Tipos, Provas e o Problema de Existência de Habitantes

Mauricio Ayala-Rincón
Grupo de Teoria da Computação - GTC/UnB
Departamento de Matemática, Universidade de Brasília

Escola de Verão 2005

Brasília, 17 de janeiro de 2005

GTC/UnB: www.mat.unb.br/∼ayala/TCgroup

Plano da Apresentação

0. Motivação: provas e programas

- $\underbrace{\text{Programas}}_{\text{programador}}$ são aplicados no tratamento de $\underbrace{\textit{sistemas críticos}}_{\text{humanos}}$!

$\Rightarrow$ $\underbrace{\text{Soluções algorítmicas de problemas devem ser } \textit{provadas} \text{ corretas}}_{\text{matemático}}$.

$\Rightarrow$ $\underbrace{\text{Implementações (} \textit{programas} \text{) de soluções algorítmicas devem ser corretas}}_{\text{matemático+cientista da computação}}$.

0. Motivação: provas e programas

Exemplo: cálculo do máximo divisor comum mdc

$$\underbrace{\textbf{Teorema}\ \text{[Euclid 320-275 BC]} \forall n \geq 0, m > 0, mdc(n, m) = mdc(m, n \text{ mod } m)}_{\text{idéia}}$$

$$\underbrace{\begin{array}{l} \text{procedimento } mdc(m, n) \\ \quad \text{se } m < n \text{ então } mdc(n, m) \\ \quad \text{se não } (m \geq n) \\ \qquad mdc(m - n, n) \\ \text{Fim procedimento} \end{array}}_{\text{programa}}$$

0. Motivação: provas e programas

$$\underbrace{mdc(6,4) \rightarrow mdc(2,4) \rightarrow mdc(4,2) \rightarrow mdc(2,2) \rightarrow mdc(0,2) \rightarrow mdc(2,0) \rightarrow \cdots}_{\text{problema: loop infinito}}$$

Prova de totalidade: Domínio $\mathbb{N}$ (Tipo de objetos)

BI: $mdc(0,n)$ indefinida! Defina $mdc(0,n) = n$.
PI: Suponha $mdc(k,n)$ bem definida para qualquer n e $k < m$, com $m > 0$.
Então $mdc(m,n)$ bem definida:
Caso 1: $m > n$. $mdc(m,n) = mdc(m-n,n)$ Aplica-se HI somente se $n > 0$!
Defina $mdc(m,0) = m$.
Caso 2: $m \leq n$. $mdc(m,n) = mdc(n,m)$ que está bem definido por HI.

0. Motivação: provas e programas

$$\underbrace{\begin{array}{l}
\text{procedimento } mdc(m,n) \\
\quad \text{se } m = 0 \text{ então } n \\
\quad \text{se não } (m > 0) \\
\quad\quad \text{se } m < n \text{ então } mdc(n,m) \\
\quad\quad \text{se não } (m > 0 \& m \geq n) \\
\quad\quad\quad \text{se } n = 0 \text{ então } m \\
\quad\quad\quad \text{se não } (m > 0 \& n > 0 \& m \geq n) \\
\quad\quad\quad\quad mdc(m-n,n) \\
\text{Fim procedimento}
\end{array}}_{\text{programa extraído da especificação provada correta}}$$

1. λ-calculus

• [30's, século XX] Alonzo Church (e Haskell Curry) defin(em) o λ-calculus (e a lógica combinatória) como um mecanismo algorítmico para computar funções numéricas.

• [1935/36] Stephen Kleene e John Rosser demonstram que todas as funções recursivas parciais podem ser representadas no λ-calculus.

• [1936] Alan Turing demonstra que exatamente as funções Turing-computáveis podem ser representadas no lambda calculus.

$\Rightarrow$ Tese de Church-Turing: As funções "computáveis" são as recursivas parciais.

1. λ-calculus

Operações básicas do λ-calculus:

$$\left.\begin{array}{ll} (M \quad N) & \text{Aplicação} \\ \lambda_x.M & \text{Abstração} \end{array}\right\} \quad \text{Operadores}$$

$(\lambda_x.M \quad N) \to_\beta M[x/N]$	β-conversão ou contração
$c\lambda_x.(M \quad x) \to_\eta M$, se $x \notin \mathcal{F}V(M)$	η-conversão ou contração

1. λ-calculus

Exemplos

$$\begin{cases} (\lambda_x.x \;\; \lambda_x.x) \rightarrow_\beta \lambda_x.x & \text{auto-aplicação} \\ \\ (\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x)) \rightarrow_\beta (\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x)) & \text{auto-reprodução} \end{cases}$$

1. λ-calculus

Computações numéricas:

$$C_n \equiv \lambda_x.\lambda_y.(x^n \;\; y) \qquad \text{Numerais de Church}$$

Defina: $A_+ \equiv \lambda_x.\lambda_y.\lambda_p.\lambda_q.((x \;\; p) \;\; ((y \;\; p) \;\; q));$
$A_* \equiv \lambda_x.\lambda_y.\lambda_z.(x \;\; (y \;\; z));$
$A_{exp} \equiv \lambda_x.\lambda_y.(y \;\; x).$

Proposição [Rosser] $\forall n \in \mathbb{N} \begin{cases} A_+C_nC_m = C_{n+m}; \\ A_*C_nC_m = C_{n*m}; \\ A_{exp}C_nC_m = C_{n^m}, \text{ exceto para } m = 0. \end{cases}$

1. λ-calculus

Prova do caso A_*

$A_* C_n C_m = \lambda_{xyz}.x(y\ \ z)C_n C_m \to_\beta \lambda_{yz}.C_n(y\ \ z)C_m \to_\beta \lambda_z.C_n(C_m\ \ z) \to_\beta \lambda_z.\lambda_v.((C_m\ \ z)^n\ \ v) \to^*_\beta \lambda_{zv}.(z^{n*m}\ \ v) = C_{n*m}$.

$((C_n x)^m\ \ (y)) \to^*_\beta (x^{n*m}\ \ (y))$:
BI $((C_n x)^0\ \ (y)) = y$.
PI Suponha vale para m e qualquer n. Então,

$((C_n x)^{m+1}\ \ (y)) = ((C_n x)\ \ ((C_n x)^m\ \ (y))) \overbrace{\to^*_\beta}^{HI} ((C_n x)\ \ (x^{n*m}\ \ (y))) \to_\beta (\lambda_u.x^n u\ \ (x^{n*m}\ \ (y))) \to_\beta (x^n\ \ (x^{n*m}\ \ (y))) = (x^{n*(m+1)}\ \ (y))$.

Casos A_+ e A_{exp} são similares.

1. λ-calculus

λ-implementação de operadores computacionais:

Defina: True $\equiv \lambda_{xy}.x$
False $\equiv \lambda_{xy}.y$
$\langle M, N\rangle \equiv \lambda_z.(zMN)$, para todo par de λ-termos M, N

"*If* B *Then* M *Else* N" $= BMN \begin{cases} \text{True}MN \to_\beta (\lambda_y.M \;\; N) \to_\beta M \\ \text{False}MN \to_\beta (\lambda_y.y \;\; N) \to_\beta N \end{cases}$

Seleção *boolena*: $\begin{cases} \langle M, N\rangle\text{True} \to_\beta \text{True}MN = M \\ \langle M, N\rangle\text{False} \to_\beta \text{False}MN = N \end{cases}$

1. λ-calculus

λ-implementação de operadores computacionais:

Iteração para uma função definida recursivamente

$$\boxed{f(n) = \begin{cases} f_0, & \text{se } n = 0 \\ h(f(n-1), n), & \text{se } n > 0 \end{cases}}$$

Supondo H λ-define h, define-se:

$$T_H \equiv \lambda_p.\langle S(p \ \mathsf{True}), H(p \ \mathsf{False})S(p \ \mathsf{True})\rangle$$

$\forall k$, $T_H(\langle C_k, C_{f(k)}\rangle) = \lambda_p.\langle S(p \ \mathsf{True}), H(p \ \mathsf{False})S(p \ \mathsf{True})\rangle(\langle C_k, C_{f(k)}\rangle) \to_\beta$
$\langle S(\langle C_k, C_{f(k)}\rangle \ \mathsf{True}), H(\langle C_k, C_{f(k)}\rangle \ \mathsf{False})S(\langle C_k, C_{f(k)}\rangle \ \mathsf{True})\rangle =$
$\langle S(C_k), H(C_{f(k)})S(C_k)\rangle$.

1. λ-calculus

λ-implementação de operadores computacionais: Por indução demonstra-se: $T_H^n(\langle C_0, C_{f_0}\rangle) \to_\beta^* \langle C_n, C_{f(n)}\rangle$

Assim, $C_n T_H(\langle C_0, C_{f_0}\rangle)$ False $\to_\beta^* C_{f(n)}$
$$\begin{cases} C_n T_H(\langle C_0, C_{f_0}\rangle) \text{ False} \to_\beta \\ \lambda_v.T_H^n v(\langle C_0, C_{f_0}\rangle) \text{ False} \to_\beta \\ (T_H^n \ (\langle C_0, C_{f_0}\rangle)) \text{ False} \to_\beta^* \\ \langle C_n, C_{f(n)}\rangle \text{ False} = C_{f(n)} \end{cases}$$

Concluindo, f pode ser especificada no lambda calculus como:

$$F \equiv \lambda_x.x T_H(\langle C_0, C_{f_0}\rangle) \text{ False}$$

1. λ-calculus

Aplicação para factorial: $fact(n) = \begin{cases} 1, & \text{se } n = 0 \\ fact(n-1) * n, & \text{se } n > 0 \end{cases}$

$$FACT \equiv \lambda_x.xT_{A_*}(\langle C_0, C_1\rangle) \text{ False}$$

Ou equivalentemente

$$\lambda_x.x\underbrace{\lambda_p.\langle S(p \text{ True}), A_*(p \text{ False})S(p \text{ True})\rangle}_{T_{A_*}}(\langle C_0, C_1\rangle) \text{ False}$$

$$\lambda_x.x\lambda_p.\lambda_z.(z(S(p\lambda_{xy}.x)A_*(p\lambda_{xy}.y)S(p\lambda_{xy}.x)))(\lambda_z.(z\lambda_{uv}.v\lambda_{uv}.uv))\lambda_{xy}.y$$

onde $S \equiv A_+C_1 = \lambda_{xyuv}.((xu)((yu)v))\lambda_{xy}.xy \to_\beta \lambda_{xyuv}.((\lambda_{xy}.xyu)((yu)v))$.

2. λ-calculus com tipos simples

• Alonzo Church (e Haskell Curry) estende(m) o λ-calculus [1940] (e a lógica combinatória [1934]) com tipos.

$$\underbrace{\Lambda \ : \ a \ ::= \ (a \ \ a) \mid \lambda_V.a}_{\text{Sintaxe dos } \lambda\text{-termos livres de tipos}}$$

$$\boxed{\text{Anotações de tipos:} \qquad \lambda x{:}A.M}$$

$$\left.\begin{array}{l} (\lambda_{x:A}.M \ \ N) \rightarrow_\beta M\{N/x\} \\ \lambda_{x:A}.(M \ \ x) \rightarrow_\eta M, \text{ se } x \notin \mathcal{F}V(M) \end{array}\right\} \qquad \text{Beta- e Eta-contração}$$

2. λ-calculus com tipos simples

$$\underbrace{\text{Exemplos} \begin{cases} (\lambda_x.x \;\; \lambda_x.x) \rightarrow_\beta \lambda_x.x & \text{auto-aplicação} \\ (\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x)) \rightarrow_\beta (\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x)) & \text{auto-reprodução} \end{cases}}_{\text{Argumentações paradoxais}}$$

Auto-aplicação faz sentido:

$$(\overbrace{\lambda_{x:A\rightarrow A}.x}^{(A\rightarrow A)\rightarrow A\rightarrow A} \;\; \overbrace{\lambda_{x:A}.x}^{A\rightarrow A}) \rightarrow_\beta \overbrace{\lambda_{x:A}.x}^{A\rightarrow A}$$

Polimorfismo!

2. λ-calculus com tipos simples

Auto-reprodução não faz sentido:

$$(\lambda_{x:\tau_1}.(x\ \ x)\ \ \lambda_{x:\tau_2}.(x\ \ x)) \to_\beta (\lambda_{x:\tau_3}.(x\ \ x)\ \ \lambda_{x:\tau_4}.(x\ \ x))$$

Termo aceitável na linguagem do λ-calculus tipado, mas não tipável!

2. λ-calculus com tipos simples

Julgamentos de tipos $\left\{ \begin{array}{l} \boxed{\Gamma \vdash M : A} \\ \text{“}M \text{ tem tipo } A \text{ no } \textit{contexto } \Gamma\text{”} \end{array} \right.$

contexto $\left\{ \begin{array}{l} \underbrace{x_1{:}A_1, \ldots, x_n{:}A_n} \\ \text{lista de declarações de variavéis} \end{array} \right.$

2. λ-calculus com tipos simples

$$\frac{x \notin \Gamma}{x{:}A, \Gamma \vdash x : A}\ \text{(Start)} \qquad \frac{x \notin \Gamma \qquad \Gamma \vdash M : B}{x{:}A, \Gamma \vdash M : B}\ \text{(Weak)}$$

$$\frac{x{:}A, \Gamma \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \to B}\ \text{(Abs)} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash (M \ \ N) : B}\ \text{(Appl)}$$

Fig. 1: Regras de tipagem do λ-calculus com tipos simples

2. λ-calculus com tipos simples

Exemplo: inferência de tipos para $(\lambda_{x:\tau}.x \;\; \lambda_{x:\rho}.x)$

$$\underbrace{\dfrac{x:\tau \vdash x:\tau(\text{Start})}{\vdash \lambda_{x:\tau}.x:\tau\to\tau}(\text{Abs})}_{\dfrac{x:\rho\to\rho\vdash x:\rho\to\rho(\text{Start})}{\vdash\lambda_{x:\rho\to\rho}.x:(\rho\to\rho)\to\rho\to\rho}(\text{Abs})} \qquad \dfrac{x:\rho\vdash x:\rho(\text{Start})}{\vdash\lambda_{x:\rho}.x:\rho\to\rho}(\text{Abs})$$

$$\dfrac{\vdash \lambda_{x:\rho\to\rho}.x:(\rho\to\rho)\to\rho\to\rho, \qquad \vdash\lambda_{x:\rho}.x:\rho\to\rho}{\vdash(\lambda_{x:\rho\to\rho}.x \;\; \lambda_{x:\rho}.x):\rho\to\rho}(\text{Appl})$$

2. λ-calculus com tipos simples

Exemplo: tentativa de inferência de tipos para $\lambda_{x:\rho}.(x \;\; x)$

$$\frac{x : \tau \vdash x : \tau(\mathsf{Start}) \qquad\qquad x : \rho \vdash x : \rho(\mathsf{Start})}{\text{impossível redefinir } \tau \text{ e/ou } \rho \text{ para a regra Appl!}}(\mathsf{Appl})$$

2. λ-calculus com tipos simples

Problemas relevantes em teoria de tipos:

- Verificação de tipos: Dados M e A determine se existe Γ tal que $\Gamma \vdash M : A$.
- Inferência de tipos: dado M determine Γ e A tais que $\Gamma \vdash M : A$.
- Existência de habitantes: Dado o tipo A. Existem *habitantes* no contexto Γ se, e unicamente se existe um λ-termo M tal que $\Gamma \vdash M : A$.

2. λ-calculus com tipos simples

Revisitando os problemas relevantes em teoria de tipos:

$$\boxed{\underbrace{\Gamma}_{\text{declarações de variáveis}} \vdash \underbrace{M}_{\lambda\text{-termo ou programa}} : \underbrace{A}_{\text{tipos}}}$$

- Verificação de tipos: os tipos desinados para o programa são corretos.
- Inferência de tipos: o programa é correto.
- Existência de habitantes: extração de um programa de uma prova.

3. Isomorfismo de Curry-Howard

Relação entre provas e programas detetada por Haskell Curry [1934-1942], mas so aplicada até a década de 60 (XX) por Nicolaas Govert de Bruijn e William Howard.

$\boxed{\text{Teoria de Tipos}}$ versus $\underbrace{\boxed{\text{Lógica instuicionista}}}_{\text{Luitzen Egbertus Jan Brouwer [1920]}}$

Regras de tipagem do λ-calculus com tipos simples correspondem 1-1 às regras de dedução da lógica intuicionista minimal: regras de tipagem são regras lógicas decoradas com λ-termos tipados.

3. Isomorfismo de Curry-Howard

Lógica intuicionista implicacional

Fórmulas implicacionais são constroídas de *variáveis proposicionais* (denotadas por $A, B, C, \ldots$) usando o conetivo implicacional $\rightarrow$ assim: se σ e τ são fórmulas implicacionais, então $(\sigma \rightarrow \tau)$ o é.

3. Isomorfismo de Curry-Howard

Um julgamento na lógica intuicionista $\boxed{\Omega \vdash_I A}$ denota que "A é uma consequência lógica de Ω".

$$\boxed{\frac{}{\Omega, A \vdash_I A}\,(\text{Axiom}) \quad \frac{\Omega, A \vdash_I B}{\Omega \vdash_I A \to B}\,(\text{Intro}) \quad \frac{\Omega \vdash_I A \to B \qquad \Omega \vdash_I A}{\Omega \vdash_I B}\,(\text{Elim})}$$

Fig. 2: Regras de dedução da lógica intuicionista minimal

Uma fórmula A é uma *tautologia* se, e unicamente se o julgamento $\vdash_I A$ é provável.

3. Isomorfismo de Curry-Howard

Exemplo. $A \to ((A \to B) \to B)$ é uma tautologia:

$$\dfrac{\dfrac{\dfrac{\overline{A, A \to B \vdash_I A \to B}\ (\text{Axiom}) \qquad \overline{A, A \to B \vdash_I A}\ (\text{Axiom})}{A, A \to B \vdash_I B}\ (\text{Elim})}{A \vdash_I (A \to B) \to B}\ (\text{Intro})}{\vdash_I A \to ((A \to B) \to A)}\ (\text{Intro})$$

No contexto do λ-calculus temos:

$$\vdash \lambda x{:}A.\lambda y{:}A \to B.(y\ \ x) : A \to ((A \to B) \to A)$$

3. Isomorfismo de Curry-Howard

Exemplo. **Lei de Peirce**: (PL) $((A \to B) \to A) \to A$

Não é válida na lógica intuicionista!

Exemplo. Uma prova de $(A \to A \to C) \to A \to C$.

3. Isomorfismo de Curry-Howard

Isomorfismo de Curry-Howard: $\Omega \vdash_I A$ é provável na lógica intuicionista minimal se, e somente se $\Gamma \vdash M : A$ é um julgamento de tipos válido no λ-calculus com tipos simples, onde Γ é uma lista de declarações de variáveis de proposições, observadas como tipos, em Ω. O termo M é um λ-term que representa a derivação da prova.

3. Isomorfismo de Curry-Howard

• Inferência de tipos: dado M determine Γ e A tais que $\Gamma \vdash M : A$
corresponde a provas de correção de programas

• Existência de habitantes: O tipo A é *habitado* em Γ se, e somente se existe um λ-term M tal que $\Gamma \vdash M : A$
corresponde à extração de programas de provas

4. O problema de existência de habitantes

• Quantos habitantes fechados tem um tipo τ?

$\Rightarrow$ Resposta: infinitos ou nenhum!

(use $\lambda_{x:\tau}.x$)

• Mais interessante: Quantos habitantes fechados em forma normal tem um tipo τ?

• Um **habitante β-normal** de um tipo é um habitante em β-nf.

$\Rightarrow$ Habitantes β-normais caraterizam os habitantes de um tipo.

4. O problema de existência de habitantes

Algoritmo de Ben-Yeles (1979): para um τ dado decide se o número de termos fechados em β-nf que habitam τ é finito ou infinito; computa o número no caso finito e lista os termos relevantes em ambos os casos.

Richard Statman (1979): O problema de existência de habitantes é PSPACE-completo

5. Conclusões

- Desenvolvimento de programas corretos é essencial.
  – Erros e omisões nas especificações são possíveis
  – Erros na interpretação/compreensão dos métodos e transcrição são possíveis

$$\boxed{\underbrace{\text{Desenvolvimento do método}}_{\text{Tudo bem, obvio :)}}} \rightsquigarrow \boxed{\underbrace{\text{Implementação}}_{\text{Entendi! Fácil, tá pronto :)}}} \rightsquigarrow \boxed{\underbrace{\text{Aplicação}}_{\text{Me ferre ;(}}}$$

Fluxo padrão de desenvolvimento de software/hardware

5. Conclusões

$$\boxed{\boxed{\underbrace{\text{Prova fórmal de correção do método}}_{\text{Asistentes de prova}}} \quad \overbrace{\text{extração do programa}}^{\rightsquigarrow} \quad \boxed{\underbrace{\text{Aplicação}}_{:)}}}$$

Fluxo seguro de desenvolvimento de software/hardware

- Investigação de mecanismos de inferência de tipos e de extração de programas de provas em outras teorias de tipos: subtipos, tipos dependentes, tipos de interseção, etc.

Referências

* J.R. Hindley, Basic Simple Type Theory. Cambridge Tracts in Theoretical Computer Science, n. 42, Cambridge University Press, 1997.

* H. Simmons, Derivation and Computation: Taking the Curry-Howard Correspondence Seriously. Cambridge Tracts in Theoretical Computer Science, n. 51, Cambridge University Press, 2000.

* Kamareddine, F. D. and T. Laan and R. Nederpelt, A Modern Perspective on Type Theory Applied Logic Series 29, Kluewer, 2004.

* F. Kamareddine, T. Laan and R. Nederpelt, Types in logic and mathematics before 1940, in Bull. Symbolic Logic, 8(2):185-245, 2002.

* Proof Assistants using Dependent Type Systems , by Henk Barendregt and

Herman Geuvers, Handbook of Automated Reasoning, edited by Alan Robinson and Andrei Voronkov, 2001, (89 pages)

* Lectures on the Curry-Howard Isomorphism by Morten Heine B. Sorensen (University of Copenhagen), Pawel Urzyczyn (University of Warsaw) (275 pages)