Módulo IV:

λ-calculus com tipos

MCC - 2ºano

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

Departamento de Informática
Universidade do Minho

Motivação

Temos vindo a explorar a analogia

λ-termo	$\Leftrightarrow$	programa
redução-β	$\Leftrightarrow$	computação
forma normal	$\Leftrightarrow$	resultado
seq. infinita de reduções	$\Leftrightarrow$	não terminação

Um programa que não termina só pode ser entendido como *um programa incorrecto* pelo que somos levados a concluir que o λ-calculus é *demasiadamente expressivo*, na medida em que nos permite representar programas que gostariamos de *excluir*. Surge por isso a questão:

> *Não nos será possível* ajustar *o λ-calculus por forma a excluir os programas (i.e. termos) incorrectos (i.e. que não dispõe de forma normal)?*

Para se perceber a natureza do problema é conveniente estudar o exemplo paradigmático de um λ-termo sem forma normal

$$\Omega \doteq (\lambda x.x\ x)\ \lambda x.x\ x$$

A essência do problema está na *auto-aplicação* a que a variável ligada está sujeita, i.e.

$$\lambda x.x\ x$$

Note-se que nos deparamos desde logo com um problema na leitura intuitiva deste termo: *representa a função que aplica ao argumento esse próprio argumento* — mas então, o argumento recebido é uma função ou um valor do domínio dessa função???

Tipos

Para eliminar a possibilidade da *auto-aplicação* devemos *regular* a aplicação dos termos classificando-os devidamente. Para o fazer vamos introduzir uma nova entidade no sistema λ: **os tipos**. O sistema resultante será designado por **lambda calculus com tipos** e denotado sinteticamente por $\boldsymbol{\lambda^{\rightarrow}}$.

Os tipos são determinados pela seguinte gramática:

$$\mathcal{T} ::= \mathcal{U} \mid \mathcal{T} \rightarrow \mathcal{T}$$

onde $\mathcal{U}$ representa um conjunto de *variáveis de tipo*.

Notação e convenções:

- Para designar variáveis de tipos iremos utilizar o caracter grego σ (e.g. $\sigma_1, \sigma_2, \cdots, \sigma', \cdots$)
- Para designar tipos iremos utilzar o τ (e.g. $\tau_1, \tau_2, \cdots, \tau', \cdots$)
- Por convenção, o operador $\rightarrow$ associa *à direita* (e.g. o tipo $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ representa o tipo $\tau_1 \rightarrow (\tau_2 \rightarrow \tau_3)$).

Ideia base do sistema: Estabeleceer uma relação entre os λ-termos e os tipos $(:) \subseteq \Lambda \times \mathcal{T}$ (dita a **relação de habitabilidade** — $t : \tau$ lê-se *o termo t habita o tipo τ*). Essa relação irá "regular" as condições para a aplicação dos termos.

λ-calculus *de Curry*

A forma mais simples de introduzirmos os tipos no λ-calculus é mantermos inalteradas todas as definições referentes aos termos (i.e. conjunto Λ dos termos; reduções; etc.) e definir-se sobre esse sistema a relação de habitabilidade.

É importante notar que, no estabelecimento da relação de habitabilidade, dependemos sempre dos tipos que estão associadas às variáveis livres (as variáveis livres traduzem sempre uma interferência do "contexto" no termo considerado). Somos, por isso, conduzidos às seguintes definições:

> *Se x for uma variável e τ um tipo então $x : \tau$ é uma* ***declaração***[a]*. Um conjunto finito de declarações (de variáveis distintas) diz-se um* ***contexto*** *e será denotado por caracteres gregos maiúsculos (e.g. $\Gamma, \Sigma, \cdots$).*

Será normal apresentarmos um contexto $\Gamma = \{x_1 : \tau_1, \cdots, x_n : \tau_n\}$ simplesmente pela sequência $x_1 : \tau_1, \cdots, x_n : \tau_n$ e a união de contextos pela concatenação dessas sequências. Quando $x_i : \tau_i \in \Gamma$ dizemos que $x_i \in dom(\Gamma)$. Quando escrevemos $\Gamma, x : \tau$ fica implícito que $x \notin dom(\Gamma)$.

[a] Utilizamos nas *declarações* o mesmo símbolo que na relação de *habitabilidade*. Daqui não resultará qualquer ambiguidade – como se compreenderá em breve...

$\lambda^{\rightarrow}$ de Curry (cont.)

Estamos agora em condições de apresentar a definição indutiva da relação de habitabilidade. Utilizaremos aqui *regras* análogas às da apresentação dos sistemas lógicos (e.g. dedução natural).

$$\text{Ax}\quad \frac{}{\Gamma, x:\tau \vdash x:\tau}$$

$$\text{Abs}\quad \frac{\Gamma, x:\tau_1 \vdash M:\tau_2}{\Gamma \vdash \lambda x.M:\tau_1 \rightarrow \tau_2}$$

$$\text{App}\quad \frac{\Gamma \vdash M:\tau_1 \rightarrow \tau_2 \quad \Gamma \vdash N:\tau_1}{\Gamma \vdash M\ N:\tau_2}$$

Asserções de tipos:

A asserção fundamental no sistema $\lambda^{\rightarrow}$ é $\boldsymbol{\Gamma \vdash M : \tau}$ — dizemos, nesse caso, que o termo M habita o tipo τ no contexto Γ. Mesmo correndo o risco de sermos redundantes, convém frisar que:

> *A* asserção $\Gamma \vdash M : \tau$ verifica-se se e só se existir uma árvore (construída com as regras estabelecidas) com raiz $\Gamma \vdash M : \tau$.

Quando, para um λ-termo M, existe um contexto Γ e um tipo τ tal que $\Gamma \vdash M : \tau$ dizemos que o termo M **é tipável**. O objectivo inerente à introdução de tipos é *restringirmo-nos ao sub-conjunto de* Λ *constituído pelos termos tipáveis.*

Exemplos: (construa as árvores correspondentes)

$$\begin{aligned} \lambda x.x &: \tau \rightarrow \tau \\ \lambda xy.x &: \tau_1 \rightarrow \tau_2 \rightarrow \tau_1 \\ \lambda xyz.x\ z\ (y\ z) &: (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3) \rightarrow (\tau_1 \rightarrow \tau_2) \rightarrow \tau_1 \rightarrow \tau_3 \end{aligned}$$

Propriedades do $\lambda^{\rightarrow}$ de Curry

PROPRIEDADES SIMPLES: (demonstráveis por indução sobre $\Gamma \vdash M : \tau$)

- $\Gamma \vdash M : \tau \quad \Rightarrow \quad dom(\Gamma) \subseteq FV(M)$
- $\Gamma \vdash M : \tau \ \& \ \Gamma \subseteq \Gamma' \quad \Rightarrow \quad \Gamma' \vdash M : \tau$
- $\Gamma \vdash M : \tau \quad \Rightarrow \quad \Gamma\dagger FV(M) \vdash M : \tau$, onde $\Gamma\dagger FV(M)$ denota a restrição do contexto Γ por $FV(M)$ (i.e. só as declarações das variáveis em $FV(M)$ são mantidas).

Um termo pode, neste sistema, possuir vários tipos. De facto, O **lema da substituição dos tipos** diz-nos[a]

$$\Gamma \vdash M : \tau \quad \Rightarrow \quad \Gamma[\sigma := \tau'] \vdash M : \tau[\sigma := \tau']$$

Este resultado decorre directamente do facto de, no *Axioma*, o tipo introduzido ser arbitrário. Este resultado pode facilmente ser estendido para **substituições** (das variáveis de tipos)[b]

Mesmo se não nos podemos referir *ao tipo de um termo*, podemos referir o **tipo principal** de um termo:

Um tipo τ diz-se o ***tipo principal*** *do termo M se,*

$$\Gamma \vdash M : \tau' \quad \Rightarrow \quad \exists \text{ uma substituição } s \in \mathcal{U} \rightarrow \mathcal{T} : \tau' = \tau[s]$$

Os tipos principais *são essencialmente únicos* (a menos da renomeação de variáveis).

EXEMPLO: $\emptyset \vdash S : (\tau \rightarrow \tau \rightarrow \tau) \rightarrow (\tau \rightarrow \tau) \rightarrow \tau$ — Mas não se trata de um tipo principal para S.

[a] Neste resultado, $\tau[\sigma := \tau']$ denota a substituição, no tipo τ, da variável σ pelo tipo τ'. $\Gamma[\sigma := \tau']$ denota a substituição, em todos os tipos intervenientes nas declarações do contexto Γ, da substituição referida.

[b] Uma substituição é uma função de valoração para variáveis distinta da identidade num número finito de pontos do domínio, i.e. $s \in \mathcal{U} \rightarrow \mathcal{T}$ tal que $\{\tau | s(\tau) \neq \tau\}$ é finito.

Propriedades de $\lambda^{\rightarrow}$

Os tipos do λ-calculus devem interagir correctamente com a redução-β. O *lema da redução* estabelece esta compatibilidade.

Lema da redução: Se $\Gamma \vdash M : \tau$ e $M \twoheadrightarrow N$ então $\Gamma \vdash N : \tau$.

Este resultado é facilmente estabelecido por indução sobre a definição da relação de redução.

A justificação última dos tipos encontra eco nos próximos resultados.

Normalização Fraca: Se $\Gamma \vdash M : \tau$ então M possui uma forma normal.

Normalização Forte: Se $\Gamma \vdash M : \tau$ então qualquer sequência de redução $M \rightarrow M_1 \rightarrow \cdots$ é finita.

Estes resultados vem justificar a introdução dos tipos no λ-calculus. A sua prova não é evidente pelo que o vamos admitir sem mais reservas.

Inferência de Tipos

Tendo referido que o objectivo do sistema é restringirmo-nos aos termos tipáveis, levanta-se imediatamente a questão de *como decidir se um dado termo admite um tipo.*

Problema de **Inferência de tipos**: Dado um termo M, determinar (se existir) um contexto Γ e um tipo τ tal que $\Gamma \vdash M : \tau$. A este problema dá-se o nome de **inferência de tipos**.

Observação: No sistema de tipos que consideramos, podemos observar que a estrutura do termo determina univocamente qual é a regra a utilizar – o problema está em "como preencher devidamente" os espaços referentes aos contextos e aos tipos nos nodos da árvore construída...

Notação: Se τ é um tipo, Γ um contexto e s uma substituição, denotamos por $\tau[s]$ a aplicação da substituição ao tipo τ; $\Gamma[s]$ à aplicação da substituição a todos os tipos das declarações de Γ. A composição de substituições é denotada pelo operador $\circ$.

Ideia do algoritmo `infereT`: Dado um contexto Γ, um termo M e um tipo τ [a] determinar a substituição s (se existir) para a qual se verifica a asserção $\Gamma[s] \vdash M : \tau[s]$[b]

Assim, para aplicar o algoritmo `infereT` para resolver o problema descrito atrás devemos, para um termo M, construír um contexto Γ e um tipo τ que não nos restrinja o espaço de busca do algoritmo. Mas esse objectivo é trivialmente conseguido atribuindo a todos os tipos envolvidos variáveis de tipos *novas* e considerando um contexto onde se declaram somente as variáveis livres do termo.

[a] Mas note-se que **não se requer** que se verifique a asserção $\Gamma \vdash M : \tau$ — nesse caso estava o problema resolvido.

[b] De facto, vamos pretender a substituição mais geral, i.e. qualquer outra substituição que satisfaça a asserção pode ser factorizada por essa.

Inferência de tipos (cont.)

Para simplificar a apresentação do algoritmo, *vamos assumir que sempre que uma unificação ou uma invocação recursiva do algoritmo falhe, todo o algoritmo falha.*

tipoDe(M)

$\sigma_0, \sigma_1, \sigma_2, \cdots \leftarrow$ variáveis novas
$x_1, \cdots, x_n \leftarrow$variáveis livres de M
$\Gamma \leftarrow [x_1 : \sigma_1, \cdots, x_n : \sigma_n]$
$s \leftarrow infereT(\Gamma, \sigma_0, M)$
$res \leftarrow \sigma_0[s]$

infereT(Γ, τ, M)

$M \equiv x$
$\tau' \leftarrow$ tipo de x declarado em Γ
$res \leftarrow unif(\tau, \tau')$

$M \equiv \lambda z.M'$
$\sigma_1, \sigma_2 \leftarrow$ variáveis novas
$s_1 \leftarrow unif(\tau, (\sigma_1 \rightarrow \sigma_2))$
$s_2 \leftarrow infereT((\Gamma, z : \sigma_1)[s_1], \sigma_2[s_1], M')$
$res \leftarrow s_2 \circ s_1$

$M \equiv (M'N')$
$\sigma_1 \leftarrow$ variável nova
$s_1 \leftarrow infereT(\Gamma, \sigma_1, N')$
$s_2 \leftarrow infereT(\Gamma[s_1], (\sigma_1 \rightarrow \tau)[s_1], M')$
$res \leftarrow s_2 \circ s_1$

O algoritmo apresentado determina *o tipo principal* de um termo. Esta propriedade resulta de o procedimento $unif$ determinar o unificador mais geral ($m.g.u$) entre dois tipos.

Síntese de Termos

O problema dual ao da inferência de tipos é designado por **síntese de termos**: dado um tipo τ determinar um *termo fechado*[a] M tal que

$$\emptyset \vdash M : \tau$$

Algumas questões são pertinentes relativamente ao problema proposto:

- Será que existe, para qualquer tipo τ, um termo $M : \tau$?
- Existindo $M : \tau$, será fácil determinar esse termo (em particular, será um problema mais ou menos complicado do que a inferência de tipos)?

Para responder a estas questões vamos estabelecer uma analogia entre as soluções deste problema e as provas de proposições lógicas.

[a] Só faz sentido considerar-se o problema da síntese de termos para termos fechados – admitir termos abertos torna trivial (e sem qualquer interesse) a sua resolução: a asserção $x : \tau \vdash x : \tau$ é trivialmente satisfeita (axioma).

Lógica proposicional mínima

Vamos considerar um fragmento da lógica proposicional com uma única conectiva: a implicação. Denominamos essa lógica por *lógica proposicional mínima*.

Sintaxe:

$$\mathcal{L} ::= Prop \quad | \quad \mathcal{L} \supset \mathcal{L}$$

Regras de Dedução Natural:

(Apresentadas na forma de sequentes – recorde ELP I)

$$(\text{I} \supset) \quad \frac{\Gamma, P \vdash Q}{\Gamma \vdash P \supset Q}$$

$$(\text{E} \supset) \quad \frac{\Gamma \vdash P \supset Q \qquad \Delta \vdash P}{\Gamma, \Delta \vdash Q}$$

Sobre o problema de síntese de termos

A um dado tipo podemos associar uma fórmula da lógica proposicional mínima — às variáveis de tipos associamos proposições e a $\rightarrow$ associamos $\supset$. Por outro lado, as regras de dedução natural estão em correspondência com as regras de tipos do sistema-λ (quando atendemos unicamente à porção referente aos tipos).

Assim, estamos em condições de estabelecer um paralelo entre a possibilidade de resolução do problema de síntese de termos com a prova da fórmula lógica associada:

> *O problema de síntese de termos tem solução se e só se o tipo visto como uma fórmula proposicional for uma tautotogia.*

Este resultado permite considerar os λ-termos como uma sintaxe para as provas da lógica proposicional minima.

Analogia de *Curry-Howard*

O resultado anterior permite-nos avançar com uma analogia entre os sistemas λ e o domínio lógico.

Computação	**λ-calculus**	**Lógica**
programa	λ-termo	prova
tipo (especificação)	tipo	fórmula
computação	redução	normalização de provas

O sistema $\lambda^{\rightarrow}$ pode ser identificado com a *lógica proposicional mínima*. Mas esta correspondência é apenas "a ponta do *iceberg*". O que é realmente interessasnte é estender a analogia para sistemas de tipos mais ricos (ou, visto de outro ângulo, lógicas mais expressivas).

Sistema de *Hilbert* para lógica mínima

A analogia de *Curry-Howard* permite-nos concretizar a equivalência entre diferentes apresentações de sistemas lógicos.

Considere-se uma apresentação de *Hilbert* para a lógica minimal.

Axiomas

- $\tau_1 \supset \tau_2 \supset \tau_1$
- $(\tau_1 \supset \tau_2 \supset \tau_3) \supset (\tau_1 \supset \tau_2) \supset \tau_1 \supset \tau_3$

Regra de inferência

$$\text{(Modus Ponens)} \qquad \tau_1 \supset \tau_2 \quad \& \quad \tau_1 \;\Rightarrow\; \tau_2$$

À luz da analogia de Curry-Howard identificamos facilmente os axiomas como correspondendo aos combinadores K e S. A regra de inferência pode ser identificada com a regra do sistema de tipos da *aplicação*. Assim, uma prova neste sistema corresponde a um termo construido unicamente à custa da aplicação de combinadores S e K.

Se nos recordarmos do algoritmo que permite exprimir um termo fechado como a aplicação de combinadores S e K, vemos que esse algoritmo *codifica* a prova de equivalência entre os sistemas lógicos (dedução natural e Hilbert).

Observação: O algoritmo referido não considera tipos. Assim, e para validar esse algoritmo no contexto do sistema com tipos, é necessário verificar que este algoritmo preserva a propriedade de um termo possuir um tipo (o lema da redução e a correção do algoritmo garantem que o próprio tipo é preservado).

λ-calculus com tipos *de Church*

O sistema $\lambda^{\rightarrow}$ de Curry foi definido sobre os termos do λ-calculus sem tipos. Dessa forma, para um termo M, somos obrigados a estabelecer uma asserção de tipo ($\Gamma \vdash M : \tau$) por forma a fazermos uso dos importantes resultados do sistema. Esse passo envolve tipicamente um processo de inferência de tipos.

Como alternativa, podemos considerar um sistema onde os próprios termos já contenham informação relativa aos tipos envolvidos. A essa formulação damos o nome de $\lambda^{\rightarrow}$ **de Church** e os termos são definidos directamente com as regras de tipos.

Regras de boa-formação:

$$\text{Ax} \quad \frac{}{\Gamma, x : \tau \vdash x : \tau}$$

$$\text{Abs} \quad \frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . M : \tau_1 \rightarrow \tau_2}$$

$$\text{App} \quad \frac{\Gamma \vdash M : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash N : \tau_1}{\Gamma \vdash M\ N : \tau_2}$$

Note-se que, nos termos, *só se inclui informação dos tipos nas abstrações.* É facil de ver (e teremos oportunidade de o verificar) que essa informação é suficiente para reconstruir *toda a árvore de formação* do termo.

$\lambda^{\rightarrow}$ de Church (cont.)

A sintaxe dos termos que foi apresentada é muito complexa (corresponde a uma gramática dependente do contexto). Por esse motivo é normal considerar-se uma noção de **pré-termo** (um possível termo) definido pela gramática

$$\begin{array}{rcl} \mathcal{T} & ::= & \mathcal{U} \mid \mathcal{T} \rightarrow \mathcal{T} \\ \Lambda' & ::= & \mathcal{V} \mid (\Lambda'\ \Lambda') \mid \lambda \mathcal{V} : \mathcal{T}.\Lambda' \end{array}$$

Sobre esta noção resta determinar quando um *pré-termo* corresponde realmente a um *termo*, i.e. pode ser construído por uma dada árvore — a esse problema damos a designação de **verificação da boa formação do termo**[a].

NOTAÇÃO: Consideram-se as mesmas convenções sintácticas do λ-termos sem tipos...

EXEMPLOS:

termos bem formados

- $\lambda x : \tau.x$
- $\lambda x : \tau_1, y : \tau_2.x$
- $\lambda x : \tau_1 \rightarrow \tau_2 \rightarrow \tau_2, y : \tau_1 \rightarrow \tau_2, z : \tau_1.x\ z\ (y\ z)$

termos mal formados

- $\lambda x : \tau.x\ x$
- $\lambda x : \tau_1 \rightarrow \tau_2, y : \tau_2.x\ y$

[a]Num abuso de linguagem designamos os "pré-termos" por "termos" a que sujeitamos postriormente a um teste de boa-formação. Só os "termos bem formados" é que são realmente "termos"

Verificação da *boa-formação* dos termos

verificaTermo$(\Gamma; M)$

$M \equiv x$

$res \leftarrow$ tipo de x declarado em Γ

$M \equiv \lambda z : \tau_1 . M'$

$\tau_2 \leftarrow infereT(\Gamma, z : \tau_1; M')$

$res \leftarrow \tau_1 \rightarrow \tau_2$

$M \equiv (M'N')$

$(\tau_1 \rightarrow \tau_2) \leftarrow verificaTermo(\Gamma; M')$ —obs.: (*)pode falhar...

$\tau_3 \leftarrow verificaTermo(\Gamma; N')$

Se $\tau_1 == \tau_3$ então $res \leftarrow \tau_2$ senão $FALHA$

Observações:

- O algoritmo pode falhar se o termo não for bem formado. Em particular, em (*) o algoritmo falha se a concordância de padrões não for possível.
- Note-se que a estrutura do algoritmo é a mesma do de inferência de tipos. No entanto, existe uma *diferença fundamental*: a operação de unificação é substituida pelo teste de igualdade.
- No caso do termo ser bem formado o algoritmo retorna o tipo do termo verificado. Esse tipo é completamente caracterizado pelo próprio termo (i.e. é único), facto que resulta do algoritmo não permitir qualquer flexibilidade no resultado.

Propriedades do sistema $\lambda^{\rightarrow}$ de Church

As propriedades do sistema $\lambda^{\rightarrow}$ de Church são essencialmente as mesmas do sistema de Curry. Assim, podemos estabelecer:

Propriedade de Church-Rosser - Resulta da propriedade do sistema sem tipos.

Normalização Fraca e Forte - Estabelecida de forma análoga ao sistema de Curry.

Lema da redução - Estabelecida de forma análoga ao sistema de Curry.

A principal diferença em relação ao sistema de Curry surge nas propriedades das asserções de tipos:

Unicidade dos tipos:

$$\text{Se } \Gamma \vdash M : \tau \text{ e } \Gamma \vdash M : \tau' \text{ então } \tau == \tau'$$

Comparação entre $\lambda^{\rightarrow}$ de *Curry* e *Church*

Vimos que, ao nível das principais propriedades dos sistemas, pouco separa as formulações de Curry e Church. De facto, estes devem ser considerados como *formulações diferentes do mesmo sistema-*λ (λ-calculus com tipos simples)[a]. Numa perspectiva prática, poderíamos comparar o comportamento de um hipotético animador do sistema como:

Curry

1. utilizador fornece o termo M (sem tipos)
2. faz-se uso do algoritmo de inferência de tipos para verificar se o termo M possui um tipo
3. (caso M possua um tipo) o termo M é reduzido — processo que termina pelo resultado de normalização forte.

Church

1. utilizador fornece o termo M (abstrações estão etiquetadas com os tipos)
2. sistema verifica a boa formação do termo
3. (caso M seja bem formado) o termo M é reduzido — termina...

A diferença fundamental está na complexidade do 2º passo do procedimento. A inferência de tipos é um processo mais complexo do que a verificação do termo. *Mais importante ainda*, estas diferenças tendem a aumentar assim que nós elaboramos o sistema de tipos — em particular, pode ocorrer o caso (e ocorre mesmo) de sistemas para os quais a *inferência de tipos* não ser possível continuando a ser facilmente realizada a verificação dos termos.

EM RESUMO: *podemos dizer que as formulações são equivalentes se dispusermos de um algoritmo que realize a inferência de tipos...*

[a] Esta afirmação pode ser formalizada estabelecendo um isomorfismo entre ambos os sistemas