

# Tipos e Provas Matemáticas

MAURICIO AYALA-RINCÓN

GRUPO DE TEORIA DA COMPUTAÇÃO - GTC/UnB

Departamento de Matemática, Universidade de Brasília



ESCOLA DE VERÃO 2007

BRASÍLIA, 9 DE FEVEREIRO DE 2007

GTC/UNB: [www.mat.unb.br/~ayala/TCgroup](http://www.mat.unb.br/~ayala/TCgroup)

# Plano da Apresentação

0. Motivação: provas e programas
1. Cálculo Lambda (Cálculo  $\lambda$ )
2. Cálculo Lambda com tipos simples
3. Isomorfismo de Curry-Howard
4. Conclusões

## 0. Motivação: provas e programas

- Programas são aplicados no tratamento de sistemas críticos!  
programador humanos
- ⇒ Soluções algorítmicas de problemas devem ser *provadas* corretas.  
matemático
- ⇒ Implementações (*programas*) de soluções algorítmicas devem ser corretas.  
matemático+cientista da computação

## 0. Motivação: provas e programas

Exemplo: cálculo do máximo divisor comum  $mdc$

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

(Detalhe: “ $n \text{ MOD } m$ ” computa-se como “ $(n - m) \text{ MOD } m$ ”)

procedimento  $mdc(m, n)$   
 se  $m < n$  então  $mdc(n, m)$   
 se não ( $m \geq n$ )  
 $mdc(m - n, n)$

Fim procedimento

programa

## 0. Motivação: provas e programas

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

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

procedimento  $mdc(m, n)$

se  $m = 0$  então  $n$

se não ( $m > 0$ )

se  $m < n$  então  $mdc(n, m)$

se não ( $m > 0 \& m \geq n$ )

se  $n = 0$  então  $m$

se não ( $m > 0 \& n > 0 \& m \geq n$ )

$mdc(m - n, n)$

Fim procedimento

programa extraído da especificação provada correta

# 1. Cálculo $\lambda$

- [30's, século XX] Alonzo Church (e Haskell Curry) define(m) o cálculo  $\lambda$  (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 cálculo  $\lambda$ .
  - [1936] Alan Turing demonstra que exatamente as funções Turing-computáveis podem ser representadas no lambda calculus.
- ⇒ Tese de Church-Turing: As funções “computáveis” são as recursivas parciais.

# 1. Cálculo $\lambda$

Operações básicas do cálculo  $\lambda$ :

$(M \ N)$	} Operadores
$\lambda_x.M$	

Aplicação  
Abstração

$(\lambda_x.M \ N) \rightarrow_{\beta} M[x/N]$

$\beta$ -conversão ou contração

$\lambda_x.(M \ x) \rightarrow_{\eta} M, \text{ se } x \notin \mathcal{FV}(M)$

$\eta$ -conversão ou contração



# 1. Cálculo $\lambda$

Exemplos	{	$(\lambda_x.x \ \lambda_x.x) \rightarrow_{\beta} \lambda_x.x$	auto-aplicação
		$(\lambda_x.(x \ x) \ \lambda_x.(x \ x)) \rightarrow_{\beta} (\lambda_x.(x \ x) \ \lambda_x.(x \ x))$	auto-reprodução

# 1. Cálculo $\lambda$

Computações numéricas:

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

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]**  $\forall n \in \mathbb{N} \left\{ \begin{array}{l} A_+ C_n C_m = C_{n+m}; \\ A_* C_n C_m = C_{n*m}; \\ A_{exp} C_n C_m = C_n^m, \text{ exceto para } m = 0. \end{array} \right.$

# 1. Cálculo $\lambda$

## Prova do caso $A_*$

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

$$((C_nx)^m\ (y)) \rightarrow_{\beta}^* (x^{n*m}\ (y)):$$

**BI**  $((C_nx)^0\ (y)) = y.$

**PI** Suponha vale para  $m$  e qualquer  $n$ . Então,

$$((C_nx)^{m+1}\ (y)) = ((C_nx)\ ((C_nx)^m\ (y))) \xrightarrow{HI} \rightarrow_{\beta}^* ((C_nx)\ (x^{n*m}\ (y))) \rightarrow_{\beta} (\lambda_u.x^n u\ (x^{n*m}\ (y))) \rightarrow_{\beta} (x^n\ (x^{n*m}\ (y))) = (x^{n*(m+1)}\ (y)).$$

Casos  $A_+$  e  $A_{exp}$  são similares.

# 1. Cálculo $\lambda$

$\lambda$ -implementação de operadores computacionais:

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

"If  $B$  Then  $M$  Else  $N$ "  $= BMN \left\{ \begin{array}{l} \text{True}MN \rightarrow_{\beta} (\lambda_y.M \ N) \rightarrow_{\beta} M \\ \text{False}MN \rightarrow_{\beta} (\lambda_y.y \ N) \rightarrow_{\beta} N \end{array} \right.$

Seleção *booleana*:  $\left\{ \begin{array}{l} \langle M, N \rangle \text{True} \rightarrow_{\beta} \text{True}MN = M \\ \langle M, N \rangle \text{False} \rightarrow_{\beta} \text{False}MN = N \end{array} \right.$

# 1. Cálculo $\lambda$

$\lambda$ -implementação de operadores computacionais:

Iteração para uma função definida recursivamente

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

Supondo  $H$   $\lambda$ -define  $h$ , define-se:

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

$$\begin{aligned} \forall k, T_H(\langle C_k, C_{f(k)} \rangle) &= \lambda_p. \langle S(p \text{ True}), H(p \text{ False})S(p \text{ True}) \rangle(\langle C_k, C_{f(k)} \rangle) \rightarrow_{\beta} \\ &\langle S(\langle C_k, C_{f(k)} \rangle \text{ True}), H(\langle C_k, C_{f(k)} \rangle \text{ False})S(\langle C_k, C_{f(k)} \rangle \text{ True}) \rangle = \\ &\langle S(C_k), H(C_{f(k)})S(C_k) \rangle. \end{aligned}$$

# 1. Cálculo $\lambda$

$\lambda$ -implementação de operadores computacionais: Por indução demonstra-se:

$$T_H^n(\langle C_0, C_{f_0} \rangle) \rightarrow_{\beta}^* \langle C_n, C_{f(n)} \rangle$$

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

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. Cálculo $\lambda$

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

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

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

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

## 2. Cálculo $\lambda$ com tipos simples

- Alonzo Church (e Haskell Curry) estende(m) o cálculo  $\lambda$  [1940] (e a lógica combinatória [1934]) com tipos.

$$\Lambda : a ::= (a \ a) \mid \lambda_V.a$$

Sintaxe dos  $\lambda$ -termos livres de tipos

Anotações de tipos:  $\lambda x:A.M$

$$(\lambda_{x:A}.M \ N) \rightarrow_{\beta} M\{N/x\}$$

$$\lambda_{x:A}.(M \ x) \rightarrow_{\eta} M, \text{ se } x \notin \mathcal{FV}(M)$$

Beta- e Eta-contração



## 2. Cálculo $\lambda$ com tipos simples

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

Auto-aplicação faz sentido:

$$\left( \underbrace{(\lambda_{x:A \rightarrow A}.x \ \lambda_{x:A \rightarrow A}.x)}_{(A \rightarrow A) \rightarrow A \rightarrow A} \ \underbrace{\lambda_{x:A}.x}_{A \rightarrow A} \right) \rightarrow_{\beta} \underbrace{\lambda_{x:A}.x}_{A \rightarrow A}$$

Polimorfismo!

## 2. Cálculo $\lambda$ com tipos simples

Auto-reprodução não faz sentido:

$$(\lambda_{x:\tau_1} \cdot (x \ x) \ \lambda_{x:\tau_2} \cdot (x \ x)) \rightarrow_{\beta} (\lambda_{x:\tau_3} \cdot (x \ x) \ \lambda_{x:\tau_4} \cdot (x \ x))$$

Termo aceitável na linguagem do cálculo  $\lambda$  tipado, mas não tipável!

## 2. Cálculo $\lambda$ com tipos simples

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

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

## 2. Cálculo $\lambda$ com tipos simples

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

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

$$\frac{x:A, \Gamma \vdash M : B}{\Gamma \vdash \lambda x:A.M : A \rightarrow B} \text{ (Abs)}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash (M \ N) : B} \text{ (Appl)}$$

Fig. 1: Regras de *tipagem* do cálculo  $\lambda$  com tipos simples

## 2. Cálculo $\lambda$ com tipos simples

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

$$\frac{x : \tau \vdash x : \tau \text{ (Start)}}{\vdash \lambda_{x:\tau}.x : \tau \rightarrow \tau} \text{ (Abs)}$$

$$\frac{x : \rho \vdash x : \rho \text{ (Start)}}{\vdash \lambda_{x:\rho}.x : \rho \rightarrow \rho} \text{ (Abs)}$$

$$\frac{x : \rho \rightarrow \rho \vdash x : \rho \rightarrow \rho \text{ (Start)}}{\vdash \lambda_{x:\rho \rightarrow \rho}.x : (\rho \rightarrow \rho) \rightarrow \rho \rightarrow \rho} \text{ (Abs)}$$

$$\frac{\vdash \lambda_{x:\rho \rightarrow \rho}.x : (\rho \rightarrow \rho) \rightarrow \rho \rightarrow \rho, \quad \vdash \lambda_{x:\rho}.x : \rho \rightarrow \rho}{\vdash (\lambda_{x:\rho \rightarrow \rho}.x \ \lambda_{x:\rho}.x) : \rho \rightarrow \rho} \text{ (Appl)}$$

## 2. Cálculo $\lambda$ com tipos simples

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

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

## 2. Cálculo $\lambda$ com tipos simples

Problemas relevantes em teoria de tipos:

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

## 2. Cálculo $\lambda$ com tipos simples

Revisitando os problemas relevantes em teoria de tipos:

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

- Verificação de tipos: os tipos designados 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.

Teoria de Tipos

versus

Lógica intuicionista

Luitzen Egbertus Jan Brouwer [1920]

Regras de *tipagem* do cálculo  $\lambda$  com tipos simples correspondem 1-1 às regras de dedução da lógica intuicionista minimal: regras de *tipagem* são regras lógicas decoradas com  $\lambda$ -termos tipados.

## 3. Isomorfismo de Curry-Howard

Lógica intuicionista implicacional

**Fórmulas implicacionais** são construídas de *variáveis proposicionais* (denotadas por  $A, B, C, \dots$ ) usando o conectivo implicacional  $\rightarrow$  assim: se  $\sigma$  e  $\tau$  são fórmulas implicacionais, então  $(\sigma \rightarrow \tau)$  o é.

### 3. Isomorfismo de Curry-Howard

Um julgamento na lógica intuicionista  $\Omega \vdash_I A$  denota que “ $A$  é uma consequência lógica de  $\Omega$ ”.

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

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.  $A \rightarrow ((A \rightarrow B) \rightarrow B)$  é uma tautologia:

$$\frac{\frac{\frac{}{A, A \rightarrow B \vdash_I A \rightarrow B} \text{(Axiom)}}{}{A, A \rightarrow B \vdash_I A} \text{(Elim)}}{\frac{A, A \rightarrow B \vdash_I B}{A \vdash_I (A \rightarrow B) \rightarrow B} \text{(Intro)}}{\vdash_I A \rightarrow ((A \rightarrow B) \rightarrow A)} \text{(Intro)}$$

No contexto do cálculo  $\lambda$  temos:

$$\vdash \lambda x:A. \lambda y:A \rightarrow B. (y \ x) : A \rightarrow ((A \rightarrow B) \rightarrow A)$$

### 3. Isomorfismo de Curry-Howard

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

Não é válida na lógica intuicionista!

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

### 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 cálculo  $\lambda$  com tipos simples, onde  $\Gamma$  é uma lista de declarações de variáveis de proposições, observadas como tipos, em  $\Omega$ . O termo  $M$  é um termo  $\lambda$  que representa a derivação da prova.

### 3. Isomorfismo de Curry-Howard

- Inferência de tipos: dado  $M$  determine  $\Gamma$  e  $A$  tais que  $\Gamma \vdash M : A$  corresponde a provas de correção de programas
  
- Existência de habitantes: O tipo  $A$  é *habitado* em  $\Gamma$  se, e somente se existe um  $\lambda$ -termo  $M$  tal que  $\Gamma \vdash M : A$  corresponde à extração de programas de provas

## 5. Conclusões

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



Fluxo “padrão” de desenvolvimento de software/hardware



## 5. Conclusões



Fluxo “seguro” de desenvolvimento de software/hardware

- Investiga33o de mecanismos de infer3ncia de tipos e de extra33o de programas de provas em outras teorias de tipos: subtipos, tipos dependentes, tipos de interse33o, 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, Kluwer, 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)