Introducción al λ-cálculo

Otro camino en la resolución de problemas

Marco Antonio Garrido Rojo

4 de julio de 2016

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 decisíon 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:

- **Redución:** 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 aplicación 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:** $(\lambda x.M[x])N = M[x := N]$
- **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 $c_n = \lambda fx.f^n(x)$ y los operadores

- $A_+ \equiv \lambda xypq.xp(ypq)$
- $A_* \equiv \lambda xyz.x(yz)$
- $A_{exp} \equiv \lambda xy.yx$

¿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:

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

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 $F\lceil n_1\rceil \dots \lceil n_p\rceil = \lceil \varphi(n_1 \dots n_p)\rceil$. Las funciones anteriores junto con las proyecciones son λ-definibles.

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

- **Cerrada bajo composición** si dada $\varphi(n) = \chi(\psi_1(n), \dots, \psi_m(n))$ donde $\chi, \psi_i \in \mathcal{A}$, se tiene que $\varphi \in \mathcal{A}$.
- **Cerrada bajo recursión** si dada $\varphi(0, n) = \chi(n)$ y $\varphi(k+1, n) = \psi(\varphi(k, n), k, n)$ donde $\chi, \psi \in \mathcal{A}$, se tiene $\varphi \in \mathcal{A}$.
- **Cerrada bajo la condición de mínimo** si dada $\varphi(n) = \mu m[\chi(n, m) = 0]$ con $\chi \in \mathcal{A}$ tal que $\forall n \exists m \chi(n, m) = 0$, 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?
No hay problema porque podemos convertirlos a la notación alternativa (y viceversa) mediante

- $T \equiv \lambda x.xS^+ \lceil 0 \rceil$
- $T^{-1} \equiv \lambda x.[c_0, S_c^+(T^{-1}(P^- x))]Zero\lceil x \rceil$

donde definimos

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

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.