

Formalización del teorema de Church-Rosser

Juan Agustín Bongiovanni

29 de julio de 2019

Índice general

1. Introduccion	5
2. Cálculo Lambda	7
2.1. Historia	7
2.2. Introducción	8
2.3. Definición formal	9
2.4. Teorema Church-Rosser	10
3. Agda	13
4. Sintaxis	15
5. Teorema Church-Rosser	17
6. Conclusiones	19

Capítulo 1

Introduccion

Capítulo 2

Cálculo Lambda

2.1. Historia

Gottfried Wilhelm Leibniz tenía como ideal conseguir lo siguiente:

1) Crear un lenguaje universal en el que todos los problemas posibles puedan estar.

2) Encontrar un método con el que se pueda resolver todos los problemas establecidos en el lenguaje universal.

El problema 1 se puede cumplir adoptando alguna forma de teoría de conjuntos en el lenguaje de la lógica predicada de primer orden. El punto 2 trae una difícil pregunta: ¿se puede resolver todos los problemas formulados en el lenguaje universal? Esto se puede ver que no, pero no es muy claro probar eso. Esta pregunta se conoce como el Entscheidungsproblem. En 1936 con el objetivo de resolver el Entscheidungsproblem, Alonzo Church y Alan Turing adoptaron una formalización de la noción 'computable'. Church y Turing presentaron dos modelos de computación diferentes:

1) Church (1936) inventó un sistema formal llamado Cálculo Lambda y definió la idea de computabilidad a través de este sistema.

2) Turing (1936/1937) inventó una clase de máquinas (luego llamadas las máquinas de Turing) y definió la idea de computabilidad basada en estas máquinas.

El cálculo lambda es un lenguaje universal ya que cualquier función computable puede ser expresada y evaluada a través de él, este lenguaje es capaz de decidir que función puede ser computable o no. Este tipo de demostraciones son equivalentes a las máquinas de Turing, con la diferencia de que el cálculo lambda no hace demasiado uso de reglas de transformación (usa sustituciones de variables) y no está pensado para que pueda ser implementado en máquinas reales.

Considerado el primer lenguaje funcional, este lenguaje es el fundamento de los lenguajes funcionales. El cálculo lambda se puede considerar como el lenguaje de programación más pequeño, ya que consiste en una regla de sustitución de variables, además de un esquema simple de definición de funciones.

El primer lenguaje funcional en aparecer fue LISP, diseñado en 1958 por John

McCarthy en el entorno de la computación simbólica. A este lenguaje siguieron otros como ML, el Miranda y el Haskell.

2.2. Introducción

Un programa funcional consiste en una expresión E la cual está sujeta a reglas de reescritura. La reducción se basa en reemplazar una parte P de E por otra expresión P' de acuerdo a reglas de reescritura dadas. La notación sería la siguiente

$$E[P] \rightarrow E[P']$$

Este proceso de reducción se debe repetir hasta obtener una expresión la cual no tenga más partes que puedan ser reemplazadas. A esta última se la llama forma normal E^* de la expresión E , y es el dato de salida del programa funcional. Se puede dar un ejemplo, en el cual las reglas de reducción consisten en las operaciones sobre números:

$$\begin{aligned} (20-5)+(11+8*2) &\rightarrow 15+(11+8*2) \\ &\rightarrow 15+(11+16) \\ &\rightarrow 15+27 \\ &\rightarrow 42 \end{aligned}$$

El sistema de reducción usualmente satisface la propiedad Church-Rosier que establece que la forma normal obtenida es independiente del orden en el que se evalúen los subterminos. Tomando el mismo ejemplo, lo podemos ver:

$$\begin{aligned} (20-5)+(11+8*2) &\rightarrow (20-5)+(11+16) \\ &\rightarrow (20-5)+27 \\ &\rightarrow 15+27 \\ &\rightarrow 42 \end{aligned}$$

o evaluando varias sub-expresiones al mismo tiempo:

$$\begin{aligned} (20-5)+(11+8*2) &\rightarrow 15+(11+16) \\ &\rightarrow 15+27 \\ &\rightarrow 42 \end{aligned}$$

Tenemos dos operaciones básicas en el cálculo lambda: la aplicación y la abstracción. La primera de ellas se denota a través de la expresión $F.A$ o FA , donde la F es considerada un algoritmo al cual se le aplica como entrada el dato A . Es de tipo libre, esto permite considerar expresión como por ejemplo FF , donde F es aplicada a sí misma. Esto es útil para simular la recursión. La otra operación básica es la abstracción. Si $M \equiv M[x]$ es una expresión que contiene a x , entonces $\lambda x.M[x]$ denota la función $x \mapsto M[x]$. Estas dos operaciones trabajan juntas. Esto lo podemos ver en la siguiente fórmula:

$$(\lambda x.2 * x + 1)3 = 2 * 3 + 1 (= 7)$$

2.3. Definición formal

La sintaxis oficial del calculo lambda esta contenida en la siguiente definición:

Definición 1. *El alfabeto del cálculo lambda contiene los parentesis y corchetes izquierdos y derechos, el simbolo λ y un conjunto infinito de variables. Los términos de tipo lambda se definen inductivamente de la siguiente manera:*

1. *Toda variable es un λ -términos*
2. *Si M y N son λ -términos, entonces (MN) lo es también*
3. *Si M es un λ -término y x es una variable, entonces $(\lambda x[M])$ es un λ -término*

Los términos formados a partir de la regla (2) son llamados términos de aplicación, mientras que los formados a partir de la regla (3) son término de abstracción. Para poder omitir paréntesis y ahorrarse escribir paréntesis innecesarios se adopta la convención de asociación a la izquierda, por ejemplo cuando se juntan mas de dos términos como $M_1M_2M_3\dots M_n$ se pueden recuperar los paréntesis faltantes asociando a la izquierda y quedaría $((M_1M_2)M_3)\dots M_n$

Variable ligadas y libres

Se puede definir la noción de variables libres y ligadas de la siguiente manera:

Definición 2. *Las funciones sintácticas FV and BV (variables libres y variables ligadas, respectivamente) son definidas en el conjunto de λ -términos de forma inductiva:*

Para cualquier variable x y términos M y N :

1. $FV(x) = x$
2. $FV(MN) = FV(M) \cup FV(N)$
3. $FV(\lambda x[M]) = FV(M) - x$

1. $BV(x) = \text{emptyset}$
2. $BV(MN) = BV(M) \cup BV(N)$
3. $BV(\lambda x[M]) = BV(M) \cup x$

A continuación, definimos sustitución:

Definición 3 (Sustitución). *Escribimos $M[x:=N]$ para denotar la sustitución de N por las ocurrencias libres de x en M . Una definición precisa por recursion en el conjunto de λ -términos es la siguiente: para todos términos A, B y M , y para todas variables x, y tenemos:*

1. $x[x:=M] \equiv M$
2. $y[x:=M] \equiv y$ (y distinto a x)
3. $(AB)[x:=M] \equiv A[x:=M]B[x:=M]$
4. $(\lambda x[A])[x:=M] \equiv \lambda x[A]$
5. $(\lambda y[A])[x:=M] \equiv \lambda y[A[x:=M]]$ (y distinto a x)

La parte (1) dice que si queremos sustituir M por x y estamos simplemente tenemos a x , entonces el resultado será M . La parte (2) dice que no pasa nada

cuando queremos sustituir x y estamos tratando con una variable distinta. La cláusula (3) dice que la sustitución se distribuye en la aplicación. Las cláusulas (4) y (5) se refieren a términos de abstracción y cláusulas paralelas (1) y (2): si la variable enlazada z del término de abstracción $\lambda z[A]$ es idéntico a la variable x para la que debemos aplicar la sustitución entonces no realizamos ninguna sustitución. Esto es porque $M[x:=N]$ denota la sustitución de N por las ocurrencias libres de x en M . Si M es un término de abstracción $\lambda x[A]$ cuya variable ligada es x , entonces x no ocurre libremente en M , por lo que no hay nada para hacer. Esto explica la cláusula (4). En cambio, en la (5) si la variable ligada de un término de abstracción difiere de x , entonces al menos tiene la posibilidad de ocurrir libremente en el término de abstracción, y la sustitución continúa en el cuerpo de este término.

Definición 4 (Cambio de variable ligadas, α -conversión). *El término N es obtenido del término M mediante el cambio de variable ligada si cualquier término de abstracción $\lambda x[A]$ dentro de M ha sido reemplazado por $\lambda y[A[x:=y]]$*

Decimos que dos términos M y N son α -convertibles si hay una secuencia de cambios de variables ligadas empezando por M y que termina en N .

Reducción

Hay varias nociones de reducción que son válidas en el cálculo lambda, pero la principal es *beta-reducción*

Definición 5 (Un paso de β -reducción). *Para λ -términos A y B , decimos que A se β -reduce en un paso a B , si existe un subtérmino C en A , una variable x y λ -términos M y N tal que $C \equiv (\lambda[M])N$ y B es A a excepción de que una ocurrencia de C en A sea reemplazada por $M[x:=N]$.*

Podemos ver varios ejemplos:

1) $(\lambda x[x])a \rightarrow_i \beta a$ 2) $(\lambda x[y])a \rightarrow_i \beta y$ y 3) $(\lambda y[y5])(\lambda x[3*x]) \rightarrow_i \beta (\lambda x[3*x])5 \rightarrow_i \beta 3*5$ 4) El término $(\lambda x[(\lambda y[xy])a])b$ se puede reducir en un paso en dos diferentes λ -términos: $(\lambda x[(\lambda y[xy])a])b \rightarrow_i \beta (\lambda y[by])a$ y $(\lambda x[(\lambda y[xy])a])b \rightarrow_i \beta (\lambda x[xa])b$

Vemos que la reducción no es otra cosa que el reemplazo textual de un parámetro formal en el cuerpo de una función por el parámetro real dado. Si una secuencia de reducciones ha llegado a su fin y no es posible realizar mas reducciones, decimos que el término se ha reducido a su forma normal. Uno esperaría que un término después de una serie de reducciones llegue a una forma donde no sea posible aplicar más reducciones. Pero esto no siempre es posible. Se puede ver esto con este ejemplo:

$(\lambda x[xx])(\lambda x[xx])$

Este término siempre se reduce a si mismo. Este término, como muchos otros, no tiene una forma normal.

2.4. Teorema Church-Rosser

Es posible que un término ofrezca muchas oportunidades de reducción al mismo tiempo. Para que todo el cálculo tenga sentido, es necesario que el resultado

de la computación sea independiente del orden de reducción. Es necesario expresar esta propiedad para todos los términos, no sólo para aquellos que tengan una forma normal. Esto es posible con el siguiente teorema:

Teorema 1 (Church-Rosser). *Si un término M puede reducirse (en varios pasos) a términos N y P , entonces existe un término Q al que tanto como N y P pueden reducirse (en varios pasos).*

Imagen rombo

Capítulo 3

Agda

Capítulo 4

Sintaxis

Capítulo 5

Teorema Church-Rosser

Capítulo 6

Conclusiones