Módulo II: Implementação do λ-calculus

MCC - 2ºano

José Bernardo Barros (jbb@di.uminho.pt)
José Carlos Bacelar Almeida (bacelar@di.uminho.pt)

Departamento de Informática
Universidade do Minho

Motivação

Implementar directamente a operação de susbtituição é complicado e resulta num procedimento particularmente pouco eficiente. Basta observar a clausula da abstração:

$$y \in FV(E) \quad \Longrightarrow \quad (\lambda y.E)[x \leftarrow N] = \lambda z.((E[y/z])[x \leftarrow N])$$

,sendo z uma **variável nova**

Aqui, só para testar as condições de aplicabilidade da regra, necessitamos de realizar uma travessia para cada um dos termos envolvidos ($y \in FV(E)$ e encontrar uma variável z nova).

Já referimos atrás que este problema se deve ao facto de a sintaxe não nos permitir trabalhar directamente ao nível que desejariamos: o das *classes α-equivalentes*.

Vamos agora apresentar uma formulação da sintaxe dos termos-λ que pode ser entendida como uma sintaxe concreta para as referidas classes de equivalência: os *Índices de Bruijn*.

A ideia básica consiste em representar uma variável ligada pelo número de λs que necessita "saltar" até atingir o que a vincula. As variáveis livres são as que esgotam todos os λs disponíveis (sendo o índice da variável o valor original menos o número de λs saltados).

Índices *de Bruijn*

Na notação *de Bruijn* as variáveis são representadas por inteiros e as abstrações dispensam o identificador da variável abstraida:

$$\mathcal{T} ::= n|(\mathcal{T}\ \mathcal{T})|\lambda.\mathcal{T}$$

Uma variável $(n-1)$ "liga-se" á n-ésima abstração que a precede (na árvore sintática). Se não dispuser dessa quantidade de abstrações é uma variável livre.

Exemplos:

$$\begin{aligned} \lambda x.x &\rightsquigarrow \lambda 0 \\ \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) &\rightsquigarrow \lambda(\lambda 1(00))(\lambda 1(00)) \\ \lambda z.zxy &\rightsquigarrow \lambda 012 \\ \lambda z.zx(\lambda y.zxy) &\rightsquigarrow \lambda 01(\lambda 120) \end{aligned}$$

no último termo fica patente que, nesta representação, as mesmas variáveis podem ficar representadas com índices diferentes (1ªs ocorrências de z e x surgem com números 0 e 1 respectivamente, enquanto que as 2ªs surgem com 1 e 2) (note que, nesse termo, x é livre e z ligada).

A adequação desta representação fica patente considerando as funções[a]

$$\begin{aligned} \lceil - \rceil &: \Lambda \to \mathcal{T} \\ \overline{-} &: \mathcal{T} \to \Lambda \end{aligned}$$

e notando que, para M, N termos fechados,

- $M \stackrel{\alpha}{=} N \implies \lceil M \rceil \equiv \lceil N \rceil$
- $M \stackrel{\alpha}{=} \overline{\lceil M \rceil}$

[a] Serão definidas como um exercício nas aulas práticas.

Substituição com Índices *de Bruijn*

Considere-se o βredex,

$$(\lambda a.(\lambda b.\underline{a}\ b)\ c\ \underline{a})\ (\lambda y.x\ y)$$

um passo de redução conduz-nos a $((\lambda b.a\ b)\ c\ a)[a \leftarrow \lambda y.x\ y]$, ou seja,

$$(\lambda b.\underline{(\lambda y.x\ y)}\ b)\ c\ \underline{(\lambda y.x\ y)}$$

Vejamos agora utilizando índices *de Bruijn*. O redex fica,

$$(\lambda(\lambda\underline{1}\ 0)\ 1\ \underline{0})\ (\lambda 20)$$

e o termo resultante da substituição:

$$(\lambda.\underline{(\lambda.3\ 0)}\ 0)\ 0\ \underline{(\lambda.2\ 0)}$$

Podemos observar os seguintes factos:

- as variáveis livres que não são afectadas pela substituição (c do termo apresentado) veem o seu índice decrementado (reflete o λ que desaparece);
- as variáveis livres do termo que substitui a variável (x)são ajustados conforme a posição que ocupa no termo (necessitam de "saltar" mais ou menos λs);

(cont.)

Tendo notado estes aspectos podemos então definir a operação de substituição. Para o efeito utilizamos uma função auxiliar (`promove`) que é responsável por ajustar o índice das variáveis livres de um termo (e.g. `promove` $2\ (\lambda.1\ 0) = \lambda.3\ 0$).

```
data Term = Ref Int | Abs Term | App Term Term
promove :: Int -> Term -> Term
promove n = promoveAux 0
  where promoveAux k (Ref i) = if i<k
                               then Ref i
                               else Ref (n+i)
        promoveAux k (Abs m) = Abs (promoveAux (k+1) m)
        promoveAux k (App m n) = App (promoveAux k m)
                                     (promoveAux k n)
```

A substituição pode agora ser definida como

```
subst :: Term -> Term -> Term
subst t = substAux 0
  where substAux n (Ref k) = if k==n
                             then promove n t
                             else if k<n
                                  then Ref k
                                  else Ref (k-1)
        substAux n (Abs t') = Abs (substaux (n+1) t')
        substAux n (App t' t'') = App (substAux n t')
                                      (substAux n t'')
```

Combinadores

Consideremos os combinadores[a]

$$\begin{aligned} S &\doteq \lambda xyz.x\ z\ (y\ z) \\ K &\doteq \lambda xy.x \\ I &\doteq \lambda x.x \end{aligned}$$

podemos considerar um sistema $\mathcal{C}$ onde todos os termos são construidos por aplicação de S, K e I, i.e.

$$\mathcal{C} ::= S \mid K \mid I \mid (\mathcal{C}\ \mathcal{C})$$

Sobre este sistema definimos uma relação de *reescrita*:

$$\begin{aligned} I\ x &\longrightarrow x \\ K\ x\ y &\longrightarrow x \\ S\ x\ y\ z &\longrightarrow x\ z\ (y\ z) \\ x \longrightarrow y &\Longrightarrow (x\ w) \longrightarrow (y\ w) \\ x \longrightarrow y &\Longrightarrow (w\ x) \longrightarrow (w\ x) \end{aligned}$$

(note que aqui as variáveis são do *meta-nível*, i.e. são utilizadas para exprimir as regras de reescrita — não pertencem à sintaxe do sistema). O sistema de reescrita $\mathcal{C}$ assim definido é confluente (i.e. exibe a propriedade de Church-Rosser).

Note-se que existe uma diferença fundamental entre a relação de reescrita apresentada e a redução β do λ-calculus: ***aqui não está envolvida nenhuma substituição*** *(de resto, no sistema $\mathcal{C}$ nem sequer dispomos de variáveis). Este facto faz com que seja muito mais facil (e eficiente) a sua implementação.*

[a] Dado que $I \overset{\beta}{=} S\ K\ K$, poderiamos restringir o sistema aos combinadores S, K.

Combinadores (cont.)

Tendo definido o sistema $\mathcal{C}$ não é óbvio como o utilizar para reduzir λ-termos. Aparentemente só dispomos capacidade de representar termos muito particulares: os que resultam da aplicação dos combinadores S e K (agora vistos como λ-termos).

Existe um importante resultado que nos diz que qualquer termo fechado pode ser expresso por aplicações exclusivas dos combinadores S e K. O algoritmo que nos permite transformar um termo fechado é dado por

$$\begin{array}{rcl}
\lceil \lambda x.x \rceil & \rightsquigarrow & I \\
\lceil \lambda x.M \rceil & \rightsquigarrow & K\ M \qquad \text{, se } M \text{ é atómico (const./var.) diferente de } x \\
\lceil \lambda x.MN \rceil & \rightsquigarrow & S\ \lceil \lambda x.M \rceil\ \lceil \lambda x.N \rceil \\
\lceil \lambda xy.M \rceil & \rightsquigarrow & \lceil \lambda x \lceil \lambda y.M \rceil \rceil \\
\lceil M\ N \rceil & \rightsquigarrow & \lceil M \rceil\ \lceil N \rceil
\end{array}$$

No sentido inverso podemos considerar correspondência óbvia que associa aos combinadores S, K, I a sua definição como λ-termos.

$$\begin{array}{rcl}
\overline{\ } & : & \mathcal{C} \to \Lambda \\
\overline{K} & = & \lambda xy.x \\
\overline{S} & = & \lambda xyz.x\ z\ (y\ z) \\
\overline{(x\ y)} & = & (\overline{x}\ \overline{y})
\end{array}$$

Combinadores (cont.)

A adequação da transformação apresentada fica demonstrada pelo seguinte resultado (prova por indução na estrutura dos termos)

$$\overline{\lceil M \rceil} \stackrel{\beta}{=} M$$

EXEMPLO: Consideremos o λ-termo $\lambda xy.y$. Por aplicação do algoritmo obtemos $K\ I$ que, visto novamente como um λ-termo se reduz em $\lambda xy.y$.

Outras propriedades que se demonstram facilmente:

- Se $x \rightarrow^\star y$ então $\overline{x} \stackrel{\beta}{\longrightarrow} \overline{y}$
- Se $\overline{x}$ é um λ-termo na forma normal então x também está na forma normal.

Mas a propriedade que realmente nos interessa é a implicação no sentido inverso ao apresentado na segunda propriedade (i.e. uma forma normal em $\mathcal{C}$ é transposta numa forma normal em Λ). Aqui deparamo-nos com um resultado negativo que parece comprometer a abordagem sugerida:

contra-exemplo: $K\ I$ está na forma normal em $\mathcal{C}$ mas $(lambdaxy.x)\ (\lambda x.x)$ não é um termo na forma normal.

Mesmo não atingindo a forma normal, a simplificação de λ-termos utilizando a reescrita de combinadores é "computacionalmente" adequada. Note-se que as formas normais que o sistema não atinge resultam sempre de um número insuficiente de argumentos fornecidos aos combinadores S, K e por isso correspondem a formas normais que são forçosamente abstrações. *Na perspectiva de "linguagens de programação" essas reduções em falta acabam por não ser relevantes.*

Combinadores: optimizações adicionais

Tendo observado que os combinadores S, K são suficientes para exprimir todos os λ-termos, nada nos impede de introduzir combinadores adicionais para procurar aumentar a eficiencia do processo. Um primeiro exemplo que introduzimos desde logo foi o combinador I. Outros combinadores normalmente utilizados são:

$$\begin{aligned} B &\doteq \lambda xyz.x\ (y\ z) \\ C &\doteq \lambda xyz.x\ y\ z \end{aligned}$$

e considerarmos as seguintes regras de reescrita adicionais[a]:

$$\begin{aligned} B\ x\ y\ z &\longrightarrow x\ (y\ z) \\ C\ x\ y\ z &\longrightarrow x\ y\ z \end{aligned}$$

$$\begin{aligned} S\ (K\ x)\ I &\longrightarrow x \\ S\ (K\ x)\ (K\ y) &\longrightarrow K\ (x\ y) \\ S\ (K\ x)\ y &\longrightarrow B\ x\ y \\ S\ x\ (K\ y) &\longrightarrow C\ x\ y \end{aligned}$$

Note-se que podemos incluir o último conjunto de regras como uma optimização do processo de tradução (produzindo então um termo composto pelos combinadores S, K, I, B, C), e o primeiro conjunto de regras extender as regras originais no processo de redução dos termos.

[a] A correcção do segundo conjunto de regras faz uso da extensionalidade.

Exemplo da utilização de combinadores

Consideremos o termo:

$$(\lambda xy.x\ (\lambda nsz.s\ (n\ s\ z))\ y)\ ((\lambda nsz.s\ (n\ s\ z))\ (\lambda sz.z))\ (\lambda sz.z)$$

A forma normal deste termo é $\lambda sz.s\ z$ obtida após 10 passos de reduçao-β.

Por aplicaçao do algoritmo obtemos:

```
S (S (K S) (S (S (K S) (S (K K) I)) (S (S (K S) (S (S (K S) (S (K K) (K S))) (S (S (K S) (S (S (K S) (S (K K) (K S)))
.....................................(20 linhas de Ss e Ks...).....................................
(S (S (K S) (S (K K) (K K))) (S (K K) I)))) (S (S (K S) (S (K K) (K K))) (K I))))) (S (K K) (K I))) (K I)) (K I)
```

que se reduz em 765 passos de reescrita em

```
S (S (K S) (S (K K) I)) (S (S (K S) (S (S (K S) (S (K K) (K (K I)))) (S (K K) I))) (K I))
```

Já se realizarmos a optimizaçao obtemos:

```
C (B S (S (B S K) (S (B S (S (B S (K (K S))) (S (B S (S (B S (K (K S))) (C (B S (K (K K))) (K S)))) (S (B S (S (B S
.....................................(12 linhas de Ss,Ks,Is,Bs e Cs...).....................................
(S (B S (S (B S (K (K S))) (S (B S (K (K K))) K))) (C (B S (K (K K))) I)))) (K I)) (K I)) (K I)
}%
```

que com 79 reduçoes obtém I (note que $I \overset{\beta\eta}{=} \lambda sz.s\ z$ — a forma normal).