# Sustituciones Explícitas
## en la implementación de ambientes computacionales

Mauricio Ayala Rincón

Grupo de Teoria da Computação
Universidade de Brasília - UnB, Brasil

Investigación financiada por el
*Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq*

2CCC, Bogotá, 19 de Abril de 2007

# Organización de la Charla



# Substitución

- Escencial para implementación de sistemas computacionales:
  - Lenguajes de programación
  - Lenguajes de especificación
- *Matching*
- Reducción o simplificación
- Deducción

# Substitución

Traducción via simplificación:

| La | classe | de | español | es | gratis |
|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| A | aula | de | espanhol | é | de graça |

# Substitución

Programación via reescritura (matching e reducción (ELAN)):

| | | | |
|---|---|---|---|
| [*spl*1] | split(nil,nil,nil) | $\Rightarrow$ | nil end |
| [*spl*1] | split(nil,nil,a.nil) | $\Rightarrow$ | a.nil end |
| [*spl*2] | split(l1,l2,a.b.l) | $\Rightarrow$ | split(a.l1,b.l2,l) end |
| [*spl*3] | split(l1,l2,a.nil) | $\Rightarrow$ | split(a.l1,l2,nil) end |
| [*spl*4] | split(l1,l2,nil) | $\Rightarrow$ | merge(split(nil,nil,l1),split(nil,nil,l2)) end |
| [*mg*1] | merge(nil,l1) | $\Rightarrow$ | l1 end |
| [*mg*2] | merge(l1,nil) | $\Rightarrow$ | l1 end |
| [*mg*3] | merge(a.l1,b.l2) | $\Rightarrow$ | a.merge(l1,b.l2) if a $\leq$ b end |
| [*mg*4] | merge(a.l1,b.l2) | $\Rightarrow$ | b.merge(a.l1,l2) if a $>$ b end |
| [*msort*] | msort(l) | $\Rightarrow$ | split(nil,nil,l) end |

Cuadro: Merge-sort em ELAN

# Substitución

## Programación via reescritura (matching e reducción en ML):

```
let rec merge l1 l2 =
match l1 with
   | [] → l2
   | a::lp1 → (match l2 with
                  | [] → l1
                  | b::lp2 → (if a ¡= b
                                 then (a :: (merge lp1 l2) )
                                 else (b :: (merge l1 lp2) ))) ;;
let rec split l1 l2 l =
match l with
   | [] → ( match l2 with
              | [] → ( match l1 with
                         | [] → []
                         | _ → (merge (split [] [] l1) (split [] [] l2) ))
              | _ → (merge (split [] [] l1) (split [] [] l2) ) )
   | a::[] → ( match l2 with
                 | [] → ( match l1 with
                            | [] → a::[]
                            | _ → (split (a::l1) l2 []) )
                 | _ → (split (a::l1) l2 []) )
   | a::b::lp → split (a::l1) (b::l2) lp ;;
let msort l = split [] [] l ;;
```

Cuadro: Merge-sort em ML

# Substitución

Programación via reescritura (matching e reducción):

$$\text{msort}(33.44.22.55.\text{nil}) \to_{msort} \text{split}(\text{nil},\text{nil},33.44.22.55.\text{nil}) \to_{spl2} \text{split}(33.\text{nil},44.\text{nil},22.55.\text{nil})$$

$$\text{split}(33.\text{nil},44.\text{nil},22.55.\text{nil}) \to_{spl2} \text{split}(22.33.\text{nil},55.44.\text{nil},\text{nil})$$

Regla: $[spl2]\ \ \text{split}(l1,l2,a.b.l) \Rightarrow \text{split}(a.l1,b.l2,l)\ \text{end}$

Sustitución: $\sigma := \{l1 \mapsto 33.\text{nil},\ l2 \mapsto 44.\text{nil},\ a \mapsto 22,\ b \mapsto 55,\ l \mapsto \text{nil}\}$

$$\to \ldots 22.33.44.55.\text{nil}$$

Universidade de Brasília

# Substitución

Análisis matemático (cálculo):

$$ln(x) := \int_1^x \frac{1}{x} dx$$

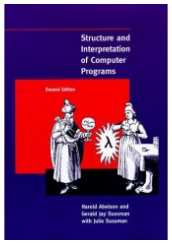
Computar $ln(n)$? Alternativas

$$\begin{cases} \int_1^n \frac{1}{n} dn \\ \int_1^n \frac{1}{x} dx \\ \int_1^x \frac{1}{n} dn \\ \dots \end{cases}$$

Universidade de Brasília

# Historia

- Tratamiento con técnicas *ad hoc*:




Abelson & Susman (Structure and Interpretation of Computer Pogramms) MIT, 1985:
*Despite the fact that substitution is a straightforward idea, it turns out to be surprisingly complicated to give a rigorous mathematical definition of the substitution process... Indeed, there is a long history of erroneous definitions of substitution in the literature of logic and programming semantics.*

- Implementaciones tratan aparte la noción implícita de sustitución.

Universidade de Brasília

# Historia

- Nicolaas Govert de Bruijn (1918-). Matemático holandés lider del Projecto Automath.
- Projecto Automath iniciado en 1967. Primer proyecto que uso tecnología computacional para mecanizar el razonamento matemático:

Especificación y verificación del libro-texto de (1877-1938) Edmund Landau's *Grundlagen der Analyses*, Leipzig 1930.

Universidade de Brasília

# Historia

- 



  `http://automath.webhop.net/`
- Automath es considerado predecesor de asistentes de demostración modernos: Coq, Nurpl, Isabelle, ...
- 

[Kam03], [NGdV94], etc.

Universidade de Brasília

# Historia

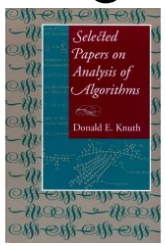
- En el proyecto Automath de Bruijn desarrollo la primera formalización de una versión del cálculo $\lambda$ con un tratamiento explícito de la operación de substitución [dB78]


- Fairouz Kamareddine (Editora): *N.G. de Bruijn was a well established mathematician before deciding in 1967 at the age of 49 to work on a new direction related to Automating Mathematics. In the 1960s he became fascinated by the new computer technology and decided to start the new Automath project where he could check, with the help of the computer, the correctness of books of mathematics. Through his work on Automath, de Bruijn started a revolution in using the computer for verification, and since, we have seen more and more proof-checking and theorem-proving systems.*

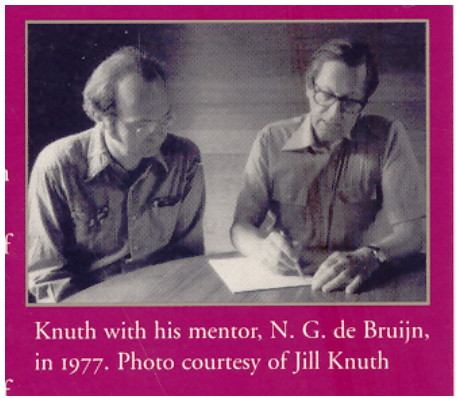
Universidade de Brasília

# Historia

- La influencia de N.G. De Bruijn en computación no se restringe a Automath:




"Selected Papers on Analysis of Algorithms"
(CSLI, 2000)
Donal Knuth dedica su libro a su *mentor* de Bruijn.


Knuth with his mentor, N. G. de Bruijn, in 1977. Photo courtesy of Jill Knuth

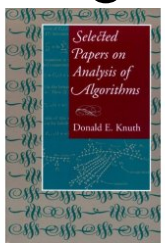
*... I'm dedicating this book to N.G. "Dick" de Bruijn because his influence can be felt on every page.*
*Ever since the 1960s he has been my chief mentor, the main person who would answer my questions when I was stuck on a problem that I had not been taught how to solve.*
*I originally wrote Chapter 26 for his $(3 \cdot 4 \cdot 5)$th birthday; now he is $3^4$ years young as I gratefully present him with this book.*

*Donald E. Knuth*

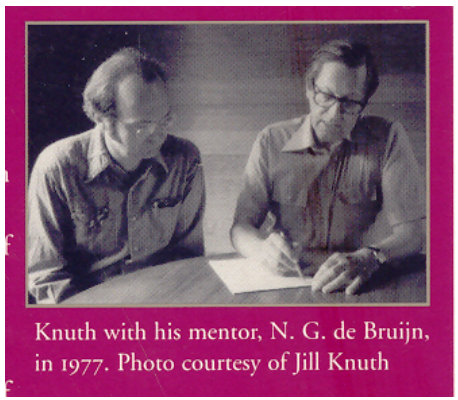
Universidade de Brasília

# Historia

- La influencia de N.G. De Bruijn en computación no se restringe a Automath:




"Selected Papers on Analysis of Algorithms"
(CSLI, 2000)
Donal Knuth dedica su libro a su *mentor* de Bruijn.

- 
Knuth with his mentor, N. G. de Bruijn, in 1977. Photo courtesy of Jill Knuth

*... I'm dedicating this book to N.G. "Dick" de Bruijn because his influence can be felt on every page.*
*Ever since the 1960s he has been my chief mentor, the main person who would answer my questions when I was stuck on a problem that I had not been taught how to solve.*
*I originally wrote Chapter 26 for his $(3 \cdot 4 \cdot 5)$th birthday; now he is $3^4$ years young as I gratefully present him with this book.*

*Donald E. Knuth*

Universidade de Brasília

# Tipos

- Discriminación de clases de objetos
- Usados implicitamente en sistemas intuitivos
  - *Elementos* de Euclides
- Necesidad de definición explícita para sistemas abstractos

# Tipos

- Discriminación de clases de objetos
- Usados implicitamente en sistemas intuitivos
  - *Elementos* de Euclides
- Necesidad de definición explícita para sistemas abstractos

# Tipos

- Discriminación de clases de objetos
- Usados implicitamente en sistemas intuitivos
  - *Elementos* de Euclides
- Necesidad de definición explícita para sistemas abstractos

# Historia (tipos)

- Tratamiento de paradojas e inconsistencias en la formalización de las matemáticas:
  - Auto-referencia, auto-reproducción
- Tipos simples en el cálculo $\lambda$ [A. Church 1940]
- Tipos implícitos [H. Curry 1958]
- Lenguajes libres de tipos: LISP [J. McCarthy 1956-9]
- Lenguajes tipados: Fortran, Algol,...
- Lenguajes con tipos *à la* Curry: ML [R. Milner 1980]

# Motivación: tipos, programas, demostraciones

- $\underbrace{\text{Programas}}_{\text{programador}}$ aplicados en el tratamiento de $\underbrace{\textit{sistemas críticos}}_{\text{humanos}}$!
- $\underbrace{\text{Soluciones algorítmicas de problemas deben ser } \textit{demostradas} \text{ correctas.}}_{\text{matemático+científico de computación}}$
- $\underbrace{\text{Implementaciones (}\textit{programas}\text{) de soluciones algorítmicas deben ser correctas.}}_{\text{matemático+científico de computación}}$

Universidade de Brasília

# Motivación: tipos, programas, demostraciones

- $\underbrace{\text{Programas}}_{\text{programador}}$ aplicados en el tratamiento de $\underbrace{\textit{sistemas críticos}}_{\text{humanos}}$!
- $\underbrace{\text{Soluciones algorítmicas de problemas deben ser } \textit{demostradas} \text{ correctas.}}_{\text{matemático+científico de computación}}$
- $\underbrace{\text{Implementaciones (}\textit{programas}\text{) de soluciones algorítmicas deben ser correctas.}}_{\text{matemático+científico de computación}}$

Universidade de Brasília

# Motivación: tipos, programas, demostraciones

- $\underbrace{\text{Programas}}_{\text{programador}}$ aplicados en el tratamiento de $\underbrace{\textit{sistemas críticos}}_{\text{humanos}}$!
- $\underbrace{\text{Soluciones algorítmicas de problemas deben ser } \textit{demostradas} \text{ correctas.}}_{\text{matemático+científico de computación}}$
- $\underbrace{\text{Implementaciones (}\textit{programas}\text{) de soluciones algorítmicas deben ser correctas.}}_{\text{matemático+científico de computación}}$

Universidade de Brasília

# Motivación: tipos, programas, demostraciones

Ejemplo: cálculo del máximo común divisor *mcd*

$$\underbrace{\textbf{Teorema}\ [\text{Euclides 320-275 BC}]\forall n \geq 0, m > 0, mcd(n,m) = mcd(m, n \text{ MOD } m)}_{\text{idea}}$$

(Detalle: "$n$ MOD $m$" se computa como "$(n-m)$ MOD $m$)

$$\underbrace{\begin{array}{l}\text{procedimiento } mcd(m,n)\\ \qquad \text{si } m < n \text{ entonces } mcd(n,m)\\ \qquad \text{si no } (m \geq n)\\ \qquad\qquad mcd(m-n,n)\\ \text{Fin procedimiento}\end{array}}_{\text{algoritmo}}$$

# Motivación: tipos, programas, demostraciones

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

Prova de totalidade: Dominio $\mathbb{N}$ (Tipo de objetos)
**BI**: $mcd(0,n)$ indefinida! Defina $mcd(0,n) = n$.
**PI**: Suponga $mcd(k,n)$ bien definida para cualquier $n$ y $k < m$, con $m > 0$.
$\Rightarrow mcd(m,n)$ bien definida:
**Caso 1**: $m > n$. $mcd(m,n) = mcd(m-n,n)$ Se aplica HI solamente si $n > 0$!
Defina $mcd(m,0) = m$.
**Caso 2**: $m \leq n$. $mcd(m,n) = mcd(n,m)$ que está bien definido por HI.

# Motivación: tipos, programas, demostraciones

procedimiento $mcd(m, n)$
si $m = 0$ entonces $n$
si no $(**m > 0**)$
si $m < n$ entonces $mcd(n, m)$
si no $(**m > 0 \,\&\, m \geq n**)$
si $n = 0$ entonces $m$
si no $(**m > 0 \,\&\, n > 0 \,\&\, m \geq n**)$
$mcd(m - n, n)$
Fim procedimento

programa extraído de la especificación demostrada correcta

# Problemas en teoria de tipos:

- Verificación, inferencia y habitación de tipos
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo objeto computacional existe un "tipage" mas general

# Problemas en teoria de tipos:

- Verificación, inferencia y habitación de tipos
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo objeto computacional existe un "tipage" mas general

# Problemas en teoria de tipos:

- Verificación, inferencia y habitación de tipos
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo objeto computacional existe un "tipage" mas general

# Cálculo $\lambda$ con nombres: Sintaxis

TERMS $a ::= x \mid (a\ a) \mid \lambda x.a$

- Operadores básicos
  - $(a\ b)$ APLICACIÓN
  - $\lambda x.a$ ABSTRACCIÓN

Universidade de Brasília

# Reglas de reescritura del cálculo $\lambda$ con nombres

- $\alpha$-conversión

$$\lambda x.a \to \lambda y.[x/y]a$$

- $\beta$-contracción

$$(\lambda x.a\ b) \to [x/b]a$$

- $\eta$-contracción

$$\lambda x.(a\ x) \to a, \text{ if } x \notin FV(a)$$

Sustitución es una meta-operación! ▸ JumpEj

# Reglas de reescritura del cálculo $\lambda$ con nombres

- $\alpha$-conversión

$$\lambda x.a \rightarrow \lambda y.[x/y]a$$

- $\beta$-contracción

$$(\lambda x.a\ b) \rightarrow [x/b]a$$

- $\eta$-contracción

$$\lambda x.(a\ x) \rightarrow a,\ \text{if } x \notin FV(a)$$

Sustitución es una meta-operación! ▸ JumpEj

Universidade de Brasília

# Reglas de reescritura del cálculo $\lambda$ con nombres

- $\alpha$-conversión

$$\lambda x.a \to \lambda y.[x/y]a$$

- $\beta$-contracción

$$(\lambda x.a\ b) \to [x/b]a$$

- $\eta$-contracción

$$\lambda x.(a\ x) \to a, \text{ if } x \notin FV(a)$$

Sustitución es una meta-operación! ▸ JumpEj

Universidade de Brasília

# Ejemplos

- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda w.(\lambda y.(wzy)yw).$
- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda z.(\lambda y.(z\underline{z}y)yz)$ Erróneo
- $(\beta)$ $\quad (\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda y.(y\underline{y})$ Erróneo

  $(\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda z.(zy)$ Correcto

# Ejemplos

- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda w.(\lambda y.(wzy)yw).$
- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda z.(\lambda y.(z\underline{z}y)yz)$ Erróneo
- $(\beta)$ $\quad (\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda y.(y\underline{y})$ Erróneo

  $(\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda z.(zy)$ Correcto

# Ejemplos

- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda w.(\lambda y.(wzy)yw).$
- $(\alpha)$ $\quad \lambda x.(\lambda y.(xzy)yx) \to_\alpha \lambda z.(\lambda y.(z\underline{z}y)yz)$ Erróneo
- $(\beta)$ $\quad (\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda y.(y\underline{y})$ Erróneo

  $(\lambda x.(\lambda y.(yx))\ y) \to_\beta \lambda z.(zy)$ Correcto

# Ejemplos

$(\lambda_x.x \;\; \lambda_x.x) \rightarrow_\beta \lambda_x.x$ auto-aplicación

$(\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x)) \rightarrow_\beta (\lambda_x.(x \;\; x) \;\; \lambda_x.(x \;\; x))$ auto-reproducción

# Computaciones con el cálculo $\lambda$

• [1930s, Sec. XX] Alonzo Church (y Haskell Curry) define(n) el cálculo $\lambda$ (y la lógica combinatória) como un mecanismo algorítmico para computar funciones numéricas.
• [1935/36] Stephen Kleene e John Rosser demonstran que todas las funciones recursivas parciales pueden ser representadas en el cálculo $\lambda$.
• [1936] Alan Turing demuestra que exactamente las funciones Turing-computables pueden ser representadas en el cálculo $\lambda$.
$\Rightarrow$ Tese de Church-Turing: las funciones "computables" son las recursivas parciales.

# Computaciones con el cálculo $\lambda$

Computaciones numéricas:

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

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

## Proposición (Rosser)

$$\forall n \in \mathbb{N} \begin{cases} A_+ C_n C_m = C_{n+m}; \\ A_* C_n C_m = C_{n*m}; \\ A_{exp} C_n C_m = C_{n^m}, \ \textit{excepto para } m = 0. \end{cases}$$

# Computaciones con el cálculo $\lambda$

**Demostración del 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** Suponga vale para $m$ y cualquier $n$. Entonces,

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

Casos $A_+$ y $A_{exp}$ son similares.

Universidade de Brasília

# Computaciones con el cálculo $\lambda$

$\lambda$-implementación de operadores computacionales:

Defina: $\quad \mathrm{True} \equiv \lambda_{xy}.x$

$\mathrm{False} \equiv \lambda_{xy}.y$

$\langle M, N\rangle \equiv \lambda_z.(zMN)$, para todo par de $\lambda$-terminos $M, N$

*"If B Then M Else N"*

$$= BMN \begin{cases} \mathrm{True}MN \to_\beta (\lambda_y.M\ \ N) \to_\beta M \\ \mathrm{False}MN \to_\beta (\lambda_y.y\ \ N) \to_\beta N \end{cases}$$

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

Universidade de Brasília

# Computaciones con el cálculo $\lambda$

$\lambda$-implementación de operadores computacionales:
Iteración para uma función definida recursivamente

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

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

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

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

# Computaciones con el cálculo $\lambda$

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

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

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

Se concluye, $f$ puede ser especificada en el cálculo $\lambda$ como:

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

# Computaciones con el cálculo $\lambda$

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

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

O equivalentemente

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

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

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

Universidade de Brasília

# Cálculo $\lambda$ en notación de de Bruijn

- Inventado por Nicolaas Govert de Bruijn[dB72].
- Tiene las mismas propiedades del cálculo $\lambda$ con nombres.
- Elimina la necesidad de $\alpha$-conversión.

# Sintaxis

TERMS $\quad a ::= \underline{n} \,|\, (a\ a) \,|\, \lambda.a \qquad\qquad n \in \mathbb{N}$

## Ejemplos

$\lambda.(\lambda.(\underline{1}\ \underline{4}\ \underline{2})\ \underline{1})$

$\lambda.\underline{1} \simeq \lambda x.x \simeq \lambda y.y$

$\beta$ y $\eta$ contracción son definidas actualizando indices libres.

# Sintaxis

TERMS $\quad a ::= \underline{n} \,|\, (a\ a) \,|\, \lambda.a \qquad\qquad n \in \mathbb{N}$

## Ejemplos

$\lambda.(\lambda.(\underline{1}\ \underline{4}\ \underline{2})\ \underline{1})$

$\lambda.\underline{1} \simeq \lambda x.x \simeq \lambda y.y$

$\beta$ y $\eta$ contracción son definidas actualizando indices libres.

# Sintaxis

TERMS $\quad a ::= \underline{n} \,|\, (a\ a) \,|\, \lambda.a \qquad\qquad n \in \mathbb{N}$

## Ejemplos

$\lambda.(\lambda.(\underline{1}\ \underline{4}\ \underline{2})\ \underline{1})$

$\lambda.\underline{1} \simeq \lambda x.x \simeq \lambda y.y$

$\beta$ y $\eta$ contracción son definidas actualizando indices libres.

# Reglas del cálculo $\lambda$ *à la* de Bruijn

- $\beta$-contracción

$$(\lambda a\ b) \to \{\underline{1}\,/b\}a$$

- $\eta$-contraction(USUAL)

$$\lambda(a\ \underline{1}\,) \to b, \quad \text{if } b^{+} = a$$

- $\eta$-contraction(CONSTRUCTIVO)

$$\lambda(a\ \underline{1}\,) \to a^{-}, \quad \text{if } a^{-} \text{ is well-defined}$$

La sustitución permanece implícita!

# Cálculos de sustituciones explícitas

- Variaciones del cálculo $\lambda$ donde la sustitución es hecha explícita.
- Teoria mas cercana a las implementaciones.

# Cálculo $\lambda\sigma$ [ACCL91]

SYNTAX

$$\begin{array}{rl} \text{Terms} & a ::= \underline{1} \mid (a\ a) \mid \lambda.a \mid a[s] \\ \text{Substitutions} & s ::= id \mid \uparrow \mid a.s \mid s \circ s \end{array}$$

- Dos clases de objetos: Terms y Substitutions
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \longrightarrow a[b.id] \quad (Beta)$$

# Cálculo $\lambda\sigma$ [ACCL91]

SYNTAX

$$\begin{array}{rl} \text{Terms} & a ::= \underline{1} \mid (a\ a) \mid \lambda.a \mid a[s] \\ \text{Substitutions} & s ::= id \mid \uparrow \mid a.s \mid s \circ s \end{array}$$

- Dos clases de objetos: Terms y Substitutions
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \longrightarrow a[b.id] \quad (Beta)$$

# Cálculo $\lambda\sigma$ [ACCL91]

SYNTAX

$$\begin{array}{rl} \text{Terms} & a ::= \underline{1} \mid (a\ a) \mid \lambda.a \mid a[s] \\ \text{Substitutions} & s ::= id \mid \uparrow \mid a.s \mid s \circ s \end{array}$$

- Dos clases de objetos: Terms y Substitutions
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \rightarrow a[b.id] \quad (Beta)$$

# $\lambda\sigma + (Eta)$

$$
\begin{array}{lcll}
(\lambda_A.a\ b) & \longrightarrow & a[b.id] & (Beta) \\
(a\ b)[s] & \longrightarrow & (a[s]\ b[s]) & (App) \\
1[a.s] & \longrightarrow & a & (VarCons) \\
a[id] & \longrightarrow & a & (Id) \\
(\lambda_A.a)[s] & \longrightarrow & \lambda_A.(a[1.(s \circ \uparrow)]) & (Abs) \\
(a[s])[t] & \longrightarrow & a[s \circ t] & (Clos) \\
id \circ s & \longrightarrow & s & (IdL) \\
\uparrow \circ (a.s) & \longrightarrow & s & (ShiftCons) \\
(s_1 \circ s_2) \circ s_3 & \longrightarrow & s_1 \circ (s_2 \circ s_3) & (AssEnv) \\
(a.s) \circ t & \longrightarrow & a[t].(s \circ t) & (MapEnv) \\
s \circ id & \longrightarrow & s & (IdR) \\
1.\uparrow & \longrightarrow & id & (VarShift) \\
1[s].(\uparrow \circ s) & \longrightarrow & s & (Scons) \\
\lambda_A.(a\ \underline{1}) & \longrightarrow & b \quad \text{if} \quad a =_\sigma b[\uparrow] & (\text{Eta})
\end{array}
$$

Universidade de Brasília

# Cálculo $\lambda s_e$ [KR97]

SYNTAX

Terms $\quad a ::= \underline{n} \mid (a\ b) \mid \lambda_A.a \mid a\,\sigma^i\,b \mid \varphi^j_k\,a$

- Unicamente una clase de objetos: Terms (como en el cálculo $\lambda$)
- Introduce los operadores de sustitución $\sigma$ y actualización $\varphi$, guiados por restricciones aritméticas.
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \longrightarrow a\,\sigma^1\,b \quad (\sigma\text{-generation})$$

Universidade de Brasília

# Cálculo $\lambda s_e$ [KR97]

SYNTAX

Terms $\quad a ::= \underline{n} \,|\, (a\ b) \,|\, \lambda_A.a \,|\, a\,\sigma^i\,b \,|\, \varphi^j_k\,a$

- Unicamente una clase de objetos: Terms (como en el cálculo $\lambda$)
- Introduce los operadores de sustitución $\sigma$ y actualización $\varphi$, guiados por restricciones aritméticas.
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \to a\,\sigma^1\,b \quad (\sigma\text{-generation})$$

# Cálculo $\lambda s_e$ [KR97]

SYNTAX

Terms $\quad a ::= \underline{n} \mid (a\ b) \mid \lambda_A.a \mid a\,\sigma^i\,b \mid \varphi^j_k\,a$

- Unicamente una clase de objetos: Terms (como en el cálculo $\lambda$)
- Introduce los operadores de sustitución $\sigma$ y actualización $\varphi$, guiados por restricciones aritméticas.
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \longrightarrow a\,\sigma^1\,b \quad (\sigma\text{-generation})$$

Universidade de Brasília

# Cálculo $\lambda s_e$ [KR97]

SYNTAX

Terms $\quad a ::= \underline{n} \mid (a\ b) \mid \lambda_A.a \mid a\,\sigma^i\,b \mid \varphi^j_k\,a$

- Unicamente una clase de objetos: Terms (como en el cálculo $\lambda$)
- Introduce los operadores de sustitución $\sigma$ y actualización $\varphi$, guiados por restricciones aritméticas.
- La simulación de la $\beta$-reducción inicia con

$$(\lambda.a\ b) \to a\,\sigma^1\,b \quad (\sigma\text{-generation})$$

# $\lambda s_e + (Eta)$

$$
\begin{array}{lcll}
(\lambda_A.a\ \ b) & \longrightarrow & a\,\sigma^1\,b & (\sigma\text{-generation}) \\
(\lambda_A.a)\,\sigma^i\,b & \longrightarrow & \lambda_A.(a\,\sigma^{i+1}\,b) & (\sigma\text{-}\lambda\text{-transition}) \\
(a_1\ \ a_2)\,\sigma^i\,b & \longrightarrow & ((a_1\,\sigma^i\,b)\ (a_2\,\sigma^i\,b)) & (\sigma\text{-app-transition}) \\
\underline{n}\,\sigma^i\,b & \longrightarrow & \begin{cases} \underline{n-1} & \text{if } n > i \\ \varphi_0^i\,b & \text{if } n = i \\ \underline{n} & \text{if } n < i \end{cases} & (\sigma\text{-destruction}) \\
\varphi_k^i\,(\lambda_A.a) & \longrightarrow & \lambda_A.(\varphi_{k+1}^i\,a) & (\varphi\text{-}\lambda\text{-transition}) \\
\varphi_k^i\,(a_1\ \ a_2) & \longrightarrow & ((\varphi_k^i\,a_1)\ (\varphi_k^i\,a_2)) & (\varphi\text{-app-transition}) \\
\varphi_k^i\,\underline{n} & \longrightarrow & \begin{cases} \underline{n+i-1} & \text{if } n > k \\ \underline{n} & \text{if } n \leq k \end{cases} & (\varphi\text{-destruction}) \\
(a_1\,\sigma^i\,a_2)\,\sigma^j\,b & \longrightarrow & (a_1\,\sigma^{j+1}\,b)\,\sigma^i\,(a_2\,\sigma^{j-i+1}\,b) \quad \text{if } i \leq j & (\sigma\text{-}\sigma\text{-transition}) \\
(\varphi_k^i\,a)\,\sigma^j\,b & \longrightarrow & \varphi_k^{i-1}\,a \quad \text{if } k < j < k+i & (\sigma\text{-}\varphi\text{-transition 1}) \\
(\varphi_k^i\,a)\,\sigma^j\,b & \longrightarrow & \varphi_k^i\,(a\,\sigma^{j-i+1}\,b) \quad \text{if } k+i \leq j & (\sigma\text{-}\varphi\text{-transition 2}) \\
\varphi_k^i\,(a\,\sigma^j\,b) & \longrightarrow & (\varphi_{k+1}^i\,a)\,\sigma^j\,(\varphi_{k+1-j}^i\,b) \quad \text{if } j \leq k+1 & (\varphi\text{-}\sigma\text{-transition}) \\
\varphi_k^i\,(\varphi_l^j\,a) & \longrightarrow & \varphi_l^j\,(\varphi_{k+1-j}^i\,a) \quad \text{if } l+j \leq k & (\varphi\text{-}\varphi\text{-transition 1}) \\
\varphi_k^i\,(\varphi_l^j\,a) & \longrightarrow & \varphi_l^{j+i-1}\,a \quad \text{if } l \leq k < l+j & (\varphi\text{-}\varphi\text{-transition 2}) \\
\lambda_A.(a\ \underline{1}) & \longrightarrow & b \quad \text{if } a =_{s_e} \varphi_0^2\,b & (\text{Eta})
\end{array}
$$

Universidade de Brasília

# Ejemplos: SUBSEXPL

SUBSEXPL permite visualizar reducciones con diversos cálculos de SE [dMARK06].

# Tipos simples

SINTAXIS

$$\begin{array}{rl} \text{TYPES} & A ::= K \mid A \to B \\ \text{TERMS} & a ::= x \mid (a\ a) \mid \lambda x{:}B.a \end{array}$$

- Un término $\lambda$ $a$ tiene tipo $B$, denotado $a : B$
- Contexto $\Gamma = \{x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n\}$
- Un término $\lambda$ $a$ tiene tipo $B$ en el contexto $\Gamma$

$$\underbrace{\Gamma \vdash a : B}_{\text{Juicio de Tipos}}$$

Universidade de Brasília

# Tipos simples

SINTAXIS

$$\begin{array}{rl} \text{TYPES} & A ::= K \mid A \to B \\ \text{TERMS} & a ::= x \mid (a\ a) \mid \lambda x{:}B.a \end{array}$$

- Un término $\lambda$ $a$ tiene tipo $B$, denotado $a : B$
- Contexto $\Gamma = \{x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n\}$
- Un término $\lambda$ $a$ tiene tipo $B$ en el contexto $\Gamma$

$$\underbrace{\Gamma \vdash a : B}_{\text{Juicio de Tipos}}$$

# Tipos simples

SINTAXIS

$$\begin{array}{rl} \text{TYPES} & A ::= K \mid A \to B \\ \text{TERMS} & a ::= x \mid (a\ a) \mid \lambda x{:}B.a \end{array}$$

- Un término $\lambda$ $a$ tiene tipo $B$, denotado $a : B$
- Contexto $\Gamma = \{x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n\}$
- Un término $\lambda$ $a$ tiene tipo $B$ en el contexto $\Gamma$

$$\underbrace{\Gamma \vdash a : B}$$

Juicio de Tipos

# Tipos simples

SINTAXIS

$$\begin{array}{rl} \text{TYPES} & A ::= K \mid A \to B \\ \text{TERMS} & a ::= x \mid (a\ a) \mid \lambda x{:}B.a \end{array}$$

- Un término $\lambda$ $a$ tiene tipo $B$, denotado $a : B$
- Contexto $\Gamma = \{x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n\}$
- Un término $\lambda$ $a$ tiene tipo $B$ en el contexto $\Gamma$

$$\underbrace{\Gamma \vdash a : B}_{\text{Juicio de Tipos}}$$

Universidade de Brasília

# Tipos simples

$$\underbrace{\text{Ejemplos}\begin{cases}(\lambda_x.x\ \ \lambda_x.x) \to_\beta \lambda_x.x & \text{auto-aplicación}\\ (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) \to_\beta (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) & \text{auto-reproducción}\end{cases}}_{\text{Argumentación paradójica}}$$

Auto-aplicación tiene sentido:

$$(\overbrace{\lambda_{x:A\to A.x}}^{(A\to A)\to A\to A}\ \overbrace{\lambda_{x:A}.x}^{A\to A}) \to_\beta \overbrace{\lambda_{x:A}.x}^{A\to A}$$

Polimorfismo!

# Tipos simples

$$\underbrace{\text{Ejemplos}\begin{cases}(\lambda_x.x\ \ \lambda_x.x) \to_\beta \lambda_x.x & \text{auto-aplicación}\\ (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) \to_\beta (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) & \text{auto-reproducción}\end{cases}}_{\text{Argumentación paradójica}}$$

Auto-reproducción no tiene sentido:

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

Término aceptable, pero no tipable!

Universidade de Brasília

# $TA_\lambda$, el sistema de tipos simples del cálculo $\lambda$

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

$$\frac{x{:}A, \Gamma \vdash a : B}{\Gamma \vdash \lambda_{x:A}.a : A \to B}(Abs) \qquad \frac{\Gamma \vdash a : B \to A \quad \Gamma \vdash b : B}{\Gamma \vdash (a\ b) : A}(App)$$

Cuadro: $TA_\lambda$

# Ejemplo: inferencia de tipos (auto-aplicación)

## Ejemplo (Inferencia de tipos (auto-aplicación))

$$\frac{\dfrac{x{:}A \vdash x : A\,(Start)}{\vdash \lambda_{x:A}.x : A \to A}(Abs) \quad \dfrac{x{:}A \to A \vdash x : A \to A\,(Start)}{\vdash \lambda_{x:A\to A}.x : (A \to A) \to (A \to A)}(Abs)}{\Gamma \vdash (\lambda_{x:A\to A}.x\ \lambda_{x:A}.x) : A \to A}(App)$$

# Revisitando problemas en teoria de tipos

- Verificación: dados $M$ y $A$ determine si existe $\Gamma$ tal que $\Gamma \vdash M : A$.
- Inferencia: dado $M$ determine $\Gamma$ y $A$ tales que $\Gamma \vdash M : A$.
- Habitación: dado el tipo $A$. Existem *habitantes* en el contexto $\Gamma$ si, y solamente si existe un término $\lambda$ $M$ tal que $\Gamma \vdash M : A$.
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo término $M$ exite un "tipage" *mas general* $(\Gamma, A)$, tal que $\Gamma \vdash M : A$.

Universidade de Brasília

# Revisitando problemas en teoria de tipos

- Verificación: dados $M$ y $A$ determine si existe $\Gamma$ tal que $\Gamma \vdash M : A$.
- Inferencia: dado $M$ determine $\Gamma$ y $A$ tales que $\Gamma \vdash M : A$.
- Habitación: dado el tipo $A$. Existem *habitantes* en el contexto $\Gamma$ si, y solamente si existe un término $\lambda$ $M$ tal que $\Gamma \vdash M : A$.
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo término $M$ exite un "tipage" *mas general* $(\Gamma, A)$, tal que $\Gamma \vdash M : A$.

Universidade de Brasília

# Revisitando problemas en teoria de tipos

- Verificación: dados $M$ y $A$ determine si existe $\Gamma$ tal que $\Gamma \vdash M : A$.
- Inferencia: dado $M$ determine $\Gamma$ y $A$ tales que $\Gamma \vdash M : A$.
- Habitación: dado el tipo $A$. Existem *habitantes* en el contexto $\Gamma$ si, y solamente si existe un término $\lambda$ $M$ tal que $\Gamma \vdash M : A$.
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo término $M$ exite un "tipage" *mas general* $(\Gamma, A)$, tal que $\Gamma \vdash M : A$.

Universidade de Brasília

# Revisitando problemas en teoria de tipos

- Verificación: dados $M$ y $A$ determine si existe $\Gamma$ tal que $\Gamma \vdash M : A$.
- Inferencia: dado $M$ determine $\Gamma$ y $A$ tales que $\Gamma \vdash M : A$.
- Habitación: dado el tipo $A$. Existem *habitantes* en el contexto $\Gamma$ si, y solamente si existe un término $\lambda$ $M$ tal que $\Gamma \vdash M : A$.
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo término $M$ exite un "tipage" *mas general* $(\Gamma, A)$, tal que $\Gamma \vdash M : A$.

Universidade de Brasília

# Revisitando problemas en teoria de tipos

- Verificación: dados $M$ y $A$ determine si existe $\Gamma$ tal que $\Gamma \vdash M : A$.
- Inferencia: dado $M$ determine $\Gamma$ y $A$ tales que $\Gamma \vdash M : A$.
- Habitación: dado el tipo $A$. Existem *habitantes* en el contexto $\Gamma$ si, y solamente si existe un término $\lambda$ $M$ tal que $\Gamma \vdash M : A$.
- Reducción del sujeto (subject reduction): todas las computaciones preservan los tipos
- "Tipage" principal (principal typing): para todo término $M$ exite un "tipage" *mas general* $(\Gamma, A)$, tal que $\Gamma \vdash M : A$.

Universidade de Brasília

# Revisitando problemas en teoria de tipos

$$\underbrace{\Gamma}_{\text{declaraciones de variables}} \vdash \underbrace{M}_{\lambda\text{-término o programa}} : \underbrace{A}_{\text{tipos}}$$

- Verificación de tipos: los tipos designados para el programa son corretos.
- Inferencia de tipos: el programa es correto.
- Existencia de habitantes: extracción de um programa de uma prova.

## Demostraciones y Programas: el isomorfismo de Curry-Howard

Relación entre demostraciones y programas detectada por Haskell Curry [1934-1942], pero unicamente aplicada hasta 1960s por N.G. de Bruijn y William Howard.

Teoria de Tipos versus Lógica instuicionista

Luitzen Egbertus Jan Brouwer [1920]

Reglas de *tipagem* del cálculo $\lambda$ con tipos simples corresponden 1-1 a las reglas de deducción de la lógica intuicionista minimal: reglas de *tipagem* son reglas lógicas decoradas con $\lambda$-términos tipados.

# Demostraciones y Programas: el isomorfismo de Curry-Howard

## Lógica intuicionista implicacional

**Fórmulas implicacionais** son construidas de *variables proposicionales* (denotadas por $A, B, C, \ldots$) usando o conectivo implicacional $\longrightarrow$:

Asi, si $\sigma$ y $\tau$ son fórmulas implicacionais, entonces $(\sigma \longrightarrow \tau)$ lo es.

## Demostraciones y Programas: el isomorfismo de Curry-Howard

Un juicio en la lógica intuicionista $\boxed{\Omega \vdash_I A}$ denota que "$A$ es una consecuencia lógica de $\Omega$".

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

Reglas de deducción de la lógica intuicionista minimal

Una fórmula $A$ es una *tautologia* si, y solamente si el juicio $\vdash_I A$ es demostrable.

# Demostraciones y Programas: el isomorfismo de Curry-Howard

## Ejemplo ($A \to ((A \to B) \to B)$ es una tautologia)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, A \to B \vdash_I A \to B}(Axiom) \quad \dfrac{}{A, A \to B \vdash_I A}(Axiom)}{A, A \to B \vdash_I B}(Elim)}{A \vdash_I (A \to B) \to B}(Intro)}{\vdash_I A \to ((A \to B) \to A)}(Intro)$$

*En el contexto del cálculo $\lambda$ tenemos:*

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

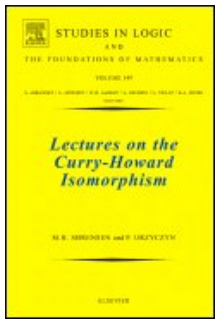
## Demostraciones y Programas: el isomorfismo de Curry-Howard

Ejemplo. **Lei de Peirce**: (PL) $((A \to B) \to A) \to A$
Vale en la lógica clásica, pero no es válida en la intuicionista!

# Demostraciones y Programas: el isomorfismo de Curry-Howard

## Isomorfismo (Curry-Howard)

*$\Omega \vdash_I A$ es demostrable en la lógica intuicionista minimal si, y solamente si $\Gamma \vdash M : A$ es un juicio de tipos válido en el cáculo $\lambda$ con tipos simples, donde $\Gamma$ es una lista de declaraciones de variables de proposiciones, observadas como tipos en $\Omega$. El término $M$ es un término $\lambda$ que representa la derivación de la demostración.*

Referencias: [Hin97], 

, [Sim00], ...

Universidade de Brasília

# $TA_{dB}$: tipos simples en $\lambda$ à la de Bruijn

$$A.\Gamma \vdash \underline{1} : A \ (var) \qquad \frac{\Gamma \vdash \underline{n} : B}{A.\Gamma \vdash \underline{n+1} : B} \ (varn)$$

$$\frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda_A.b : A \to B} \ (lambda) \qquad \frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B} \ (app)$$

Cuadro: $TA_{dB}$

# Examples

- $$\frac{A \vdash \underline{1} : A \ (var)}{\vdash \lambda_A.\underline{1} : A \to A}(lambda)$$

- $\lambda.(\underline{1}\ \underline{1})$ is not typable

# Examples

- $\dfrac{A \vdash \underline{1} : A \ (var)}{\vdash \lambda_A.\underline{1} : A \to A}(lambda)$

- $\lambda.(\underline{1}\ \underline{1})$ is not typable

# $TA_{\lambda\sigma}$: sistema de tipos simples en $\lambda\sigma$

## TERMS

$$A.\Gamma \vdash \underline{1} : A \ (var) \qquad \frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda_A.b : A \to B} \ (lambda)$$

$$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B} \ (app) \qquad \frac{\Gamma \vdash s \rhd \Gamma' \qquad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A} \ (clos)$$

## SUBSTITUTIONS

$$\Gamma \vdash id \rhd \Gamma \ (id) \qquad A.\Gamma \vdash \uparrow \rhd \Gamma \ (shift)$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash s \rhd \Gamma'}{\Gamma \vdash a.s \rhd A.\Gamma'} \ (cons) \qquad \frac{\Gamma \vdash s'' \rhd \Gamma'' \quad \Gamma'' \vdash s' \rhd \Gamma'}{\Gamma \vdash s' \circ s'' \rhd \Gamma'} \ (comp)$$

Cuadro: $TA_{\lambda\sigma}$

Universidade de Brasília

# $TA_{\lambda s_e}$: sistema de tipos simples en $\lambda s_e$

$$A.\Gamma \vdash \underline{1} : A \ (Var) \qquad \frac{\Gamma \vdash \underline{n} : B}{A.\Gamma \vdash \underline{n+1} : B} \ (Varn)$$

$$\frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda_A.b : A \to B} \ (Lambda) \qquad \frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\, b) : B} \ (App)$$

$$\frac{\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi^i_k\, a : A} \ (Phi) \qquad \frac{\Gamma_{\geq i} \vdash b : B \quad \Gamma_{<i}.B.\Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\, \sigma^i\, b : A} \ (Sigma)$$

Cuadro: $TA_{\lambda s_e}$

Universidade de Brasília

# Reducción del sujeto para $\lambda\sigma$ y $\lambda s_e$

- SR vale para $\lambda\sigma$[ACCL91] y $\lambda s_e$ [KR97]
- Pero sin restricciones existen problemas computacionales para
  - $\lambda\sigma + (Eta)$ [Har92] y
  - $\lambda s_e + (Eta)$ [ARK01]

Originalmente (Eta) restringida para $\sigma$- o $s_e$-formas normales, pero sin esas restricciones SR no vale [LARK06].

Salto 1

# Reducción del sujeto para $\lambda\sigma$ y $\lambda s_e$

**Teorema (Subjec reduction for $\lambda s_e$)**

*If* $\Gamma \vdash_{TA_{\lambda s_e}} a : A$ *and* $a \to_{\lambda s_e} a'$, *then* $\Gamma \vdash_{TA_{\lambda s_e}} a' : A$.

**Teorema (Subjec reduction for $\lambda\sigma$)**

*If* $\Gamma \vdash_{TA_{\lambda\sigma}} a : A$ *and* $a \to_{\lambda\sigma} a'$, *then* $\Gamma \vdash_{TA_{\lambda\sigma}} a' : A$.

# Exemplification of the computational problem in $\lambda s_e + (Eta)$

$$\lambda_A.(a\ \underline{1}) \to b \quad \text{if} \quad a =_{s_e} \varphi_0^2\, b \quad (Eta)$$

$$\underline{2} =_{s_e} \varphi_0^2\,(\underline{1}\,\sigma^3\lambda(\underline{1}\ \underline{1})).$$

Consequently

$$\lambda(\underline{2}\ \underline{1}) \longrightarrow_\eta \underline{1}\,\sigma^3\lambda(\underline{1}\ \underline{1})$$

# Exemplification of the computational problem in $\lambda\sigma + (Eta)$

$\lambda(\underline{2}\ \underline{1}) \longrightarrow_\eta \underline{2}\,[\lambda(\underline{1}\ \underline{1}).\underline{1}\,.\uparrow]$, since $\underline{2} =_\sigma \left(\underline{2}\,[\lambda(\underline{1}\ \underline{1}).\underline{1}\,.\uparrow]\right)[\uparrow]$:

$$
\begin{array}{l}
(\underline{1}\,[\uparrow])[\lambda(\underline{1}\ \underline{1}).\underline{1}\,.\uparrow] \longrightarrow_{Clos} \\
\underline{1}\,[\uparrow \circ (\lambda(\underline{1}\ \underline{1}).\underline{1}\,.\uparrow)] \longrightarrow_{ShiftCons} \\
\underline{1}\,[\underline{1}\,.\uparrow] \longrightarrow_{VarCons} \\
\underline{1}
\end{array}
$$

But also $\lambda(\underline{2}\ \underline{1}) \longrightarrow_\eta \underline{1}$ may be inferred through ill-typed computations!

# Resultado principal

Formulación de nuevos cálculos que

- implementan $\eta$-reducción explicitamente y
- irrestrictamente preservan SR

Salto 2

# The system $R_{\eta\lambda\sigma}$

| | | | |
|---|---|---|---|
| $(a\ b)[s]$ | $\longrightarrow$ | $(a[s]\ b[s])$ if $\diamond$ occurs in $s$ | ($\eta$-App) |
| $\underline{1}\,[a.s]$ | $\longrightarrow$ | $a$ if $\diamond$ occurs in $s$ | ($\eta$-VarCons) |
| $(\lambda_A.a)[s]$ | $\longrightarrow$ | $\lambda_A.(a[\underline{1}\,.(s\circ\uparrow)])$ if $\diamond$ occurs in $s$ | ($\eta$-Abs) |
| $(a[s])[t]$ | $\longrightarrow$ | $a[s\circ t]$ if $\diamond$ occurs in $t$ | ($\eta$-Clos) |
| $(s_1\circ s_2)\circ t$ | $\longrightarrow$ | $s_1\circ(s_2\circ t)$ if $\diamond$ occurs in $t$ | ($\eta$-AssEnv) |
| $(a.s)\circ t$ | $\longrightarrow$ | $a[t].(s\circ t)$ if $\diamond$ occurs in $(a.s)\circ t$ | ($\eta$-MapEnv) |
| $\uparrow\circ(a.s)$ | $\longrightarrow$ | $s$ if $\diamond$ occurs in $a.s$ | ($\eta$-ShiftCons) |
| $\diamond[\uparrow]$ | $\longrightarrow$ | $\diamond$ | ($\eta$-Cons) |
| $id\circ s$ | $\longrightarrow$ | $s$ if $\diamond$ occurs in $s$ | ($\eta$-IdL) |
| $\underline{1}\,[\diamond.s]$ | $\longrightarrow$ | $\diamond$ | (Error) |

Convergence of $R_{\eta\lambda\sigma}$ holds: normalisation with $R_{\eta\lambda\sigma}$ simply propagates the symbol $\diamond$ between the finite structure of the terms and joinability of all critical pairs can be checked.

Universidade de Brasília

# (Eta) explícita para $\lambda\sigma$

$$\lambda_A.(a\ \underline{1}) \to N_{R_{\eta_{\lambda\sigma}}}(a[\diamond.id]), \quad \text{si } N_{R_{\eta_{\lambda\sigma}}}(a[\diamond.id]) \text{ es un } \lambda\sigma\text{-término} \quad \text{(Eta)}$$

**Teorema (RS para (Eta))**

*Se* $\Gamma \vdash \lambda_B.(a\ \underline{1}) : A$ *y* $\lambda_B.(a\ \underline{1}) \to_{Eta} b$, *entonces* $\Gamma \vdash b : A$

Salto 3

Universidade de Brasília

# (Eta) explícita para $\lambda\sigma$

$$\lambda_A.(a\ \underline{1}) \to N_{R_{\eta_{\lambda\sigma}}}(a[\diamond.id]), \quad \text{si } N_{R_{\eta_{\lambda\sigma}}}(a[\diamond.id]) \text{ es un } \lambda\sigma\text{-término} \quad \text{(Eta)}$$

**Teorema (RS para (Eta))**

*Se* $\Gamma \vdash \lambda_B.(a\ \underline{1}) : A$ *y* $\lambda_B.(a\ \underline{1}) \to_{Eta} b$, *entonces* $\Gamma \vdash b : A$

▸ Salto 3

Universidade de Brasília

# Computational improvment of $R_{\eta_{\lambda\sigma}}$

$$
\begin{array}{lcll}
(a[s])[t] & \longrightarrow & \begin{cases} a[s \circ t] & \text{if } \diamond \text{ occurs in } t \text{ and } a = \underline{1} \\ a[N_{R_{\eta_{\lambda\sigma}}}(s \circ t)] & \text{if } \diamond \text{ occurs in } t \text{ and } N_{R_{\eta_{\lambda\sigma}}}(s \circ t) \in \Lambda_\sigma \\ \mathit{error} & \text{if } \diamond \text{ occurs in } t \text{ and } N_{R_{\eta_{\lambda\sigma}}}(s \circ t) \notin \Lambda_\sigma \end{cases} & (\eta\text{-Clos}) \\
\mathit{id} \circ s & \longrightarrow & \mathit{error} \quad \text{if } \diamond \text{ occurs in } s & (\eta\text{-IdL}) \\
\underline{1}\,[\diamond.s] & \longrightarrow & \mathit{error} & (\text{Error})
\end{array}
$$

# The system $R_{\eta s_e}$

$$
\begin{array}{lcll}
(a\ b)\,\eta^i & \longrightarrow & (a\,\eta^i\ \ b\,\eta^i) & (\eta\text{-app-transition}) \\
(\lambda_A.a)\,\eta^i & \longrightarrow & \lambda_A.a\,\eta^{i+1} & (\eta\text{-}\lambda\text{-transition}) \\
\underline{n}\,\eta^i & \longrightarrow & \begin{cases} \underline{n} & \text{if } n < i \\ \underline{n-1} & \text{if } n > i \end{cases} & (\eta\text{-destruction}) \\
(a\,\sigma^j\,b)\,\eta^i & \longrightarrow & (a\,\eta^i)\,\sigma^{j-1}\,b \quad \text{if } i < j & (\eta\text{-}\sigma\text{-transition 1}) \\
(a\,\sigma^j\,b)\,\eta^i & \longrightarrow & (a\,\eta^{i+1})\,\sigma^j(b\,\eta^{i-j+1}) \quad \text{if } i \geq j & (\eta\text{-}\sigma\text{-transition 2}) \\
(\varphi^j_k\ a)\,\eta^i & \longrightarrow & \varphi^j_{k-1}\,(a\,\eta^i) \quad \text{if } i \leq k & (\eta\text{-}\varphi\text{-transition 1}) \\
(\varphi^j_k\ a)\,\eta^i & \longrightarrow & \varphi^{j-1}_k\,a \quad \text{if } k < i < k+j & (\eta\text{-}\varphi\text{-transition 2}) \\
(\varphi^j_k\ a)\,\eta^i & \longrightarrow & \varphi^j_k\,(a\,\eta^{i-j+1}) \quad \text{if } i > k \text{ and } i \geq k+j & (\eta\text{-}\varphi\text{-transition 3})
\end{array}
$$

$R_{\eta s_e}$ is convergent: normalisation of $a\eta$ is simply a propagation of the symbol $\eta$ between the finite structure of $a$ and $R_{\eta s_e}$ is orthogonal.

Universidade de Brasília

# Explicit (Eta) for $\lambda s_e$

$$\lambda_A.(a\ \underline{1}) \to N_{R_{\eta s_e}}(a\,\eta^1), \quad \text{si } N_{R_{\eta s_e}}(a\,\eta^1) \text{ es un } \lambda s_e\text{-termino.} \qquad \text{(Eta)}$$

**Teorema (SR para (Eta))**

*Si* $\Gamma \vdash \lambda_B.(a\ \underline{1}) : A$ *y* $\lambda_B.(a\ \underline{1}) \to_{Eta} b$, *entonces* $\Gamma \vdash b : A$

Salto 4

Universidade de Brasília

# Explicit (Eta) for $\lambda s_e$

$$\lambda_A.(a\ \underline{1}) \to N_{R_{\eta s_e}}(a\,\eta^1), \quad \text{si } N_{R_{\eta s_e}}(a\,\eta^1) \text{ es un } \lambda s_e\text{-termino.} \qquad \text{(Eta)}$$

**Teorema (SR para (Eta))**

*Si* $\Gamma \vdash \lambda_B.(a\ \underline{1}) : A$ *y* $\lambda_B.(a\ \underline{1}) \to_{Eta} b$, *entonces* $\Gamma \vdash b : A$

▸ Salto 4

Universidade de Brasília

# Computational improvment of $R_{\eta s_e}$

$$\underline{n}\,\eta^i \longrightarrow \begin{cases} \underline{n} & \text{if } n < i \\ \mathit{error} & \text{if } n = i \\ \underline{n-1} & \text{if } n > i \end{cases} \qquad (\eta\text{-destruction2})$$

# Tipage Principal

## Definición (Tipage Principal [Wel02])

*Un tipage $\tau = (\Gamma, B)$ es* **principal** *en un sistema de tipos TA para un termino a si, y solamente si $TA \rhd a : \tau$ y para todo $\tau'$, si $TA \rhd a : \tau'$ entonces $\tau \leq_{TA} \tau'$ (i.e., $Terms(\tau) \subseteq Terms(\tau')$)*

## Ejemplo

*Para $TA_\lambda$, $(\varnothing, A \to A) \leq_{TA} (x{:}A, A \to A)$.*

*$(\varnothing, A \to A)$ es tipage principal de $\lambda x{:}A.x$*

# Tipage Principal

## Definición (Tipage Principal [Wel02])

*Un tipage $\tau = (\Gamma, B)$ es* **principal** *en un sistema de tipos TA para un termino a si, y solamente si $TA \rhd a : \tau$ y para todo $\tau'$, si $TA \rhd a : \tau'$ entonces $\tau \leq_{TA} \tau'$ (i.e., $Terms(\tau) \subseteq Terms(\tau')$)*

## Ejemplo

*Para $TA_\lambda$, $(\varnothing, A \to A) \leq_{TA} (x{:}A, A \to A)$.*

*$(\varnothing, A \to A)$ es tipage principal de $\lambda x{:}A.x$*

# Tipage Principal

## Definición (Tipage Principal [Wel02])

*Un tipage $\tau = (\Gamma, B)$ es* **principal** *en un sistema de tipos TA para un termino a si, y solamente si $TA \rhd a : \tau$ y para todo $\tau'$, si $TA \rhd a : \tau'$ entonces $\tau \leq_{TA} \tau'$ (i.e., $Terms(\tau) \subseteq Terms(\tau')$)*

## Ejemplo

*Para $TA_\lambda$, $(\varnothing, A \to A) \leq_{TA} (x{:}A, A \to A)$.*

*$(\varnothing, A \to A)$ es tipage principal de $\lambda x{:}A.x$*

# Tipage Principal

## Definición

*Un* **tipage principal** *en* $TA_\lambda$ *de un término* $a$ *es un tipage* $\tau = (\Gamma, B)$ *tal que*

1. $TA_\lambda \rhd a : \tau$
2. *Si* $TA_\lambda \rhd a : \tau'$ *para algun tipage* $\tau' = (\Gamma', B')$, *entonces exite una sustitución* $s$ *tal que* $s(\Gamma) \subseteq \Gamma'$ *y* $s(B) = B'$.

# Tipage principal para $\lambda\sigma$

## Definición

*Un* **tipage principal** *en* $TA_{\lambda\sigma}$ *de una expresión* $a$ *es una tipage* $\tau = (\Gamma, \mathbb{T})$ *tal que*

1. $TA_{\lambda\sigma} \rhd a : \tau$
2. *Si* $TA_{\lambda\sigma} \rhd a : \tau'$ *para alguna tipage* $\tau' = (\Gamma', \mathbb{T}')$, *entonces existe una sustitución* $s$ *tal que* $s(\Gamma) = \Gamma'_{\leq|\Gamma|}$ *y* $s(\mathbb{T}) = \mathbb{T}'$.

## Teorema (Tipage principal para $TA_{\lambda\sigma}$)

*El sistema de tipos* $TA_{\lambda\sigma}$ *admite tipages principales.*

Universidade de Brasília

# Tipage principal para $\lambda\sigma$

## Definición

*Un* **tipage principal** *en* $TA_{\lambda\sigma}$ *de una expresión* $a$ *es una tipage* $\tau = (\Gamma, \mathbb{T})$ *tal que*

1. $TA_{\lambda\sigma} \rhd a : \tau$
2. *Si* $TA_{\lambda\sigma} \rhd a : \tau'$ *para alguna tipage* $\tau' = (\Gamma', \mathbb{T}')$, *entonces existe una sustitución* $s$ *tal que* $s(\Gamma) = \Gamma'_{\leq|\Gamma|}$ *y* $s(\mathbb{T}) = \mathbb{T}'$.

## Teorema (Tipage principal para $TA_{\lambda\sigma}$)

*El sistema de tipos* $TA_{\lambda\sigma}$ *admite tipages principales.*

Universidade de Brasília

# Tipage principal para $\lambda s_e$

## Definición

*Un* **tipage principal** *en* $TA_{\lambda s_e}$ *para un término* $a$ *es un tipage* $\tau = (\Gamma, B)$ *tal que*

1. $TA_{\lambda s_e} \rhd a : \tau$
2. *Si* $TA_{\lambda s_e} \rhd a : \tau'$ *para algun tipage* $\tau' = (\Gamma', B')$, *entonces existe una sustitución* $s$ *tal que* $s(\Gamma) = \Gamma'_{\leq |\Gamma|}$ *y* $s(B) = B'$.

## Teorema (Tipage principal para $TA_{\lambda s_e}$)

*El sistema de tipos* $TA_{\lambda s_e}$ *admite tipages principales.*

Universidade de Brasília

# Tipage principal para $\lambda s_e$

## Definición

*Un* **tipage principal** *en* $TA_{\lambda s_e}$ *para un término* $a$ *es un tipage* $\tau = (\Gamma, B)$ *tal que*

1. $TA_{\lambda s_e} \rhd a : \tau$
2. *Si* $TA_{\lambda s_e} \rhd a : \tau'$ *para algun tipage* $\tau' = (\Gamma', B')$, *entonces existe una sustitución* $s$ *tal que* $s(\Gamma) = \Gamma'_{\leq |\Gamma|}$ *y* $s(B) = B'$.

## Teorema (Tipage principal para $TA_{\lambda s_e}$)

*El sistema de tipos* $TA_{\lambda s_e}$ *admite tipages principales.*

Universidade de Brasília

# Ejemplo

$$a = \lambda_{C\to A}.((\underline{1}\,\sigma^2\underline{2})\ (\varphi_0^2\,\underline{2}))$$

$$a' = (\lambda_{C\to A}.((\underline{1}^{\Gamma_1}_{A_1}\,\sigma^2\underline{2}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3}\ (\varphi_0^2\,\underline{2}^{\Gamma_4}_{A_4})^{\Gamma_5}_{A_5})^{\Gamma_6}_{A_6})^{\Gamma_7}_{A_7}$$

Se aplican las reglas de inferencia para el par $\langle R_0, \varnothing\rangle$

# Ejemplo

$$a = \lambda_{C\to A}.((\underline{1}\,\sigma^2\underline{2})\ (\varphi_0^2\,\underline{2}))$$

$$a' = (\lambda_{C\to A}.((\underline{1}^{\Gamma_1}_{A_1}\,\sigma^2\underline{2}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3}\ (\varphi_0^2\,\underline{2}^{\Gamma_4}_{A_4})^{\Gamma_5}_{A_5})^{\Gamma_6}_{A_6})^{\Gamma_7}_{A_7}$$

Se aplican las reglas de inferencia para el par $\langle R_0, \varnothing\rangle$

# Ejemplo

$$a = \lambda_{C\to A}.((\underline{1}\,\sigma^2\underline{2})\ (\varphi_0^2\,\underline{2}))$$

$$a' = (\lambda_{C\to A}.((\underline{1}^{\Gamma_1}_{A_1}\,\sigma^2\underline{2}^{\Gamma_2}_{A_2})^{\Gamma_3}_{A_3}\ (\varphi_0^2\,\underline{2}^{\Gamma_4}_{A_4})^{\Gamma_5}_{A_5})^{\Gamma_6}_{A_6})^{\Gamma_7}_{A_7}$$

Se aplican las reglas de inferencia para el par $\langle R_0, \varnothing\rangle$

# Ejemplo

1° step

$\langle R_0, \varnothing \rangle \vdash \langle R_1 = R_0 \setminus \{\underline{1}^{\Gamma_1}_{A_1}\}, E_1 = \{\Gamma_1 = A_1.\Gamma'_1\}\rangle(\text{Var})$ ▸ regra

2° step

$\langle R_1, E_1 \rangle \vdash \langle R_2 = R_1 \setminus \{\underline{2}^{\Gamma_2}_{A_2}\}, E_2 = E_1 \cup \{\Gamma_2 = A'_1.A_2.\Gamma'_2\}\rangle(\text{Varn})$

El processo continua hasta alcanzar el par $\langle \varnothing, E_7 \rangle$, donde

$$E_7 = \{\Gamma_1 = A_1.\Gamma'_1, \Gamma_2 = A'_1.A_2.\Gamma'_2, A_1 = A_3, \Gamma_1 = A'_2.A_2.\Gamma_2, \Gamma_3 = A'_2.\Gamma_2,$$
$$\Gamma_4 = A'_3.A_4.\Gamma'_3, A_4 = A_5, \Gamma_5 = A'_4.\Gamma'_4, \Gamma_4 = \Gamma'_4, \Gamma_3 = \Gamma_5, \Gamma_5 = \Gamma_6,$$
$$A_3 = A_5 \to A_6, A_7 = (C \to A) \to A_6, \Gamma_6 = C \to A.\Gamma_7\}$$

Universidade de Brasília

# Ejemplo

1° step

$\langle R_0, \varnothing\rangle \vdash \langle R_1 = R_0 \setminus \{\underline{1}^{\Gamma_1}_{A_1}\}, E_1 = \{\Gamma_1 = A_1.\Gamma_1'\}\rangle(\text{Var})$ ▸ regra

2° step

$\langle R_1, E_1\rangle \vdash \langle R_2 = R_1 \setminus \{\underline{2}^{\Gamma_2}_{A_2}\}, E_2 = E_1 \cup \{\Gamma_2 = A_1'.A_2.\Gamma_2'\}\rangle(\text{Varn})$

El processo continua hasta alcanzar el par $\langle \varnothing, E_7\rangle$, donde

$$E_7 = \{\Gamma_1 = A_1.\Gamma_1', \Gamma_2 = A_1'.A_2.\Gamma_2', A_1 = A_3, \Gamma_1 = A_2'.A_2.\Gamma_2, \Gamma_3 = A_2'.\Gamma_2,$$
$$\Gamma_4 = A_3'.A_4.\Gamma_3', A_4 = A_5, \Gamma_5 = A_4'.\Gamma_4', \Gamma_4 = \Gamma_4', \Gamma_3 = \Gamma_5, \Gamma_5 = \Gamma_6,$$
$$A_3 = A_5 \to A_6, A_7 = (C \to A) \to A_6, \Gamma_6 = C \to A.\Gamma_7\}$$

# Ejemplo

Después de simplificaciones triviales de $E_7$ se obtiene

$$\{\Gamma_2 = A'_1.A_2.\Gamma'_2\ ,\ A'_2.\Gamma_2 = A'_4.A'_3.A_4.\Gamma'_3 = C \to A.\Gamma_7\ ,\ A_1.\Gamma'_1 = A'_2.A_2.\Gamma_2,$$
$$A_1 = A_4 \to A_6\ ,\ A_7 = (C \to A) \to A_6\}$$

De donde se tiene el *mgu*, $A_7 = (C \to A) \to A$ and $\Gamma_7 = A'_3.C.\Gamma'_3$

# Ejemplo

Después de simplificaciones triviales de $E_7$ se obtiene

$\{\Gamma_2 = A'_1.A_2.\Gamma'_2\ ,\ A'_2.\Gamma_2 = A'_4.A'_3.A_4.\Gamma'_3 = C \to A.\Gamma_7\ ,\ A_1.\Gamma'_1 = A'_2.A_2.\Gamma_2,$
$A_1 = A_4 \to A_6\ ,\ A_7 = (C \to A) \to A_6\}$

De donde se tiene el *mgu*, $A_7 = (C \to A) \to A$ and $\Gamma_7 = A'_3.C.\Gamma'_3$

# Conclusiones

- Susitución es operación escencial en computación
- Ya que se implementa concretamente, es necesario desarrollar formalismos que traten la sustitucion explícitamente
- Al hacer explícita la sustitución, preservar las propiedades teóricas del modelo original no es trivial.
- Para $TA_{\lambda\sigma}$ y $TA_{\lambda s_e}$:
  - se detectaron problemas para SR en $\lambda\sigma$ y $\lambda s_e$ y se solucionaron formalizando explicitamente la regla (ETA);
  - se verificó que la noción general de Wells de PT aplica para $\lambda\sigma$ y $\lambda s_e$.

# Retos

- Desarrollar sistemas de tipos explícitos mas complejos: recursivos, intersección, dependientes, ....
- Implementación de sistemas computacionales de orden superior utilizando algoritmos de decisión para matching y unificación [dMKAR05, ARdMK05].

Universidade de Brasília


M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy.
Explicit Substitutions.
*J. of Func. Programming*, 1(4):375–416, 1991.


M. Ayala-Rincón, F.C. de Moura, and F. Kamareddine.
Comparing and Implementing Calculi of Explicit Substitutions with Eta-Reduction.
*Annals of Pure and Applied Logic*, 134:5–41, 2005.
Special Issue WoLLIC'02, R. de Queiroz, B. Poizat and A. Artemov, Eds.


M. Ayala-Rincón and F. Kamareddine.
Unification via the $\lambda s_e$-Style of Explicit Substitution.
*The Logical Journal of the Interest Group in Pure and Applied Logics*, 9(4):489–523, 2001.


N.G. de Bruijn.
Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem.
*Indag. Mat.*, 34(5):381–392, 1972.


N.G. de Bruijn.
A namefree lambda calculus with facilities for internal definition of expressions and segments.
T.H.-Report 78-WSK-03, Technische Hogeschool Eindhoven, Nederland, 1978.


F.L.C. de Moura, M. Ayala-Rincón, and F. Kamareddine.
SUBSEXPL: a Tool for Simulation and Comparing Explicit Substitutions Calculi.
*Journal of Applied Non-Classical Logics*, 16(1-2):119–150, 2006.


F.L.C. de Moura, F. Kamareddine, and M. Ayala-Rincón.
Second-Order Matching via Explicit Substitutions.

In *11th Int. Conf. on Logic Programming Artificial Intelligence and Reasoning LPAR 2004*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 433–448. Springer Verlag, 2005.

T. Hardin.

Eta-conversion for the languages of explicit substitutions.
In *Algebraic and logic programming*, volume 632 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 1992.

J. R. Hindley.

*Basic Simple Type Theory*.
Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

F. Kamareddine, editor.

*Thirty Five Years of Automating Mathematics*.
Kluwer, 2003.

C. Kirchner and C. Ringeissen.

Higher-order Equational Unification via Explicit Substitutions.
In *Proc. Algebraic and Logic Programming*, volume 1298 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1997.

D.V. Lima, M. Ayala-Rincón, and F. Kamareddine.

Explicit Substitutions Calculi with Explicit Eta rules which Preserve Subject Reduction.
In *Proc. Workshop on Logical and Semantical Frameworks*, pages 1–15, 2006.

R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer.

*Selected papers on Automath*.
North-Holland, 1994.

H. Simmons.

Universidade de Brasília

*Derivation and Computation: taking the Curry-Howard correspondence seriously*.
Number 51 in Cambridge Tracts in Theoretical Computer Science. Cambridge, 2000.

J.B. Wells.

The essence of principal typings.
In *Proc. 29th International Colloquium on Automata, Languages and Programming, ICALP 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 913–925. Springer Verlag, 2002.

# Inference rules ($TA_{\lambda s_e}$)

(Var) $$\frac{\langle R \cup \{\underline{1}^{\Gamma}_{A}\}, E\rangle}{\langle R, E \cup \{\Gamma = A.\Gamma'\}\rangle},$$ where $\Gamma'$ is a new context variable;

(Varn) $$\frac{\langle R \cup \{\underline{n}^{\Gamma}_{A}\}, E\rangle}{\langle R, E \cup \{\Gamma = A'_1.\cdots.A'_{n-1}.A.\Gamma'\}\rangle},$$ where $\Gamma'$ and $A'_1, \ldots, A'_{n-1}$ are new variables of context and type;

voltar

# $\eta$-Rules *à la de Bruijn*

$\eta$-contraction(USUAL)

$$\lambda(a\ \underline{1}) \longrightarrow b, \quad \text{if } b^{+} = a$$

$\eta$-contraction(CONSTRUCTIVO)

$$\lambda(a\ \underline{1}) \longrightarrow a^{-}, \quad \text{if } a^{-} \text{ is well-defined}$$