também alvo da troca. Em termos formais, essa troca é designada por *substituição*.

No entanto vemos, com o terceiro exemplo que a operação de substituição não é uma operação tão simples como parece. Intuitivamente não identificamos $\lambda y.(yy)$ com $\lambda x.(xy)$. O primeiro representa uma função que aceita um parâmetro y e devolve a aplicação de y a ele próprio. O segundo representa uma função que aplica o seu parâmetro x a uma variável y (que não está necessariamente ligada a x).

A distinção está na "qualidade" das variáveis. Em $\lambda x.(xy)$, a variável x está *ligada* ao λ enquanto y não. Diz-se neste caso que y é *livre*.

Essa distinção surge naturalmente nas linguagens de programação. Uma variável ligada pode ser um parâmetro ou uma variável local duma função ou procedimento. Uma variável livre, no caso duma função, pode ser uma variável global definido fora da função considerada.

Dado um termo, é relativamente fácil determinar o conjunto das suas variáveis livre e o conjunto das suas variáveis ligadas. Consideramos, por exemplo, as duas funções recursivas $BV : \Lambda \rightarrow \wp(\mathcal{V})$ e $FV : \Lambda \rightarrow \wp(\mathcal{V})$ seguintes calculam respectivamente o conjunto das variáveis ligadas dum termo e o conjunto das variáveis livres.

[1] Veremos a seguir uma definição precisa da noção de "debaixo".

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

$$BV(M) = \begin{cases} \emptyset & se\ M = x \\ BV(P) \cup \{x\} & se\ M = (\lambda x.P) \\ BV(P) \cup BV(Q) & se\ M = (PQ) \end{cases}$$

$$FV(M) = \begin{cases} \{x\} & se\ M = x \\ FV(P) - \{x\} & se\ M = (\lambda x.P) \\ FV(P) \cup FV(Q) & se\ M = (PQ) \end{cases}$$

Designaremos por Λ^0 a classe dos termos *fechados* (também designados por *combinadores*), isto é a classe dos termos sem variáveis livres: $t \in \Lambda^0$ se $FV(t) = \emptyset$.

4.2. Subtermos

Uma noção importante nas linguagens de termos, na qual se insere o cálculo λ é a noção de subtermos. Um subtermo de um termo t é uma palavra contida em t que também é λ-termo. O conjunto dos subtermos de um λ-termo M pode ser obtido pela função $Sub : \Lambda \to \wp(\Lambda)$ seguinte:

$$Sub(M) = \begin{cases} \{x\} & se\ M = x \\ Sub(P) \cup \{(\lambda x.P)\} & se\ M = (\lambda x.P) \\ Sub(P) \cup Sub(Q) \cup \{(PQ)\} & se\ M = (PQ) \end{cases}$$

4.3. Contextos

No âmbito o estudo do cálculo λ vamos precisar de estudar termos que não estão completamente definidos, isto é, que contêm “buracos”. Tais termos são qualificados de contextos, no sentido que podemos obter um termo "tradicional" preenchendo os buracos por termos λ. Se notarmos esses buracos por $[]$ então a classe dos contextos é definida da seguinte forma:

Definição 4.2 (Contextos λ)
A classe $\mathcal{C}[]$ dos contextos λ é a classe definida indutivamente por:

$$\frac{x \in \mathcal{V}}{x \in \mathcal{C}[]} \qquad \frac{}{[] \in \mathcal{C}[]} \qquad \frac{C_1 \in \mathcal{C}[] \quad C_2 \in \mathcal{C}[]}{(C_1 C_2) \in \mathcal{C}[]} \qquad \frac{C_1 \in \mathcal{C}[]}{(\lambda x.C_1) \in \mathcal{C}[]}$$

Para designar a operação que consiste em preencher o contexto aberto $C[]$ por um termo t utilizaremos a notação $C[t]$. Por exemplo, dado o contexto $C \triangleq ((\lambda x.[]x)M)$ e o termo $t \triangleq (\lambda y.y)$, o termo $C[t]$ é $((\lambda x.(\lambda y.y)x)M)$

Repare que as variáveis livre de t podem ficar ligadas no termo $C[t]$.

4.4. Teoria da Convertibilidade λ

Vamos agora apresentar a teoria da igualdade (ou convertibilidade) de termos λ. Utilizaremos mais uma vez o formalismo das definições indutivas para definir o que entendemos por igualdade de termos λ. Para esse efeito, utilizaremos a notação $M[x := N]$ para designar a operação de substituição que consiste em substituir todas as ocorrências livres de x em M por N. Daremos mais ênfase a esta noção na secção seguinte.

Definição 4.3
A relação de igualdade entre dois termos λ, designada por convertibilidade, é

definida indutivamente por

$$\beta\,\frac{}{(\lambda x.M)N = M[x := N]} \qquad \frac{}{M = M} \qquad \frac{M = N}{N = M}$$

$$\frac{M = N \quad N = L}{M = L} \qquad \frac{M = N}{MZ = NZ} \qquad \frac{M = N}{ZM = ZN}$$

$$\xi\,\frac{M = N}{\lambda x.M = \lambda x.N}$$

A regra (ξ) é, por vezes, chamada regra de extensão fraca e estabelece a igualdade entre funções. A regra (β) corresponde a igualdade entre a aplicação de uma função a um parâmetro e o seu resultado (por exemplo, se aceitamos por enquanto algumas "facilidades sintácticas", entre o termo $(\lambda x.x + 1)4$ e o termo 5).

Para designar os teoremas produzidos por esta teoria, utilizaremos a notação $\lambda \vdash M = N$ que designa que os termos M e N são convertíveis (iguais) na teoria λ ou seja que $M = N$ é um teorema.

Teorema 4.1 (Teorema do Ponto Fixo)

$$\forall F \in \Lambda, \exists X \in \Lambda.\ FX = X$$

ou seja, qualquer termo possuí um ponto fixo.

Demonstração: Dado F, basta considerar o termo $W \triangleq \lambda x.F(xx)$. Se escolhermos o termo X como sendo WW então vemos que

$$X \equiv WW \equiv (\lambda x.F(xx))W = F(WW) \equiv FX$$

$\diamond$

X é chamado *ponto fixo* de F.

4.5. Substituições

4.5.1. Captura de Variáveis

A operação de substituição notada $M[x := N]$ e utilizada da regra β da teoria necessita dum cuidado particular que detalharemos nesta secção.

O perigo de uma operação de substituição sem cuidados é a transformação abusiva de termos. Por exemplo

$$(\lambda xy.yx)y \neq \lambda y.yy$$

Se tomarmos um exemplo fora enquadramento do cálculo λ, dum programa C por exemplo

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

Se quisermos trocar o identificador x por y, não iremos de certeza transformar no programa anterior em

```
int y;
...
```

```
int f(int y) {return y+y;}
...
```

mas em

```
int y;
...
int f(int z) {return y+z;}
...
```

O fenómeno indesejável que consiste em alterar o estatuto das variáveis que ocorrem numa expressão (passar de livre a ligadas ou o oposto) chama-se *captura de variável.*

Para evitar este problema aquando da substituição é preciso impor certas restrições. Existem várias abordagens para definir um quadro adequado às substituições. Vamos aqui abordar três delas.

A abordagem clássica Trata-se da abordagem original de Church que define a substituição como a operação seguinte:

Seja t e N dois termos λ, x uma variável

$$t[x := N] \triangleq \begin{cases} N & se\ t = x \\ y & se\ t = y \ \wedge \ x \neq y \\ \lambda x.M & se\ t = \lambda x.M \\ \lambda y.(M[x := N]) & se\ t = \lambda y.M \ \wedge \ x \notin FV(M) \ \vee \ y \notin FV(N) \\ \lambda z.((M[y := z])[x := N]) & se\ t = \lambda y.M \ \wedge \ x \in FV(M) \ \wedge \ y \in FV(N) \\ & \ \wedge \ z\ variável\ fresca \\ (M_1[x := N]M_2[x := N]) & se\ t = M_1M_2 \end{cases}$$

Uma variável fresca é uma variável nova, isto é uma variável que não ocorre no termo considerado (não havendo assim riscos que haja captura de variáveis).

As regras não naturais são as regras 3, 4 e 5 que vamos descrever com mais cuidado:

- a regra 3 se aplica quando a variável para ser substituída está ligada ao nível mais exterior do termo considerado (pelo λ). Neste caso nenhuma ocorrências de x está livre. Logo a substituição não terá nenhum efeito.

- regra 4 se aplica quando não há risco de captura de variável. Ou x não ocorre livre em M ou y não ocorre livre em N. Em ambos os casos não existe o risco de captura de variável. Logo a substituição pode ser efectuada em M sem problemas.

- regra 5 se aplica quando há riscos de o fenómeno de captura de variável ocorrer. Isto é, quando x é livre em M e y em N. Neste caso é necessário realizar mudança de nome de variável para se precaver. Para esse efeito escolhe-se uma variável que não ocorra nos termos considerados, uma variável *fresca*, que substituirá y.

A regra 5 funciona se aceitamos que o nome dos parâmetros de uma função são poucos importantes. Esta hipótese é realista, principalmente se pensarmos em termos de programação. Podemos, para ficar convencidos, voltar aos exemplos C mais acima. A função `int f(int y) {return x+y;}` é igual a `int f(int z) {return x+z;}`

A formulação original da teoria da convertibilidade λ considerava o axioma suplementar seguinte:

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

A notação de De Bruijn Outra forma de resolver o problema é de modificar o cálculo λ de forma a que as variáveis desaparecem. Esta escolha, que vamos detalhar já a seguir, tem uma particular importância: é a abordagem que é computacionalmente mais razoável ao ponto que é a opção preferida quando se implementa o cálculo λ.

O conceito subjacente consiste em referenciar as variáveis por um número que representa a distancia entre a variável e o λ que os liga. Por exemplo $\lambda.\lambda.2$ representa $\lambda xy.x$.

Formalmente redefinimos o conjunto Λ da forma seguinte:

$$\mathrm{Var}\frac{n \in \mathbb{N}^*}{n \in \Lambda}$$

$$\mathrm{App}\frac{M \in \Lambda \qquad N \in \Lambda}{(M\ N) \in \Lambda} \quad \mathrm{Abs}\frac{M \in \Lambda}{(\lambda\ M) \in \Lambda}$$

Neste caso a regra β é redefinido como $\beta\frac{}{(\lambda\ P)Q = P[1 := Q]}$

onde

$$t[m := N] \triangleq \begin{cases} n & se\ t = n\ \wedge\ n < m \\ n-1 & se\ t = n\ \wedge\ n > m \\ rename_{n,1}(N) & se\ t = n\ \wedge\ n = m \\ (M_1[m := N]M_2[m := N]) & se\ t = M_1M_2 \\ y & se\ t = y\ \wedge\ x \neq y \\ \lambda(M[m+1 := N]) & se\ t = \lambda\ M \end{cases}$$

e

$$rename_{m,i}(M) \triangleq \begin{cases} j & se\ M = j \ \wedge \ j < i \\ j + m - 1 & se\ M = j \ \wedge \ j \geq i \\ rename_{m,i}(N_1)\ rename_{m,i}(N_2) & se\ M = (N_1\ N_2) \\ \lambda\ rename_{m,i+1}(N) & se\ M = (\lambda\ N) \end{cases}$$

Uma propriedade interessante desta notação é o facto de termos α equivalentes serem representado pelo mesmo termo em notação De Bruijn.

A convenção de variáveis A última abordagem consiste na hipótese de que quando efectuamos uma substituição estamos sistematicamente na situação em que não ocorre problema nenhum.

Para tal introduzimos dois conceitos:

Definição 4.4 (Mudança de variáveis ligadas) *Seja $C[]$ um contexto com um buraco. Diz-se que M' é produzido a partir de M por mudança de variáveis ligadas se $M \triangleq C[\lambda x.N]$ e $M' \triangleq C[\lambda y.(N[x := y])]$ tal que y não ocorre (nem de forma livre nem de forma ligada) em N.*

Definição 4.5 (α-congruência) *Um termo M é α-congruente a N (notação $M \equiv_\alpha N$), se N resulta de M a partir de uma sequência de mudanças de variáveis ligadas.*

Por exemplo $\lambda x.xy \equiv_\alpha \lambda z.zy$ mas $\lambda x.xy \not\equiv_\alpha \lambda y.yy$.

Tendo definido a congruência $\equiv_\alpha$, podemos então considerar um termo t como o representante de todos os termos de $[t]_{\equiv_\alpha}$

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades
Página Pessoal
Página de Rosto
Página 18 de 100
Retroceder
Ecrã Todo
Fechar
Sair

Assim tendo em conta um problema de substituição $M[x := N]$, podemos considerar um termo de $[M]_{\equiv_\alpha}$ e um termo de $[N]_{\equiv_\alpha}$ tais que todas as variáveis ligadas sejam distintas. Basicamente isto corresponde em adoptar a convenção seguinte:

Definição 4.6 *Se $M_1, \ldots, M_N$ ocorrem num determinado contexto então nesses termos todas as variáveis ligadas são escolhidas de forma a serem todas diferentes das variáveis livres.*

Desta hipótese inicial podemos então redefinir a substituição da seguinte forma:

$$t[x := N] \triangleq \begin{cases} N & \mathit{se}\ t = x \\ y & \mathit{se}\ t = y\ \wedge\ y \neq x \\ (M_1[x := N]M_2[x := N]) & \mathit{se}\ t = M_1M_2 \\ \lambda y.(M[x := N]) & \mathit{se}\ t = \lambda y.M \end{cases}$$

Embora a notação de De Bruijn seja a mais importante em termo de aplicação (e a notação realmente utilizada nos sistemas implementados) utilizaremos esta abordagem por ser a mais intuitiva e simples.

4.5.2. O Lema da Substituição

O resultado interessante seguinte permite-nos reorganizar consoante as necessidades as varias substituições pendentes.

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

Lema 4.1 (Lema da Substituição) *Sejam M, N e L três termos. Se x e y são variáveis distintas e $x \notin FV(L)$, então*

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

A substituição usufrui de outras propriedades interessantes:

Lema 4.2

- $M = M' \;\rightarrow\; M[x := N] = M'[x := N]$
- $N = N' \;\rightarrow\; M[x := N] = M[x := N']$
- $M = M', N = N' \;\rightarrow\; M[x := N] = M'[x := N']$

Lema 4.3 *(Transparência referencial) Seja $C[]$ um contexto, então*

$$N = N' \;\rightarrow\; C[N] = C[N']$$

4.6. Extensionalidade

A relação de convertibilidade $(=)$ é uma relação de igualdade por intensão. Isto é, dois termos são iguais se codificam o mesmo algoritmo. O problema é que esta relação não identifica termos que calculam a mesma coisa mas de forma diferente (que não utilizam o mesmo algoritmo para o mesmo fim).

Assim a igualdade seguinte $\lambda x.(M\ x) = M$ não é um teorema da teoria λ apesar de intuitivamente a aceitarmos como válida.

Existem duas abordagens "históricas" que extendem a teoria λ com mecanismos de igualdade por extensão.

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

A primeira propõe a junção da nova regra

$$\text{ext}\,\frac{Mx = Nx \qquad x \notin FV(M) \cup FV(N)}{M = N}$$

Tal teoria é designada por $\lambda + ext$.

A segunda abordagem, proposta pelo próprio Church, contempla o axioma seguinte:

$$\eta\,\frac{x \notin FV(N)}{\lambda x.Mx = M}$$

Tal teoria é designada por $\lambda\eta$.

Lema 4.4 *$\lambda + ext$ e $\lambda\eta$ são equivalentes.*

Demonstração:
Basta demonstrar as seguintes duas propriedades relativamente simples:

- $\lambda + ext \vdash \lambda x.Mx = M$ desde que $x \notin FV(N)$
- $\lambda\eta \vdash ext$

$\diamond$

4.7. Consistência e Completude

Para que uma teoria seja útil, é preciso que tenha teoremas e que nem todas as fórmulas fechadas (fórmulas sem variáveis livres) sejam teoremas. A primeira propriedade é satisfeita se, por exemplo, se conseguir exibir um axioma. A segunda propriedade, mais subtil e complicada, é designada por *consistência* ou ainda por *correcção*. Informalmente, um sistema consistente é um sistema que só gera frases válidas.

Todas as teorias λ que até aqui apresentamos são consistentes, mas veremos que é muito fácil, para uma teoria, deixar de o ser.

Definição 4.7 (Equação) *Uma equação é uma fórmula que segue o segue o padrão seguinte $M = N$ onde ambos M e N são termos de Λ.*

Definição 4.8 (Equação Fechada) *Uma equação é fechada se $M \in \Lambda^0$ e $N \in \Lambda^0$*

Definição 4.9 (Consistência) *Se uma teoria T é uma teoria onde as fórmulas são equações então T é* consistente, *notado $Con(T)$, se não prove qualquer equação fechada.*

Se T é um conjunto de equações então $\lambda + T$ é a teoria obtida ao acrescentar todas as equações de T como axiomas a λ. Neste caso T é consistente, notado $Con(T)$ se $Con(\lambda + T)$.

Definição 4.10 (Incompatibilidade) *Sejam $M, N \in \Lambda$, M e N são incompatíveis, notado $M \sharp N$, se $\neg Con(\lambda + (M = N))$*

A noção de incompatibilidade leva a definição de uma muito útil técnica de demonstração da inconsistência:

- utilizar a extensionalidade fraca for fim a gerar uma equação fechada;
- escolher arbitráriamente termos que serão aplicados aos termos fechados obtidos de cada lado da igualdade por fim a eliminar todos os λ exteriores aos dois termos da equação;
- utilizar a regra β quanto possível;
- instanciar os termos arbitrários com termos fechados (constantes) e tentar derivar a equação

$$\textit{termo arbitrário} = \textit{termo constante}$$

Constantes importantes são os combinadores seguintes:

$$S \triangleq \lambda xyz.xz(yz)$$

$$K \triangleq \lambda xy.x$$

$$I \triangleq \lambda x.x$$

Um exemplo é a demonstração de que $xx \sharp xy$

Demonstração:

$$
\begin{array}{lrcll}
 & xx & = & xy & \\
\rightarrow & \lambda xy.xx & = & \lambda xy.xy & \text{por duas aplicações da extensionalidade fraca} \\
\rightarrow & (\lambda xy.xx)MN & = & (\lambda xy.xy)MN & M \text{ e } N \text{ termos quaisquer} \\
\rightarrow & MM & = & MN & \text{por duas aplicações da regra } \beta \\
\rightarrow & I & = & N & \text{se escolhermos} M \equiv I
\end{array}
$$

$\diamond$

Outra propriedade importante quando se fala de teorias é a completude. Esta última garante que a teoria em questão é capaz de produzir todas as frases válidas. No contexto da teoria λ, isto significa que qualquer par de termos convertíveis pode ser verificado pela teoria. Dito de outra forma, uma teoria é completa se não falhar. Vemos assim que a consistência e a completude são propriedades importantes e complementares. A primeira estabelece que todos os teoremas da teoria considerada são válidos e a segunda estabelece que nada de válido "escapa" a teoria.

Para expressar formalmente o que entendemos por completude no enquadramento do cálculo λ, introduzimos as noções centrais seguintes:

Definição 4.11 (Formas Normais) *Seja $M \in \Lambda$, M é uma β-*forma normal, *notado $\beta - nf$ ou nf, se M não tem sub-termos da forma $(\lambda x.R)S$. O termo M tem um uma forma normal se existir um termo N em forma normal tal que $M = N$.*

*De forma similar, M é uma $\beta\eta$-*forma normal *se não conter nenhum sub-termo da forma $(\lambda x.R)S$ ou da forma $(\lambda x.Rx)$ em que $x \notin FV(R)$*

Proposição 4.1 *1. $M \in \Lambda$ tem uma $\beta\eta$-forma normal se e só se M tem uma β-forma normal;*

2. se $M, N \in \Lambda$ são β-formas normais distintas então $M = N$ não é um teorema de λ (nem de $\lambda\eta$);

3. se M e N são $\beta\eta$-formas normais distintas então $M \sharp N$.

Proposição 4.2 (Completude) *Sejam M e N dois termos com forma normal. Então ou $\lambda\eta \vdash M = N$ ou $\lambda\eta + (M = N)$ é inconsistente.*

5. Reduções e Igualdade

Até agora preocupamo-nos em estabelecer a relação de convertibilidade entre termos. Se olharmos bem a regra β:

$$\beta\,\frac{}{(\lambda x.M)N = M[x := N]}$$

reparamos que ela permite igualar o termo $(\lambda x.M)N$ com o que intuitivamente entendemos ser o resultado da aplicação da função $(\lambda x.M)$ ao parâmetro N. Destacamos também termos particulares que são as formas normais. Esses termos tem a particularidade de não conter nenhum sub-termo representando uma função por aplicar. De certa forma a ideia que temos dum cálculo, que não é explícita na teoria λ, é exactamente "reduzir" um termo aplicando todas funções por aplicar até obter um valor que veremos ser uma forma normal. A teoria λ sublinha mais a relação entre termos que representam o mesmo valor do que a transformação permitindo obter este valor.

Vamos assim ter nesta secção uma visão computacional da teoria λ que justifica o nome de cálculo λ.

A ideia base para transformar teoria λ num cálculo é relativamente simples: seguir a intuição de que $M[x := N]$ é o resultado de $(\lambda x.M)N$. Dito de outra forma, para termos um cálculo basta orientar a regra β. Obtemos assim a *regra de redução* β:

$$(\lambda x.M)N \rhd_\beta M[x := N]$$

O cálculo do valor dum termo t é então a iteração do processos codificado pela regra $\rhd_\beta$ a todos os sub-termos até que não seja mais possível aplicar

$\triangleright_\beta$. O termo obtido, se o processo terminar, é necessariamente uma forma β-normal.

Iremos a seguir apresentar toda a maquinaria envolvida a volta deste processo de redução. Apresentaremos também alguns resultados.

No entanto, várias questões surgem desde já da descrição informal dada deste processo de cálculo.

Terminação: quando pensamos na redução no cálculo λ como um cálculo, esperamos que este processo acaba. Terá a redução no cálculo λ esta propriedade?

Confluência: dado um termo t de Λ, existem várias formas de o reduzir (porque, por exemplo, vários subtermos de t são da forma $(\lambda x.M)N$) logo alvo de redução β), logo temos de escolher, e a escolha não é única, com que sub-termo começar). Será que todas as escolhas levam ao mesmo resultado?

Estratégia de redução óptima: haverá uma escolha (ou estratégia de redução) mais adequada do que as outras?

As respostas a essas questões serão, também, o alvo das secções seguintes.

6. Propriedades

6.1. Relações de Redução

Vamos começar por definir formalmente o que entendemos por relação de redução e enunciar algumas definições e propriedades das relações de redução. Aplicaremo-las então ao caso da redução β que introduzimos no início da secção.

Definição 6.1 *Seja R uma relação binária sobre Λ ($R \subseteq \Lambda^2$). R é* compatível *se para todos termos t e t' e para todo o contexto com um buraco $C[]$, $(t\ R\ t') \rightarrow (C[t]\ R\ C[t'])$.*

Definição 6.2 *Seja R uma relação binária sobre Λ ($R \subseteq \Lambda^2$), R é uma* relação de igualdade (congruência) *se for uma relação de equivalência compatível.*

Definição 6.3 *Seja R uma relação binária sobre Λ ($R \subseteq \Lambda^2$), R é uma* relação de redução *se for uma relação reflexiva, transitiva e compatível.*

Definição 6.4 *Uma* noção de redução *R é, simplesmente, uma relação binária sobre Λ. Utilizaremos preferencialmente a notação $\triangleright_R$ para representa-las.*

A definição de noção de redução só faz sentido se realizarmos que a relação $\triangleright_\beta$ definida, relembremos, por $(\lambda x.M)N \triangleright_\beta M[x := N]$ é uma noção de redução. O que pretendemos é definir uma relação de redução com base nela.

Definição 6.5 (R-redução num passo) *Seja $\triangleright_R$ uma noção de redução. Uma R-redução num passo, notado $\rightarrow_R$, é o fecho compatível de $\triangleright_R$. Explicitamente o fecho compatível é obtido extendendo $\triangleright_R$ da seguinte forma:*

$$\frac{M \ \triangleright_R \ N}{M \ \rightarrow_R \ N}$$

$$\frac{M \ \rightarrow_R \ N}{MZ \ \rightarrow_R \ NZ}$$

$$\frac{M \ \rightarrow_R \ N}{ZM \ \rightarrow_R \ ZN}$$

$$\frac{M \ \rightarrow_R \ N}{\lambda x.M \ \rightarrow_R \ \lambda x.N}$$

A relação $\rightarrow_\beta$ que reduz um dos subtermos dum termo t que tiver a forma $(\lambda x.M)N$ é uma redução num passo. Por outras palavras, $\rightarrow_\beta$ é a redução num passo induzida por $\triangleright_\beta$. Se $M \rightarrow_\beta N$ diz-se que M se (β-)reduz num passo em N.

Definição 6.6 (R-redução) *A relação de redução induzida por $\triangleright_R$, designada por $\twoheadrightarrow_R$, é o fecho transitivo e reflexivo de $\rightarrow_R$. Explicitamente temos:*

$$\frac{M \;\to_R\; N}{M \;\twoheadrightarrow_R\; N}$$

$$\frac{}{M \;\twoheadrightarrow_R\; N}$$

$$\frac{M \;\twoheadrightarrow_R\; N \qquad N \twoheadrightarrow_R\; L}{M \;\twoheadrightarrow_R\; L}$$

Se $M \twoheadrightarrow_\beta N$ diz-se que M se (β-)reduz em N.

Definição 6.7 (R-conversão) *A relação de conversão induzida por $\rhd_R$, designada por $=_R$, é a relação de equivalência induzia por $\twoheadrightarrow_R$. Isto é, $=_R$ é fecho simétrico de $\twoheadrightarrow_R$. Explicitamente temos:*

$$\frac{M \;\twoheadrightarrow_R\; N}{M \;=_R\; N}$$

$$\frac{M \;=_R\; N}{N \;=_R\; M}$$

$$\frac{M \;=_R\; N \qquad N =_R\; L}{M \;=_R\; L}$$

Se $M =_\beta N$ diz-se que M é (β-)convertível com N.

Lema 6.1 *Seja $\triangleright_R$ uma noção de redução. $\rightarrow_R$, $\twoheadrightarrow_R$ e $=_R$ são relações compatíveis.*

Lema 6.2 *Para todos os $M, N, N' \in \Lambda$ e para toda a noção de redução Se $N \twoheadrightarrow_R N'$ então $M[x := N] \twoheadrightarrow_R M[x := N']$*

6.2. Confluência

A questão que vamos abordar nesta secção é a da unicidade do valor calculado. Definiremos por forma normal o que entendemos por resultado dum cálculo e veremos que o cálculo λ tem uma propriedade designada por Churh-Rosser que garante que se existe uma forma normal, então essa é única.

Definição 6.8 *Sejam M e N termos e $\triangleright_R$ uma noção de redução. Se $Mtriangleright_R N$, diz-se que M é um R-redex (abreviatura de* reducible expression*) e N um R-contractum.*

Se um termo t está em R-forma normal (ou em R-nf) se nenhum dos seus subtermos é um R-redex. Um termo u é uma R-forma normal de t se u é uma R-forma e $t =_R u$.

Lema 6.3 *Seja $\triangleright_R$ uma noção de redução, seja M uma R-forma normal então*

- *Não existe nenhum N tal que $M \rightarrow_R N$*
- *Se $M \twoheadrightarrow_R N$ então $M \equiv N$.*

Definição 6.9 (A propriedade do diamante $\diamond$)
Seja $\blacktriangleright$ uma relação binária sobre Λ. Diz-se de $\blacktriangleright$ que verifica a propriedade $\diamond$, notação $\blacktriangleright \models \diamond$, se

$$\forall M, M_1, M_2 \in \Lambda.\ (M \blacktriangleright M_1 \wedge M \blacktriangleright M_2) \longrightarrow \exists M_3 \in \Lambda.\ (M_1 \blacktriangleright M_3 \wedge M_2 \blacktriangleright M_3)$$

Definição 6.10 (Relação Church-Rosser)
Uma noção de redução $\triangleright_R$ é dita Church-Rosser (ou CR) se $\twoheadrightarrow_R \models \diamond$.

Teorema 6.1 (Teorema de Church-Rosser)
Seja $\triangleright_R$ uma noção de redução verificando CR. Então

$$\forall M, N \in \Lambda M =_R N \ \longrightarrow\ \exists Z.(M \twoheadrightarrow_R Z \wedge N \twoheadrightarrow_R Z)$$

Corolário 6.1 *Seja $\triangleright_R$ uma noção de redução verificando CR. Então*

- *Sejam $N; M \in \Lambda$, se N é uma R-nf de M então $M \twoheadrightarrow_R N$;*
- *um termo tem no máximo uma R-forma normal*

Desses teoremas podemos deduzir que se $\triangleright_\beta$ for CR então a resposta a segunda questão (terá um cálculo um resultado único) é positiva.

A verificação de tal resultado exige toda uma tecnologia matemática que não exploraremos aqui. Referiremos somente o enunciado que nos interessa:

Teorema 6.2 (Teorema de Confluência)

- *$\triangleright_\beta$ é Church-Rosser*

- $\triangleright_{\beta\eta}$ *é Church-Rosser*

Aqui $\triangleright_{\beta\eta}$ representa a noção de redução induzida pela regra β e pela regra η.

A custa do resultado da confluência podemos enunciar:

Teorema 6.3

$$M =_\beta N \longleftrightarrow \lambda \vdash M = N$$

6.3. Terminação

Aqui olhamos para a primeira questão: será que todos os processos de cálculo terminam. Já sabemos que se terminar então o resultado é único. Vamos agora explorar a pertinência da hipótese enunciada na frase anterior.

Definição 6.11 (normalização forte)
Sejam $\triangleright_R$ uma noção de redução e M um termo:

- M R-normaliza fortemente *(notação R-$SN(M)$, e SN como abreviatura de* Strongly Normalize*) se não existir R-reduções infinitas a partir de M;*
- *M é R-infinito se não R-normaliza fortemente;*
- *$\triangleright_R$ é fortemente normalizável (notação $\triangleright_R$ é SN) se*

$$\forall M \in \Lambda.\ R\text{-}SN(M)$$

Teorema 6.4 (Teorema da (não) normalização forte)
$\triangleright_\beta$ *não é fortemente normalizável.*

Demonstração:
Basta considerar a β redução do termo $(\lambda x.xx)(\lambda x.xx)$.
$\diamond$

O ponto anterior é particularmente forte em termos de consequências. Há cálculos que não acabam. Não podemos descartar a hipótese da terminação.

Certas extensões do cálculo λ adquiram no entanto esta propriedade ao aceitar perder um pouco da expressividade (proibir as construções que levam a cálculo infinitos). Uma dessas extensões, o *cálculo λ tipificado*, e mais particularmente a *teoria dos tipos*, é essencial as ligações que o cálculo λ tem com a matemática e com a ciência da computação.

6.4. Normalização e Estratégias de Redução

Já sabemos que um termo pode conter vários redex, logo existe várias escolhas de reduções em um passo (basicamente uma por cada redex). Sabemos também que essas escolhas podem conduzir a um processo de redução "infinita". A situação é de facto problemática. O termo seguinte:

$$(\lambda xy.x)(\lambda z.z)((\lambda t.tt)(\lambda u.uu))$$

pode ser reduzido em um passo em $(\lambda z.z)$ se escolhermos o redex mais a esquerda ou levar a uma redução infinita se escolhermos o redex mais a direita.

A questão aqui é então de saber se existe uma escolha, ou melhor *estratégia*, pertinente e determinista dos redex por reduzir.

Aviso Prévio
Bibliografia
Introdução
Notações e Teoria . . .
Reduções e Igualdade
Propriedades

Página Pessoal
Página de Rosto
Página 34 de 100
Retroceder
Ecrã Todo
Fechar
Sair

6.4.1. Resíduos

Vamos utilizar nas definições seguintes a noção de *resíduo* que não definiremos de forma formal (consultar a bibliografia para tal). Um resíduo é o descendente de um redex. Esta noção é utilizada para "tracejar" as reduções efectuadas e assim obter de forma gráfica um histórico (uma evolução) da vida de um redex.

Por exemplo, dado o redex inicialmente sublinhado, os resíduos são os subtermos sublinhados no termo sujeito à redução β seguinte:

$$(\lambda xy.\underline{(\lambda zw.xz)y})MN \to_\beta (\lambda xy.\underline{(\lambda zw.Mz)y})N \to_\beta \underline{(\lambda zw.Mz)N} \to_\beta \lambda w.MN$$

Repare que no último termo o redex de que seguimos o rasto foi completamente reduzido por isso deixou de existir, logo o resíduo também.

6.4.2. Formas Normais de Cabeça

Definição 6.12 (Formas Normais de Cabeça e Redex de Cabeça) *$M \in \Lambda$ é uma* forma normal de cabeça *(hnf, de Head Normal Form) se M tem a forma seguinte:*

$$\lambda x_1 \ldots x_n.xM_1 \ldots M_m \qquad x \in \mathcal{V} \land n, m \geq 0$$

Neste caso x é designada por a variável de cabeça.

Se $M \equiv \lambda x_1 \ldots x_n.(\lambda x.M_0)M_1 \ldots M_m$ (com $n \geq 0$ e $m \geq 1$) então $(\lambda x.M_0)M_1$ é designado por redex de cabeça *de M.*

Num termo que não está em forma normal de cabeça designaremos por *redex mais a esquerda* o redex “geograficamente” mais a esquerda de todos: é o redex tal que a sua primeira ocorrência do símbolo λ está a esquerda de qualquer outra ocorrência de λ de qualquer outro redex. Consultar a bibliografia para uma definição mais formal.

Teorema 6.5 (Teorema de estandardização) *Seja t um termo. Se t tem uma forma normal u então a estratégia que consiste em escolher sempre o redex mais a esquerda se reduz em u.*

Designaremos esta estratégia pela sua terminologia inglesa, isto é *call by value* ou *normal order reduction*

Existem várias estratégias de grande interesse (ver em particular [6]), mas o teorema da estandardização nos garante que o call by value encontra a forma normal se essa existir.

Estas considerações concluem as problemáticas discutidas.

References

[1] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.

[2] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–310. Oxford University Press, New York, 1992.

[3] G. Dowek. Théories des types. DEA Programmation, Paris VII, 2002. `http://www.lix.polytechnique.fr/~dowek/Cours/theories_des_types.ps.gz`

[4] J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988.

[5] Chris Hankin. *Lambda Calculi: A Guide for Computer Scientists*, volume 3 of *Graduate Texts in Computer Science*. Clarendon Press, Oxford, 1994.

[6] G. Huet. Constructive computation theory. Course Notes, DEA Informatique Université Bordeaux I, October 1992. `http://pauillac.inria.fr/~huet/CCT/`

[7] J. L. Krivine. *Lambda-Calculus, Types and Models*. Ellis Horwood, 1993.