λ-cálculo – un modelo de cómputo que entra en un slide

λ-términos (o sea, los *programas*), se forman con esta regla:

$$x \mid \lambda x.M \mid MN$$

donde M y N son λ-términos y x es una variable

β-reducción (o sea, la semántica)

Si tengo $(\lambda x.M)N$, *reemplazar por* $M[x \mapsto N]$

donde $M[x \mapsto N]$ es: *find & replace* en M de x por N

¿Y dónde está el. . . cómputo?

Empezamos con un λ-término:

$$(\lambda x.\lambda y.x)(zw)(\lambda u.uu)$$

¿Y dónde está el. . . cómputo?

Empezamos con un λ-término:

$$(\lambda x.\lambda y.x)(zw)(\lambda u.uu)$$

buscamos algún lugar donde β-reducir, y reducimos

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

¿Y dónde está el. . . cómputo?

Empezamos con un λ-término:

$$(\lambda x.\lambda y.x)(zw)(\lambda u.uu)$$

buscamos algún lugar donde β-reducir, y reducimos

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

y volvemos a buscar algún lugar donde β-reducir

$$zw$$

¿Y dónde está el. . . cómputo?

Empezamos con un λ-término:

$$(\lambda x.\lambda y.x)(zw)(\lambda u.uu)$$

buscamos algún lugar donde β-reducir, y reducimos

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

y volvemos a buscar algún lugar donde β-reducir

$$zw$$

hasta que ya no podamos reducir más. . .

¿Pero qué es lo que está computando?

Empezamos con un λ-término (o sea, un programa) y cuando terminamos de computar obtuvimos. . . otro λ-término (o sea, otro programa!)

¿Cuál es la intuición?

- Se puede pensar $\lambda x.M$ como una función cuyo argumento es x
- Y MN sería *evaluar la función M con N de parámetro*
- La β-reducción es la que *computa* la evaluación

Funciones de orden superior

Ya vimos que:

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

- O sea que $\lambda x.\lambda y.x$ computa la función $f(x, y) = x$

Funciones de orden superior

Ya vimos que:

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

- O sea que $\lambda x.\lambda y.x$ computa la función $f(x, y) = x$
- pero $(\lambda x.\lambda y.x)M$ es $\lambda y.M$ o sea, la función constante M

Funciones de orden superior

Ya vimos que:

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

- O sea que $\lambda x.\lambda y.x$ computa la función $f(x, y) = x$
- pero $(\lambda x.\lambda y.x)M$ es $\lambda y.M$ o sea, la función constante M
- ¡$\lambda x.\lambda y.x$ es una función que devuelve una función!

Numerales de Church

- No tenemos los naturales como tipos “primitivos”

Numerales de Church

- No tenemos los naturales como tipos "primitivos"
- Solución: cada natural se representa con una función
  (al revés que la Gödelización...)

Numerales de Church

- No tenemos los naturales como tipos "primitivos"
- Solución: cada natural se representa con una función
  (al revés que la Gödelización...)
- Codificación de Church:

$$
\begin{array}{rcl}
\overline{0} & = & \lambda f \lambda x.x \\
\overline{1} & = & \lambda f \lambda x.fx \\
\overline{2} & = & \lambda f \lambda x.f(fx) \\
\overline{3} & = & \lambda f \lambda x.f(f(fx)) \\
 & \vdots & \\
\overline{n} & = & \lambda f \lambda x.\underbrace{f(f(f \ldots f(}_{n} x) \ldots))
\end{array}
$$

Funciones sobre naturales

$$\mathsf{suc} \quad = \quad \lambda n \lambda f \lambda x \; . \; f \; (n \; f \; x)$$

Funciones sobre naturales

$$\begin{aligned} \text{suc} &= \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\ \text{suma} &= \lambda n \lambda m \;.\; n\ \text{suc}\ m \end{aligned}$$

Funciones sobre naturales

$$
\begin{array}{rcl}
\mathsf{suc} & = & \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\
\mathsf{suma} & = & \lambda n \lambda m \;.\; n\ \mathsf{suc}\ m \\
\mathsf{mult} & = & \lambda n \lambda m \;.\; n\ (\mathsf{suma}\ m)\ \overline{0}
\end{array}
$$

Funciones sobre naturales

$$
\begin{array}{rcl}
\mathsf{suc} & = & \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\
\mathsf{suma} & = & \lambda n \lambda m \;.\; n\ \mathsf{suc}\ m \\
\mathsf{mult} & = & \lambda n \lambda m \;.\; n\ (\mathsf{suma}\ m)\ \overline{0} \\
\mathsf{exp} & = & \lambda n \lambda m \;.\; n\ (\mathsf{mult}\ m)\ \overline{1} \\
 & \vdots &
\end{array}
$$

Funciones sobre naturales

$$
\begin{array}{rcl}
\text{suc} & = & \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\
\text{suma} & = & \lambda n \lambda m \;.\; n\ \text{suc}\ m \\
\text{mult} & = & \lambda n \lambda m \;.\; n\ (\text{suma}\ m)\ \overline{0} \\
\text{exp} & = & \lambda n \lambda m \;.\; n\ (\text{mult}\ m)\ \overline{1} \\
 & \vdots &
\end{array}
$$

Definir la resta es más difícil. . . ¡fue un problema abierto por años!

Funciones sobre naturales

$$
\begin{array}{rcl}
\text{suc} & = & \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\
\text{suma} & = & \lambda n \lambda m \;.\; n\ \text{suc}\ m \\
\text{mult} & = & \lambda n \lambda m \;.\; n\ (\text{suma}\ m)\ \overline{0} \\
\text{exp} & = & \lambda n \lambda m \;.\; n\ (\text{mult}\ m)\ \overline{1} \\
 & \vdots &
\end{array}
$$

Definir la resta es más difícil. . . ¡fue un problema abierto por años!

$$\text{pred} \quad = \quad \lambda n \lambda f \lambda x \;.\; n\ (\lambda g \lambda h \;.\; h\ (g\ f))\ (\lambda u.x)\ (\lambda u \;.\; u)$$

Funciones sobre naturales

$$
\begin{array}{rcl}
\text{suc} & = & \lambda n \lambda f \lambda x \;.\; f\ (n\ f\ x) \\
\text{suma} & = & \lambda n \lambda m \;.\; n\ \text{suc}\ m \\
\text{mult} & = & \lambda n \lambda m \;.\; n\ (\text{suma}\ m)\ \overline{0} \\
\text{exp} & = & \lambda n \lambda m \;.\; n\ (\text{mult}\ m)\ \overline{1} \\
 & \vdots &
\end{array}
$$

Definir la resta es más difícil. . . ¡fue un problema abierto por años!

$$
\begin{array}{rcl}
\text{pred} & = & \lambda n \lambda f \lambda x \;.\; n\ (\lambda g \lambda h \;.\; h\ (g\ f))\ (\lambda u.x)\ (\lambda u \;.\; u) \\
\text{resta} & = & \lambda n \lambda m \;.\; n\ \text{pred}\ m
\end{array}
$$

Booleanos, IFs, etc

$$\begin{aligned} \text{true} &= \lambda i \lambda e \,.\, i \\ \text{false} &= \lambda i \lambda e \,.\, e \end{aligned}$$

true es como un IF que siempre da verdadero y viceversa

Booleanos, IFs, etc

$$\begin{aligned} \text{true} &= \lambda i \lambda e \,.\, i \\ \text{false} &= \lambda i \lambda e \,.\, e \end{aligned}$$

true es como un IF que siempre da verdadero y viceversa

$$\begin{aligned} \text{not} &= \lambda b \lambda i \lambda e \,.\, b\ e\ i \\ \text{and} &= \lambda b_1 \lambda b_2 \lambda i \lambda e \,.\, b_1\ (b_2\ i\ e)\ e \\ \text{or} &= \lambda b_1 \lambda b_2 \lambda i \lambda e \,.\, b_1\ i\ (b_2\ i\ e) \end{aligned}$$

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

podemos β-reducir en un sólo lugar y obtenemos...

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

podemos β-reducir en un sólo lugar y obtenemos...

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

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

podemos β-reducir en un sólo lugar y obtenemos...

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

y, otra vez, podemos β-reducir en un sólo lugar y obtenemos...

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

podemos β-reducir en un sólo lugar y obtenemos...

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

y, otra vez, podemos β-reducir en un sólo lugar y obtenemos...

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

¿El λ-cálculo computa sólo funciones totales?

Consideremos el término

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

podemos β-reducir en un sólo lugar y obtenemos...

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

y, otra vez, podemos β-reducir en un sólo lugar y obtenemos...

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

¡se cuelga!

¿Y computa todas las funciones computables?

¿Y computa todas las funciones computables?

Sí

¿Y computa todas las funciones computables?

Sí

pero hacen falta ver muchas más cosas para convencerse. . .