

# 2 - calculus

## Historia

La **numeración** que conocemos fue introducida en Occidente sobre el 1200. Su valor fundamental radica en basarse en un conjunto pequeño de dígitos, cuyo valor varía con su posición en la escritura del número. El ingenioso invento fue tomado de los científicos árabes.

(Fibonacci, Liber Abaci 1202)

La notación para las **expresiones** y las **ecuaciones** no estuvo dispuesta hasta el siglo 17, cuando Cardano Viète comenzó a hacer uso sistemático de marcadores de posición para parámetros y abreviaturas para las operaciones aritméticas! Hasta entonces los significados que encerraron las ecuaciones eran explicados literariamente, lo que hacía muy difícil el seguimiento de las explicaciones.

Pasaron cerca de 250 años hasta que Alonzo Church desarrollara una notación para funciones arbitrarias. Esta notación dio pie a una teoría conocida como  **$\lambda$ -calculus** o cálculo lambda.

La intención de Church era hacer una fundamentación funcional de la matemática, pero los matemáticos prefirieron la teoría axiomática de conjuntos.

El  $\lambda$ -calculus fue redescubierto como una generalidad de enorme potencial en las Ciencias de la Computación. El hecho se debe a personas como : McCarthy, Strachey, Louden y Scott en la década de 1960.

En 1960, McCarthy presentó un lenguaje de procesamiento de listas llamado Lisp. Este lenguaje está inspirado en el  $\lambda$ -calculus.

La línea que comenzó con Lisp ha

continuado con ML y Haskell, pero algunos piensan que ha tenido muchas dificultades para incorporar características de la programación dirigida a objetos y ha habido intentos de conciliar los dos paradigmas.

Sin embargo a algunos no nos satisface el abstracto mecanismo de creación y manipulación de objetos en lenguajes de éxito, como Python, y sin embargo nos parecen muy naturales y fáciles de usar los tipos de dato y las clases de Haskell.

El λ-calculus fue introducido, como dijimos, por Alonzo Church al principio de la década de 1930. El formalismo inicial fue evidenciado como inconsistente por Stephen Kleene y J. B. Rosser.

Así pues en 1936, Church aisgó la parte del formalismo apropiada

para la computación, lo que ahora llamamos  $\lambda$ -cálculos no tipado. En 1940 introducejo un sistema computacionalmente más débil, pero lógicamente consistente, que es conocido como  $\lambda$ -cálculos simplemente tipado.

Parece ser que la notación inicial de abstracción de una variable era  $\hat{x}$  que evolucionó a  $\lambda x$  que para facilitar la impresión evolucionó a  $\lambda x$ .

El  $\lambda$ -cálculos es un sistema formal diseñado para investigar: la definición de función, la noción de aplicación de funciones y la recursión.

Lo que introducejo al  $\lambda$ -cálculos en el estrellato de los formalismos útiles fue la respuesta por parte de Church en 1936 al conocido como Entscheidungsproblem (problema de decisión).

Este problema es el de encontrar un algoritmo general que decidiese si

una fórmula del cálculo de primer orden es universalmente válida o no. Lo que demostró Church (y luego Turing) es que es imposible encontrar un tal algoritmo porque no puede existir. Como consecuencia es imposible decidir con un algoritmo general si ciertas frases concretas de la aritmética son ciertas o falsas.

La prueba de Turing se basó en la hipercomputación y la extensión de las máquinas de Turing, tema que sirvió en 1938 para fundar en Princeton su tesis doctoral. Hemos de señalar que la mayor parte del tiempo de estudio en Princeton desde 1937 a 1938 lo pasó bajo la dirección de Church.

Otra cosa demostrada en aquel tiempo fue la imposibilidad de resolver con un algoritmo general el interrogante de si dos expresiones dadas

del λ-calculus son equivalentes.

## Carácter Schönfinkel

El λ-calculus puede ser considerado como un lenguaje de programación universal, el más minimalista.

Decimos que es universal porque cualquier función computable puede ser expresada y evaluada a través de él.

Por tanto, el formalismo es equivalente a la máquina de Turing. A diferencia de ésta, parece más cercano al software que al hardware.

Para hacernos una idea del contenido de este formalismo, pensemos en cuatro elementos:

- )  $x^2 - 2x + 5$
- ) 2
- )  $2^2 - 2 \cdot 2 + 5$
- ) 5

y lo que los vincula

En primer lugar deberíamos dejar claro qué es variable y qué no es  $x^2 - 2x + 5$ . Eso lo hacemos con la expresión abstracta:

(abstraer)  $\lambda x(x^2 - 2x + 5)$  (1)

Seguidamente vincular (1) y 2

(aplicar)  $(\lambda x(x^2 - 2x + 5))2$  (2)  
luego "sustituir"

(sustituir)  $2^2 - 2 \cdot 2 + 5$  (3)  
y simplificar en (3) para obtener:

(reducir)  $4 - 4 + 5$   
y seguidamente

(reducir) 5

Esquemáticamente podríamos escribir

$$\begin{aligned} (\lambda x(x^2 - 2x + 5))2 &\triangleright 2^2 - 2 \cdot 2 + 5 \\ &= 4 - 4 + 5 \\ &= 5 \end{aligned}$$

Pues hemos entendido entonces  
el área de nuestro formalismo  
que no es otra que la  $\beta$ -conversión

$$(\lambda x M)N \triangleright M[x := N]$$

La idea es que es posible contraer  
o reducir ( $\triangleright$ ) una aplicación

$(\lambda x M)N$   
de una una abstracción:

$\lambda x M$   
sobre un término  $N$ , con sólo arro-  
jarlo en las ocurrencias de  $x$  en  
 $M$  que puedan recibarlo

$$M[x := N]$$