

Introducción al λ -cálculo

Otro camino en la resolución de problemas

Marco Antonio Garrido Rojo¹

4 de julio de 2016

¹<https://github.com/MaSteve/lambda-calculo>

Un poco de historia

- Desde los tiempos de Leibniz se intentó conseguir la creación de un lenguaje universal para formular todos los problemas posibles y un método de decisión para resolverlos (en caso de existir).
- Lo primero se consiguió con la lógica de primer orden, mientras que para lo segundo tuvimos que esperar hasta el año 1936 para obtener como respuesta un no.
- Alan Turing inventó las máquinas de Turing mientras que Alonzo Church inventó un sistema formal llamado λ -cálculo. Posteriormente, Turing demostró que ambos métodos definían la misma noción de computabilidad.

Un poco de historia

- Las máquinas de Turing inspiraron el diseño de nuestros actuales ordenadores con arquitectura Von Neumann y los lenguajes de programación imperativos.
- Los lenguajes funcionales, por otro lado, se basan en el λ -cálculo, al igual que lo hacen las máquinas de reducción, diseñadas para la ejecución de este tipo de programas.

¿Qué es esto del λ -cálculo?

Es un sistema formal basado en funciones que expresan los datos y los algoritmos que se aplican sobre estos. Se apoya en los principios de:

- **Reducción:** reemplazar una parte P de E por otra expresión P' con el fin de simplificar. Cuando no puede aplicarse más veces este proceso a una expresión se dice que se ha alcanzado la forma normal E^* de E .
- **Aplicación:** $F \cdot A$. Emplear A como entrada del algoritmo F . Al no existir tipos en esta noción una función puede aplicarse a sí misma.
- **Abstracción:** Dada una expresión $M[x]$, $\lambda x.M[x]$ denota una función en la que a cada N le asigna el resultado de $(\lambda x.M[x])N = M[x := N]$.

¿Qué es esto del λ -cálculo?

Definimos el conjunto de los λ -términos (Λ) de forma recursiva sobre un conjunto V de variables infinito:

- $x \in V \Rightarrow x \in \Lambda$
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$
- $M \in \Lambda, x \in V \Rightarrow \lambda x.M \in \Lambda$

De forma similar se puede definir el conjunto de variables libres de una expresión M ($FV(M)$) como:

- $FV(x) = \{x\}$
- $FV(MN) = FV(M) \cup FV(N)$
- $FV(\lambda x.M) = FV(M) - \{x\}$

Una expresión sin variables libres es una **expresión cerrada**.

¿Qué es esto del λ -cálculo?

Algunos axiomas y reglas:

- **Axioma principal del λ -cálculo (regla β):**
 $(\lambda x.M[x])N = M[x := N]$
- **Regla α :** $\lambda x.M = \lambda y.M[x := y]$ donde y no aparece en M .
- **Regla η :** $\lambda x.Mx = M$ donde x no aparece en M .
- **Igualdad:**
 - $M = M$
 - $M = N \Rightarrow N = M$
 - $M = N, N = L \Rightarrow M = L$
- **Reglas de compatibilidad:** Si $M = M'$ tenemos que:
 - $MZ = M'Z$
 - $ZM = ZM'$
 - $\lambda x.M = \lambda x.M'$

¿Qué podemos hacer con el λ -cálculo?

Algunos operadores conocidos:

- $I \equiv \lambda x.x$
- $K \equiv \lambda xy.x$
- $K_* \equiv \lambda xy.y$
- $S \equiv \lambda xyz.xz(yz)$

Teorema del punto fijo: $\forall F, \exists X / FX = X$, en concreto tenemos que $F(YF) = YF$ donde $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

¿Qué podemos hacer con el λ -cálculo?

Numerales de Church: Definiendo $F^n(M)$ de forma recursiva como:

- $F^0(M) \equiv M$
- $F^{n+1}(M) \equiv F(F^n(M))$

definimos los numerales de Church como $c_n = \lambda f x. f^n(x)$ y los operadores

- $A_+ \equiv \lambda x y p q. x p (y p q)$ con $A_+ c_n c_m = c_{n+m}$
- $A_* \equiv \lambda x y z. x (y z)$ con $A_* c_n c_m = c_{n*m}$
- $A_{exp} \equiv \lambda x y. y x$ con $A_{exp} c_n c_m = c_n^m$

¿Qué podemos hacer con el λ -cálculo?

¿Qué hay de los booleanos? Definámoslos:

- $True \equiv K$
- $False \equiv K_*$

La expresión `if B then P else Q` queda ahora de la forma BPQ .

Utilizaremos por comodidad la notación $[M, N] = \lambda z.zMN$ de modo que:

- $[M, N]True = M$
- $[M, N]False = N$

¿Qué podemos hacer con el λ -cálculo?

Una alternativa a los numerales de Church:

- $[0] \equiv I$
- $[n + 1] \equiv [False, [n]]$

Con los siguientes operadores:

- $S^+ \equiv \lambda x.[False, x]$
- $P^- \equiv \lambda x.xFalse$
- $Zero \equiv \lambda x.xTrue$

Hablemos de computación

Una función φ es λ -definible si existe un operador F tal que $F[n_1] \dots [n_p] = [\varphi(n_1 \dots n_p)]$. Las funciones anteriores junto con las proyecciones son λ -definibles.

Decimos que una clase de funciones \mathcal{A} es:

- **Cerrada bajo composición** si dada $\varphi(\vec{n}) = \chi(\psi_1(\vec{n}), \dots, \psi_m(\vec{n}))$ donde $\chi, \psi_i \in \mathcal{A}$, se tiene que $\varphi \in \mathcal{A}$.
- **Cerrada bajo recursión** si dada $\varphi(0, \vec{n}) = \chi(\vec{n})$ y $\varphi(k+1, \vec{n}) = \psi(\varphi(k, \vec{n}), k, n)$ donde $\chi, \psi \in \mathcal{A}$, se tiene $\varphi \in \mathcal{A}$.
- **Cerrada bajo la condición de mínimo** si dada $\varphi(\vec{n}) = \mu m [\chi(\vec{n}, m) = 0]$ con $\chi \in \mathcal{A}$ tal que $\forall \vec{n} \exists m \chi(\vec{n}, m) = 0$, se tiene $\varphi \in \mathcal{A}$.

Hablemos de computación

La clase \mathcal{R} de las funciones recursivas es la clase más pequeña que contiene a los operadores iniciales y que es cerrada bajo composición, recursión y la condición de mínimo. Teniendo que las funciones iniciales son λ -definibles y que toda función λ -definible es cerrada respecto a las condiciones anteriores se tiene que toda función recursiva es λ -definible por construcción.

Hablemos de computación

¿Qué pasa con los numerales de Church?

- Previamente los hemos definido como un método de numeración pero en la definición de λ -definible no se emplean.
- Es conveniente ver que ambas notaciones son equivalentes y para ello se emplean las funciones
 - $T \equiv \lambda x.xS^+[0]$
 - $T^{-1} \equiv \lambda x.[c_0, S_c^+(T^{-1}(P^-x))]Zero[x]$

que definimos junto con los operadores

- $S_c^+ \equiv \lambda xyz.y(xyz)$
- $P_c^- \equiv \lambda xyz.x(\lambda pq.q(py))(Kz)I$
- $Zero_c \equiv \lambda x.x(KFalse)True$

De este modo se puede replantear la definición de λ -definible con los numerales de Church.

Alguna curiosidad más

- Originalmente, la teoría del λ -cálculo no contaba con tipos. Una ampliación al respecto se debe a Curry que incorporó la noción de tipo a la teoría.
- Aunque ya sepamos que toda función computable se puede expresar con λ -términos, existe una versión extendida que admite constantes para facilitar el manejo de ciertas operaciones. Mediante una nueva regla de reducción (δ) aportamos significado a dichas constantes.

Alguna curiosidad más sobre el λ -cálculo con tipos

- Definimos el conjunto \mathbb{T} de tipos sobre el conjunto \mathbb{V} de variables de tipo y \mathbb{B} con “tipos constantes” como:
 - $\alpha \in \mathbb{V} \Rightarrow \alpha \in \mathbb{T}$
 - $B \in \mathbb{B} \Rightarrow B \in \mathbb{T}$
 - $\sigma, \tau \in \mathbb{T} \Rightarrow (\sigma \rightarrow \tau) \in \mathbb{T}$
- Usaremos la notación $M : \sigma$ para asignar M al tipo σ . Una base Γ será un conjunto de asignaciones.
- La derivación de un tipo se sigue de las siguientes reglas, asumiendo $x : \sigma$:
 - $M : \sigma \rightarrow \tau, N : \sigma \Rightarrow MN : \tau$
 - $x : \sigma, M : \tau \Rightarrow \lambda x.M : \sigma \rightarrow \tau$

Alguna curiosidad más sobre el λ -cálculo con tipos

- A partir de un Γ (que puede ser vacío) decimos que $\Gamma \vdash M : \sigma$ si hay una derivación en la que toda asignación no cancelada está en Γ .
- Lema de generación:
 - $\Gamma \vdash x : \sigma \Rightarrow (x : \sigma) \in \Gamma$
 - $\Gamma \vdash MN : \tau \Rightarrow \exists \sigma / \Gamma \vdash M : (\sigma \rightarrow \tau) \wedge \Gamma \vdash N : \sigma$
 - $\Gamma \vdash \lambda x.M : \rho \Rightarrow \exists \sigma, \tau / \Gamma \cup \{x : \sigma\} \vdash M : \tau \wedge \rho \equiv (\sigma \rightarrow \tau)$
- Lema de sustitución:
 - $\Gamma \vdash M : \sigma \Rightarrow \Gamma[\alpha := \tau] \vdash M : \sigma[\alpha := \tau]$
 - Si $\Gamma \cup \{x : \sigma\} \vdash M : \tau$ y $\Gamma \vdash N : \sigma$, entonces $\Gamma \vdash M[x := N] : \tau$
- Tipado de los subtérminos: Si M tiene un tipo, todo subtérmino de M también tiene un tipo.
- Teorema de reducción: Si M es reducible a M' mediante la aplicación de la regla β , ambos términos tienen el mismo tipo.

Alguna curiosidad más sobre la normalización

- A partir de las herramientas antes discutidas podemos llegar a un resultado importante de la teoría del λ -cálculo.
- Mientras que en el λ -cálculo sin tipos podemos encontrar expresiones del estilo $(\lambda x.xx)(\lambda x.xx)$, estas resultan imposibles de concebir en la teoría con tipos.
- El λ -cálculo con tipos resulta fuertemente normalizable, lo que quiere decir que toda expresión tiene una forma normal alcanzable en un número finito de pasos sea cual sea el camino escogido. (¿Alguien ve el problema que tenemos con la Turing completitud?)

Alguna curiosidad más sobre la normalización

- Al menos podemos conformarnos con que el λ -cálculo cumple el teorema de Church–Rosser que nos dice que el orden en que tomemos las transformaciones no afecta al resultado final, es decir, se da el resultado de confluencia.