



El cálculo λ

Y los fundamentos de la computación

Eduardo Acuña Yeomans

24 de agosto de 2016

Universidad de Sonora

1. Introducción
2. Conceptos básicos
3. Cómputo
4. Códigos

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

$$\lambda K\beta, \Gamma \vdash M = N$$

¿Qué es el cálculo λ ?



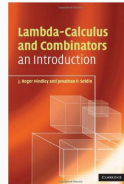
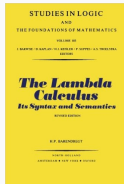
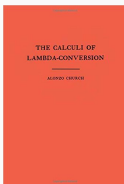
- Un lenguaje para expresar cómputo

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

- Teoría matemática

$$\lambda K\beta, \Gamma \vdash M = N$$

- Sistema para estudiar matemáticas y computación

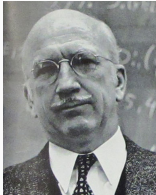


Alonzo Church

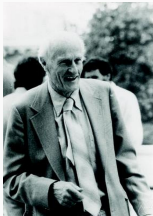


1932 Un conjunto de postulados para la fundamentación de la lógica

Barkley Rosser



Stephen Kleene



1932 Un conjunto de postulados para la fundamentación de la lógica

1935 La inconsistencia de ciertas lógicas formales

Alonzo Church



- 1932** Un conjunto de postulados para la fundamentación de la lógica
- 1935** La inconsistencia de ciertas lógicas formales
- 1936** Un problema sin solución de la teoría de números elemental

$$f : A \rightarrow B$$

$$f(x) = E$$

$$f(a) = b$$

$$f : A \rightarrow B$$

$$f(x) = E$$

$$f(a) = b$$

↓

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

$$(f\ a) = b$$

$$f : A \rightarrow B$$

$$f(x) = E$$

$$f(a) = b$$

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

↓

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

$$(f\ a) = b$$

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

Conceptos básicos

Λ

- Variables
- Abstracciones
- Aplicaciones

Λ

- Variables
- Abstracciones
- Aplicaciones

w, x, y, z , etc.

$$M \in \Lambda$$

- Variables
- Abstracciones
- Aplicaciones

w, x, y, z , etc.

$(\lambda x.M)$

$$M, N \in \Lambda$$

- Variables
- Abstracciones
- Aplicaciones

w, x, y, z , etc.

$(\lambda x. M)$

$(M N)$

Λ

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

El alfabeto tiene como elementos los símbolos:

$(\quad) \quad \lambda \quad .$

y a todas las variables

Identificando la estructura de un término λ

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

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

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

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

$$M[x := N]$$

Ejemplos de sustituciones

$$x[x := (\lambda y.y)] \rightarrow (\lambda y.y)$$

$$x[z := (\lambda y.y)] \rightarrow x$$

$$(x\ z)[x := (\lambda y.y)] \rightarrow (x[x := (\lambda y.y)]\ z[x := (\lambda y.y)])$$

$$(\lambda x.z)[x := (\lambda y.y)] \rightarrow (\lambda x.z)$$

$$(\lambda z.x)[x := (\lambda y.y)] \rightarrow (\lambda z.x[x := (\lambda y.y)])$$

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

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

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

$$((\lambda x.M) N) \rightarrow_{\beta} M[x := N]$$

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

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

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

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

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

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

- Las aplicaciones tienen asociatividad a la izquierda

$$((M_1 M_2) M_3) = M_1 M_2 M_3$$

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

- 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

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

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

- 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

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

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

- 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

Codificación de valores de verdad verdadero y falso.

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

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

Codificación de operaciones

$$\mathbf{if} = \lambda p \times y. p \times y$$

$$\mathbf{not} = \lambda p. \mathbf{if} \ p \ F \ T$$

$$\mathbf{and} = \lambda p \ q. \mathbf{if} \ p \ (\mathbf{if} \ q \ T \ F) \ F$$

$$\mathbf{or} = \lambda p \ q. \mathbf{if} \ p \ T \ (\mathbf{if} \ q \ T \ F)$$

Generalización a 3 valores de verdad

$$V_1 = \lambda x y z. x$$

$$V_2 = \lambda x y z. y$$

$$V_3 = \lambda x y z. z$$

$$if_3 = \lambda p x y z. p x y z$$

Codificación de números naturales.

$$0 = \lambda f x. x$$

$$1 = \lambda f x. f x$$

$$2 = \lambda f x. f (f x)$$

$$3 = \lambda f x. f (f (f x))$$

...

$$n = \lambda f x. \underbrace{f (f (\dots (f x) \dots))}_n$$

Codificación de las operaciones elementales.

$$\mathbf{suc} = \lambda n. \lambda f x. f (n f x)$$

$$\mathbf{+} = \lambda m n. n \mathbf{suc} m$$

$$\mathbf{\times} = \lambda m n. n (\mathbf{+} m) \mathbf{0}$$

$$\mathbf{\uparrow} = \lambda m n. n (\mathbf{\times} m) \mathbf{1}$$

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

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

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

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

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

Generalización de operaciones elementales: **Hiperoperaciones**.

$$\uparrow_1 = \lambda m n.n (\times m) 1$$

$$\uparrow_2 = \lambda m n.n (\uparrow_1 m) 1$$

$$\uparrow_3 = \lambda m n.n (\uparrow_2 m) 1$$

...

$$\uparrow_n = \lambda m n.n (\uparrow_{n-1} m) 1$$

Es posible codificar mecanismos de **recursividad**.

Teorema de punto fijo para el cálculo λ :

Para todo término M existe un término N tal que

$$M N =_{\beta} N$$

Los **combinadores de punto fijo** son términos que “computan” el punto fijo de un término.

Si Y es un combinador de punto fijo, entonces

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

Los **combinadores de punto fijo** son términos que “computan” el punto fijo de un término.

Si Y es un combinador de punto fijo, entonces

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

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

Los **combinadores de punto fijo** son términos que “computan” el punto fijo de un término.

Si Y es un combinador de punto fijo, entonces

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

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

$$(Y M) =_{\beta} M(Y M) =_{\beta} M(M(Y M)) =_{\beta} \dots$$

¿Cómo se pudiera codificar el algoritmo *factorial*?

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

¿Cómo se pudiera codificar el algoritmo *factorial*?

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

$$\mathbf{cero} = \lambda n. n ((\lambda x y. x) \mathbf{F}) \mathbf{T}$$

$$\mathbf{pre} = \lambda n. \lambda f x. (n \ \mathbf{suc} (\lambda x y z. y)) f x (\lambda z. z)$$

Estructuras recursivas

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

$$\mathbf{cons} = \lambda a\ b.\lambda p.p\ a\ b$$

$$\mathbf{primero} = \lambda c.c\ (\lambda a\ b.a)$$

$$\mathbf{segundo} = \lambda c.c\ (\lambda a\ b.b)$$

Estructuras recursivas

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

$$\mathit{cons} = \lambda a\ b.\lambda p.p\ a\ b$$

$$\mathit{primero} = \lambda c.c\ (\lambda a\ b.a)$$

$$\mathit{segundo} = \lambda c.c\ (\lambda a\ b.b)$$

$$\mathit{primero}\ (\mathit{cons}\ M\ N) =_{\beta} M$$

$$\mathit{segundo}\ (\mathit{cons}\ M\ N) =_{\beta} N$$

Codificando listas enlazadas.

$$\mathit{cons} \ M_1 (\mathit{cons} \ M_2 (\mathit{cons} \ M_3 \ \emptyset))$$

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

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.

¿Se pudieran codificar términos λ en el cálculo λ ?



¿Se pudieran codificar términos λ en el cálculo λ ?

¡Si!

Utilizando pares ordenados

variable $x = \mathbf{cons\ 1\ }x$

abstraccion $x\ M = \mathbf{cons\ 2\ (cons\ }x\ M)$

aplicacion $M\ N = \mathbf{cons\ 3\ (cons\ }M\ N)$

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)))))))
```

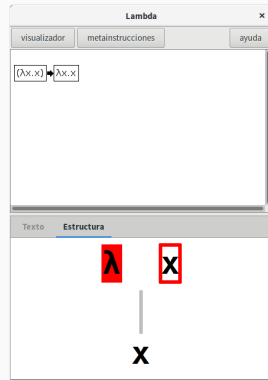
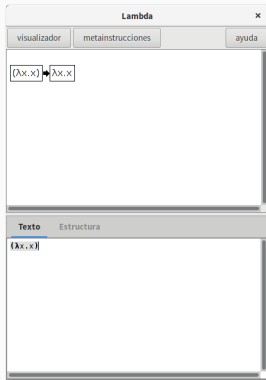
```
(define N_1 (N_suc N_0))
```

```
(define N_2 (N_suc N_1))
```


Interactuando con λ

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

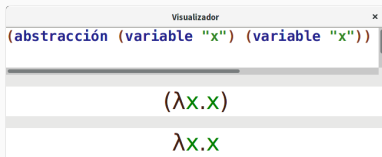
Tiene un editor de texto, un editor estructural, un intérprete y un visualizador de términos.



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`



(Atribución-Licenciamiento Recíproco)