El cálculo λ

Y los fundamentos de la computación

Eduardo Acuña Yeomans

14 de septiembre de 2016

Universidad de Sonora

Contenido

Introducción

¿Qué es el cálculo λ?

🤔

¿Qué es el cálculo λ?

- Un lenguaje para expresar cómputo

$$\lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

¿Qué es el cálculo λ?

🤔

- Un lenguaje para expresar cómputo

$$\lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

- Teoría matemática

$$\boldsymbol{\lambda}\mathbf{K}\boldsymbol{\beta}, \Gamma \vdash M = N$$

¿Qué es el cálculo λ?

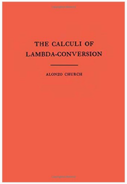
- Un lenguaje para expresar cómputo

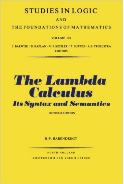
$$\lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

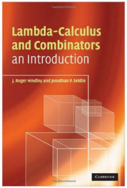
- Teoría matemática

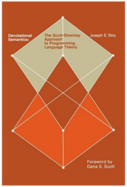
$$\boldsymbol{\lambda}\mathbf{K}\beta, \Gamma \vdash M = N$$

- Sistema para estudiar matemáticas y computación









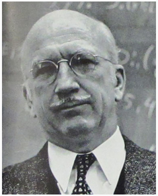
Orígen

Alonzo Church

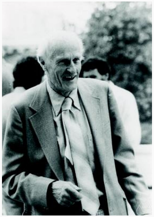
1932 Un conjunto de postulados para la fundamentación de la lógica

Orígen

Barkley Rosser



Stephen Kleene



1932 Un conjunto de postulados para la fundamentación de la lógica

1935 La insconsistencia de ciertas lógicas formales

Orígen

Alonzo Church

1932 Un conjunto de postulados para la fundamentación de la lógica

1935 La insconsistencia de ciertas lógicas formales

1936 Un problema sin solución de la teoría de números elemental

Concepto de función

$$f : A \to B$$

$$f(x) = E$$

$$f(a) = b$$

Concepto de función

$$f : A \to B$$
$$f(x) = E$$
$$f(a) = b$$

$$\downarrow$$

$$f = (\lambda x.E)$$
$$(f\ a) = b$$

Concepto de función

$$f : A \to B$$

$$f(x) = E$$

$$f(a) = b$$

$$\{(a, b) \mid a \in A,\ b \in B\}$$

$$\downarrow$$

$$f = (\lambda x.E)$$

$$(f\ a) = b$$

$$(\quad \lambda \quad x \quad . \quad E \quad)$$

Conceptos básicos

Términos λ

$$\Lambda$$

- Variables
- Abstracciones
- Aplicaciones

Términos λ

$$\Lambda$$

- Variables w, x, y, z, etc.
- Abstracciones
- Aplicaciones

Términos λ

$$M \in \Lambda$$

- Variables w, x, y, z, etc.
- Abstracciones $(\lambda x.M)$
- Aplicaciones

Términos λ

$$M,\ N \in \Lambda$$

- Variables $w,\ x,\ y,\ z,$ etc.
- Abstracciones $(\lambda x.M)$
- Aplicaciones $(M\,N)$

Términos λ

$$\Lambda$$

- Variables w, x, y, z, etc.
- Abstracciones $(\lambda x.M)$
- Aplicaciones $(M\,N)$

El alfabeto tiene como elementos los símbolos:

$$(\qquad) \qquad \lambda \qquad .$$

y a todas las variables

Términos λ

Identificando la estructura de un término λ

$$((\lambda y.(y\,z))\,y)$$

Términos λ

Las aplicaciones tienen la forma $(M\, N)$

$$((\lambda y.(y\, z))\, y)$$

Las abstracciones tienen la forma $(\lambda x.M)$

$$((\lambda y.(y\, z))\, y)$$

Las variables se clasifican por su posición en la expresión

$$((\lambda y.(y\, z))\, y)$$

- Variables vinculadas
- Variables ligadas
- Variables libres

Operaciones

Un término λ puede ser transformado a otros términos λ utilizando el concepto de sustitución.

Operaciones

Un término λ puede ser transformado a otros términos λ utilizando el concepto de sustitución.

Sustituír una variable x por un término N en un término M se denota

$$M[x := N]$$

Operaciones

Ejemplos de sustituciones

$$
\begin{aligned}
x[x := (\lambda y.y)] &\to (\lambda y.y) \\
x[z := (\lambda y.y)] &\to x \\
(x\,z)[x := (\lambda y.y)] &\to (x[x := (\lambda y.y)]\ z[x := (\lambda y.y)]) \\
(\lambda x.z)[x := (\lambda y.y)] &\to (\lambda x.z) \\
(\lambda z.x)[x := (\lambda y.y)] &\to (\lambda z.x[x := (\lambda y.y)])
\end{aligned}
$$

Operaciones

A una abstracción $(\lambda x.M)$ se le pueden cambiar sus variables ligadas a λx.

$$(\lambda x.M) \xrightarrow{y}_{\alpha} (\lambda y.M[x := y])$$

Operaciones

La aplicación de una abstracción $(\lambda x.M)$ a un término cualquiera N se puede reducir.

$$((\lambda x.M)\,N) \to_\beta M[x := N]$$

Equivalencias

La α-convertibilidad relaciona dos términos que pueden ser transformados al mismo término utilizando cambios de variable ligada.

$$(\lambda f.(\lambda x.(f\ (f\ x)))) =_\alpha (\lambda g.(\lambda y.(g\ (g\ y))))$$

Equivalencias

La β-convertibilidad relaciona dos términos que pueden ser transformados al mismo término utilizando cambios de variable ligada y reducciones.

$$(((\lambda f.(\lambda x.(x\, f)))\,(\lambda w.w))\,(\lambda z.z)) =_\beta (\lambda w.w)$$

Abusos de notación

$$(((\lambda x.(\lambda y.(x\,(y\,x))))\,a)\,b)$$

Abusos de notación

$$(((\lambda x.(\lambda y.(x\,(y\,x))))\,a)\,b)$$

- Las aplicaciones tienen asociatividad a la izquierda

$$((M_1\,M_2)\,M_3) = M_1\,M_2\,M_3$$

Abusos de notación

$$(((\lambda x.(\lambda y.(x\,(y\,x))))\,a)\,b)$$

- Las aplicaciones tienen asociatividad a la izquierda

$$((M_1\,M_2)\,M_3) = M_1\,M_2\,M_3$$

- Una abstracción cuyo cuerpo es abstracción puede agrupar los argumentos

$$(\lambda x.(\lambda y.M)) = (\lambda x\,y.M)$$

Abusos de notación

$$(((\lambda x.(\lambda y.(x\,(y\,x))))\,a)\,b)$$

- Las aplicaciones tienen asociatividad a la izquierda

$$((M_1\,M_2)\,M_3) = M_1\,M_2\,M_3$$

- Una abstracción cuyo cuerpo es abstracción puede agrupar los argumentos

$$(\lambda x.(\lambda y.M)) = (\lambda x\,y.M)$$

- Los paréntesis se omiten cuando no hay ambigüedad

$$(M\,N) = M\,N \text{ y } (\lambda x.M) = \lambda x.M$$

Abusos de notación

$$(((\lambda x.(\lambda y.(x\,(y\,x))))\,a)\,b) = (\lambda x\,y.x\,(y\,x))\,a\,b$$

- Las aplicaciones tienen asociatividad a la izquierda

$$((M_1\,M_2)\,M_3) = M_1\,M_2\,M_3$$

- Una abstracción cuyo cuerpo es abstracción puede agrupar los argumentos

$$(\lambda x.(\lambda y.M)) = (\lambda x\,y.M)$$

- Los paréntesis se omiten cuando no hay ambigüedad

$$(M\,N) = M\,N \text{ y } (\lambda x.M) = \lambda x.M$$

Cómputo

Álgebra booleana

Codificación de valores de verdad verdadero y falso.

$$\mathbf{T} = \lambda x\, y.x$$
$$\mathbf{F} = \lambda x\, y.y$$

Álgebra booleana

Codificación de operaciones

$$\textbf{if} = \lambda p\, x\, y. p\, x\, y$$

$$\begin{aligned}
\textbf{not} &= \lambda p.\textbf{if}\; p\, \textbf{F}\, \textbf{T} \\
\textbf{and} &= \lambda p\, q.\textbf{if}\; p\, (\textbf{if}\; q\, \textbf{T}\, \textbf{F})\, \textbf{F} \\
\textbf{or} &= \lambda p\, q.\textbf{if}\; p\, \textbf{T}\, (\textbf{if}\; q\, \textbf{T}\, \textbf{F})
\end{aligned}$$

Álgebra booleana

Generalización a 3 valores de verdad

$$\mathbf{V_1} = \lambda x\, y\, z.x$$
$$\mathbf{V_2} = \lambda x\, y\, z.y$$
$$\mathbf{V_3} = \lambda x\, y\, z.z$$

$$\mathbf{if_3} = \lambda p\, x\, y\, z.p\, x\, y\, z$$

Aritmética

Codificación de números naturales.

$$
\begin{aligned}
\mathbf{0} &= \lambda f\, x.x \\
\mathbf{1} &= \lambda f\, x.f\, x \\
\mathbf{2} &= \lambda f\, x.f\,(f\, x) \\
\mathbf{3} &= \lambda f\, x.f\,(f\,(f\, x)) \\
&\dots \\
\mathbf{n} &= \lambda f\, x.\underbrace{f\,(f\,(\dots(f}_{n}\; x)\dots))
\end{aligned}
$$

Aritmética

Codificación de las operaciones elementales.

$$
\begin{aligned}
\textbf{suc} &= \lambda n.\lambda f\, x.f\,(n\,f\,x) \\
\boldsymbol{+} &= \lambda m\, n.n\,\textbf{suc}\,m \\
\boldsymbol{\times} &= \lambda m\, n.n\,(\boldsymbol{+}\,m)\,\mathbf{0} \\
\boldsymbol{\uparrow} &= \lambda m\, n.n\,(\boldsymbol{\times}\,m)\,\mathbf{1}
\end{aligned}
$$

Aritmética

La codificación de los números naturales provee un mecanismo de iteración.

$$\mathbf{n} = \lambda f\, x.\, \overbrace{f\,(f\,(...(f}^{n}\; x)...))$$

Si **E** el estado inicial de un cómputo y **C** es una abstracción que se reduce un estado para obtener otro.

$$\mathbf{n}\,\mathbf{C}\,\mathbf{E} =_\beta \mathbf{E'}$$

Donde **E′** es el estado del cómputo después de n iteraciones de **C**.

Aritmética

Generalización de operaciones elementales: Hiperoperaciones.

$$
\begin{aligned}
\uparrow_{\mathbf{1}} &= \lambda m\, n.n\, (\times\ m)\, \mathbf{1} \\
\uparrow_{\mathbf{2}} &= \lambda m\, n.n\, (\uparrow_{\mathbf{1}}\ m)\, \mathbf{1} \\
\uparrow_{\mathbf{3}} &= \lambda m\, n.n\, (\uparrow_{\mathbf{2}}\ m)\, \mathbf{1} \\
&\dots \\
\uparrow_{\mathbf{n}} &= \lambda m\, n.n\, (\uparrow_{\mathbf{n-1}}\ m)\, \mathbf{1}
\end{aligned}
$$

Procesos recursivos

Es posible codificar mecanismos de recursividad.

Teorema de punto fijo para el cálculo λ:

Para todo término M existe un término N tal que

$$M\,N =_\beta N$$

Procesos recursivos

Los combinadores de punto fijo son términos que "computan" el punto fijo de un término.

Si $\mathbf{Y}$ es un combinador de punto fijo, entonces

$$M\,(\mathbf{Y}\,M) =_\beta (\mathbf{Y}\,M)$$

Procesos recursivos

Los combinadores de punto fijo son términos que "computan" el punto fijo de un término.

Si $\mathbf{Y}$ es un combinador de punto fijo, entonces

$$M\,(\mathbf{Y}\,M) =_\beta (\mathbf{Y}\,M)$$

$$\mathbf{Y} = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

Procesos recursivos

Los combinadores de punto fijo son términos que "computan" el punto fijo de un término.

Si **Y** es un combinador de punto fijo, entonces

$$M\,(\mathbf{Y}\,M) =_\beta (\mathbf{Y}\,M)$$

$$\mathbf{Y} = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

$$(\mathbf{Y}\,M) =_\beta M\,(\mathbf{Y}\,M) =_\beta M\,(M\,(\mathbf{Y}\,M)) =_\beta \ldots$$

Procesos recursivos

¿Cómo se pudiera codificar el algoritmo **factorial**?

$$\mathbf{factorial} = \mathbf{Y}\,(\lambda f\ n.\mathbf{if}\,(\mathbf{cero}\,n)\,\mathbf{1}\,(\boldsymbol{\times}\ n\,(f\ (\mathbf{pre}\,n))))$$

Procesos recursivos

¿Cómo se pudiera codificar el algoritmo **factorial**?

$$\textbf{factorial} = \mathbf{Y}\,(\lambda f\; n.\textbf{if}\;(\textbf{cero}\;n)\;\mathbf{1}\;(\boldsymbol{\times}\;n\,(f\;(\textbf{pre}\;n))))$$

$$\begin{aligned}\textbf{cero} &= \lambda n.n\,((\lambda x\,y.x)\,\mathbf{F})\,\mathbf{T}\\ \textbf{pre} &= \lambda n.\lambda f\;x.(n\,\textbf{suc}\,(\lambda x\,y\,z.y))\;f\;x\,(\lambda z.z)\end{aligned}$$

Estructuras recursivas

Es posible codificar estructuras más complejas que los valores de verdad y los naturales utilizando pares ordenados.

$$\begin{aligned} \textbf{cons} &= \lambda a\, b.\lambda p.p\; a\; b \\ \textbf{primero} &= \lambda c.c\,(\lambda a\, b.a) \\ \textbf{segundo} &= \lambda c.c\,(\lambda a\, b.b) \end{aligned}$$

Estructuras recursivas

Es posible codificar estructuras más complejas que los valores de verdad y los naturales utilizando pares ordenados.

$$\begin{aligned} \textbf{cons} &= \lambda a\, b.\lambda p.p\, a\, b \\ \textbf{primero} &= \lambda c.c\, (\lambda a\, b.a) \\ \textbf{segundo} &= \lambda c.c\, (\lambda a\, b.b) \end{aligned}$$

$$\begin{aligned} \textbf{primero}\, (\textbf{cons}\, M\, N) &=_\beta M \\ \textbf{segundo}\, (\textbf{cons}\, M\, N) &=_\beta N \end{aligned}$$

Codificando listas enlazadas.

$$\mathbf{cons}\, M_1\, (\mathbf{cons}\, M_2\, (\mathbf{cons}\, M_3\, \emptyset))$$

El término $\emptyset$ depende de la forma de los elementos de la lista.

Estructuras recursivas

Codificando árboles y gráficas.

Los árboles se pueden codificar como una lista cuyo primer elemento es la raíz y cuyo segundo elemento es una lista de subárboles.

Las gráficas se pueden codificar como listas de adyacencia: Listas de vértices, donde cada vértice tiene asociada una lista de vértices.

Estructuras recursivas

¿Se pudieran codificar términos λ en el cálculo λ?

¿Se pudieran codificar términos λ en el cálculo λ?

¡Si!

Utilizando pares ordenados

$$\begin{aligned} \textbf{variable}\, x &= \textbf{cons}\, \textbf{1}\, x \\ \textbf{abstraccion}\, x\; M &= \textbf{cons}\, \textbf{2}\, (\textbf{cons}\, x\; M) \\ \textbf{aplicacion}\, M\; N &= \textbf{cons}\, \textbf{3}\, (\textbf{cons}\, M\; N) \end{aligned}$$

Códigos

Codificaciones del cálculo λ en Haskell

```
T x y = x
F x y = y

N_0 f x = x
N_suc n = \x y->x (n x y)
N_1 = N_suc N_0
N_2 = N_suc N_1
```

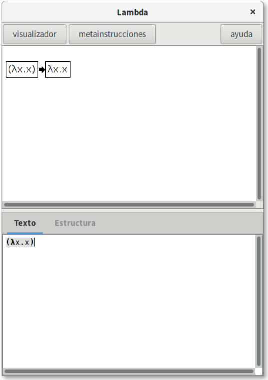
Codificaciones del cálculo λ en Scheme

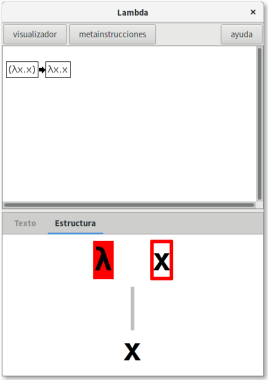
```
(define T (lambda (x) (lambda (y) x)))
(define F (lambda (x) (lambda (y) y)))

(define N_0 (lambda (f) (lambda (x) x)))
(define N_suc (lambda (n) (lambda (f) (lambda (x)
                (f ((n f) x))))))
(define N_1 (N_suc N_0))
(define N_2 (N_suc N_1))
```

Interactuando con λ

`Lambda` es un programa que sirve para estudiar y programar el contenido del trabajo.

Tiene un editor de texto, un editor estructural, un intérprete y un visualizador de términos.

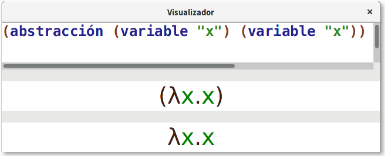




Interactuando con λ

`Lambda` es un programa que sirve para estudiar y programar el contenido del trabajo.

Tiene un editor de texto, un editor estructural, un intérprete y un visualizador de términos.



¿Preguntas?

El código fuente de esta presentación, la tesis y los programas se pueden descargar de

github.com/eduardoacye/tesis