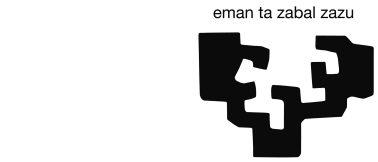

Lenguajes, Computación y Sistemas Inteligentes

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Escuela de Ingeniería de Bilbao (UPV/EHU)

2^o curso

Curso académico 2023-2024

Tema 4: λ-cálculo

JOSÉ GAINTZARAIN IBARMIA

Departamento de Lenguajes y Sistemas Informáticos

Última actualización: 01 - 10 - 2023

Índice general

4 λ-cálculo **11**

4.1 Introducción **13**

4.2 λ-expresiones en el lenguaje de programación Haskell **17**
4.2.1 λ-expresiones a modo de funciones anónimas en Haskell 17
4.2.2 λ-expresiones para definir funciones con nombre en Haskell 18

4.3 λ-expresiones o λ-términos en el λ-cálculo **21**
4.3.1 Variables, abstracciones y aplicaciones: Definición 21
4.3.2 Variables, abstracciones y aplicaciones: Ejemplos 22
4.3.3 λ-expresiones que definen funciones de más de un parámetro 23
4.3.4 Apariciones libres, ligadas y vinculantes de las variables 24
4.3.4.1 Definición 24
4.3.4.2 Ejemplos 25

4.4 Computación en el λ-cálculo **27**
4.4.1 α-equivalencia 27
4.4.2 Renombramiento de todas las apariciones de una variable mediante otra variable 28
4.4.3 Sustitución de todas las apariciones libres de una variable γ por un λ-término S 34
4.4.4 α-conversión 37
4.4.4.1 α-conversión: primera versión 38
4.4.4.2 α-conversión: segunda versión 38
4.4.4.3 α-conversión: comparación de las dos versiones 40
4.4.4.4 La α-conversión puede ser aplicada a subtérminos 41
4.4.5 β-reducción 41
4.4.5.1 Regla de la β-reducción 42
4.4.5.2 Ejemplos de β-reducción 42
4.4.5.3 La β-reducción puede ser aplicada a subtérminos 44
4.4.5.4 La β-reducción en términos con dos o más parámetros 44
4.4.6 Forma β-normal 47
4.4.6.1 Definición de la forma β-normal 47
4.4.6.2 Ejemplos sobre la forma β-normal 47
4.4.7 Cálculo de la forma β-normal 49
4.4.7.1 Procedimiento para calcular la forma β-normal 49

4.4.7.2 λ-términos que no tienen forma β-normal 50
4.4.7.3 Todas las formas β-normales de un término E son α-equivalentes . . . 51
4.4.7.4 Estrategias para calcular la forma β-normal de un λ-término E 53
4.4.8 η-reducción 56
4.4.8.1 La regla de la η-reducción 56
4.4.8.2 Regla de la β-reducción 56
4.4.8.3 Significado de la η-reducción 57
4.4.8.4 Ejemplos de η-reducción 57
4.4.8.5 Se puede aplicar η-reducción a subtérminos 60
4.4.9 Forma $\beta\eta$-normal 61
4.4.9.1 Definición de la forma $\beta\eta$-normal 61
4.4.10 Cálculo de la forma $\beta\eta$-normal 62
4.4.10.1 Procedimiento para calcular la forma $\beta\eta$-normal 62
4.4.10.2 λ-términos que no tienen forma $\beta\eta$-normal 62
4.4.11 Noción de computación en el λ-cálculo 63

4.5 Valores booleanos y operaciones sobre booleanos en el λ-cálculo 65
4.5.1 Valores booleanos y elección entre dos opciones 65
4.5.2 Elementos y normas para escribir fórmulas de la lógica proposicional 65
4.5.3 Significado o valor de las fórmulas de la lógica proposicional 67
4.5.4 Equivalencias en lógica proposicional 68
4.5.4.1 Definición de la noción de equivalencia 68
4.5.4.2 Leyes algebraicas de Bool 68
4.5.4.3 Equivalencias relacionadas con la implicación 68
4.5.4.4 Operadores lógicos que se requieren para expresar todas las fórmulas lógicas 68
4.5.5 Los operadores T y F en el λ-cálculo: TRUE y FALSE 70
4.5.5.1 El operador T en el λ-cálculo: TRUE 70
4.5.5.2 El operador F en el λ-cálculo: FALSE 73
4.5.5.3 TRUE2 y FALSE2: relación entre las funciones TRUE y FALSE 75
4.5.6 La estructura $if\text{-}then\text{-}else$ en el λ-cálculo: ITE 76
4.5.6.1 Definición de ITE 76
4.5.6.2 ITE: ejemplos 80
4.5.7 Expresiones lógicas en el λ-cálculo 80
4.5.7.1 TRUE y FALSE son expresiones booleanas 80
4.5.7.2 Las variables serán consideradas expresiones booleanas cuando convenga: la noción de contexto (la función Contexto) 80
4.5.7.3 El caso de ITE 83
4.5.7.4 Más mecanismos para formular expresiones booleanas: los λ-términos correspondientes a los operadores lógicos 83
4.5.8 El operador $\neg$ en el λ-cálculo: NOT 85
4.5.8.1 NOT: definición 85
4.5.8.2 Ejemplos de uso de NOT 86

4.5.8.3 Cuando NOT recibe un dato que no es ni TRUE ni FALSE 87
4.5.9 El operador $\wedge$ en el λ-cálculo: AND 88
4.5.9.1 AND: definición 88
4.5.9.2 Ejemplos de uso de AND 89
4.5.9.3 Cuando AND recibe datos que no son ni TRUE ni FALSE 90
4.5.10 El operador $\vee$ en el λ-cálculo: OR 91
4.5.10.1 OR: definición 91
4.5.11 El operador $\rightarrow$ en el λ-cálculo: IMP 92
4.5.11.1 IMP: definición 92
4.5.12 El operador $\leftrightarrow$ en el λ-cálculo: EQUI 94
4.5.12.1 EQUI: definición 94

4.6 Números naturales y operaciones sobre naturales en el λ-cálculo 105
4.6.1 Definición algebraica de los números naturales: $Cero$ y S 105
4.6.2 Definición de los números naturales en el λ-cálculo 107
4.6.3 El operador lógico F y el número cero son iguales en λ-cálculo: NOT y $\mathbb{0}$ 109
4.6.4 El número siguiente con respecto a un número dado: SIG 109
4.6.4.1 SIG: definición 109
4.6.4.2 Ejemplos de uso de SIG 110
4.6.5 La suma en el λ-cálculo: SUMAR y SUMAR2 110
4.6.5.1 Definición de SUMAR 110
4.6.5.2 Ejemplos de uso de SUMAR 112
4.6.5.3 Otra opción para definir la suma: SUMAR2 113
4.6.6 La multiplicación en el λ-cálculo: MULT, MULT2 y MULT3 113
4.6.6.1 Definición de MULT 113
4.6.6.2 Segunda opción: MULT2 116
4.6.6.3 Ejemplo de uso de MULT 117
4.6.6.4 Ejemplo de uso de MULT2 117
4.6.6.5 Otra opción para definir la multiplicación: MULT3 119
4.6.7 Exponenciación en el λ-cálculo: EXP y EXP2 119
4.6.7.1 Definición de EXP 119
4.6.7.2 Ejemplos de uso de EXP 121
4.6.7.3 EXP2: otra opción para definir la exponenciación 121
4.6.8 El predecesor de un número en λ-cálculo: PRED 125
4.6.8.1 Definición de PRED 125
4.6.8.2 Ejemplo de uso de PRED 126
4.6.9 La resta en el λ-cálculo: RESTAR 126
4.6.9.1 Definición de RESTAR 126
4.6.10 Decidir si un número es cero en el λ-cálculo: ES_CERO 130
4.6.10.1 Definición de ES_CERO 130
4.6.11 La comparación 'menor o igual' en el λ-cálculo: MIG 130
4.6.11.1 Definición de MIG 130
4.6.12 La igualdad entre números en el λ-cálculo: IGUAL 131

4.6.12.1 Definición de IGUAL 131

4.7 Recursividad en el λ-cálculo **133**
4.7.1 La recursividad directa no es posible en el λ-cálculo 133
4.7.2 Funciones recursivas en el λ-cálculo: la función Y 136
4.7.3 Suma de los números del intervalo $[0..x]$, utilizando la función Y 136
4.7.3.1 La función *sumatorio_aux*, utilizando la función Y 136
4.7.3.2 La función *sumatorio* 139
4.7.4 Factorial de x, utilizando la función Y 139
4.7.4.1 La función *factorial_aux*, utilizando la función Y 139
4.7.4.2 La función *factorial* 141

4.8 Tuplas en el λ-cálculo **143**
4.8.1 Pares ordenados en el λ-cálculo: PAR 143

4.9 Listas en el λ-cálculo **145**
4.9.1 Estructura recursiva de las listas 145
4.9.2 Lista vacía: VACIA 146
4.9.3 Ejemplos de listas 146
4.9.4 Operaciones sobre listas 147
4.9.4.1 Operación que decide si una lista es vacía: ES_VACIA 147
4.9.4.2 Primer elemento y resto de la lista: TRUE y FALSE 147
4.9.4.3 Suma de los elementos de una lista numérica: SUMA_LISTA 147
4.9.4.3.1 La función *suma_lista_aux* utilizando la función Y 148
4.9.4.3.2 La función *suma_lista* 155

4.10 La función universal en el λ-cálculo **157**
4.10.1 El concepto de función universal 157
4.10.2 λ-término universal: la función universal en el λ-cálculo 158
4.10.2.1 U, U_2, U_3, U_4, ... 158
4.10.2.2 $U_{\mathbb{N}}$ y $U'_{\mathbb{N}}$ 159

4.11 Resumen **163**
4.11.1 Términos del λ-cálculo y clasificación de variables 163
4.11.2 Recursividad en el λ-cálculo: función Y 163
4.11.3 Pares ordenados en el λ-cálculo: PAR 163
4.11.4 Listas en el λ-cálculo 163

4.12 Símbolos griegos **167**

Índice de figuras

4.1.1 λ-cálculo y $\lambda^{\rightarrow}$-cálculo: λ-cálculo sin tipos y λ-cálculo de tipado simple. . . . 14

4.5.1 Ejemplo de uso de la función TRUE2. 77
4.5.2 Ejemplo de uso de la función FALSE2. 78
4.5.3 Ejemplo de uso de la función ITE para el caso en el que el primer parámetro es TRUE. 81
4.5.4 Ejemplo de uso de la función ITE para el caso en el que el primer parámetro es FALSE. 82
4.5.5 Ejemplo de uso de la función ITE para el caso en el que solo se le da un dato s en vez de tres datos. 84
4.5.6 Desarrollo del término (ITE s) cuando el valor de Contexto(s) es TRUE. 84
4.5.7 Desarrollo del término (ITE s) cuando el valor de Contexto(s) es FALSE. 96
4.5.8 Cálculo de la forma β-normal correspondiente al término (NOT TRUE). 97
4.5.9 Cálculo de la forma β-normal correspondiente al término (NOT FALSE). 98
4.5.10 Cálculo de la forma β-normal correspondiente al término (NOT $(\lambda j.j)$). . . . 99
4.5.11 Cálculo de la forma β-normal correspondiente al término (NOT $(\lambda j.r)$). . . . 100
4.5.12 Cálculo de la forma β-normal correspondiente al término (NOT $((\lambda x.((xx)y))(\lambda x.((xx)y)))$.101
4.5.13 Cálculo de la forma β-normal correspondiente al término ((AND TRUE)TRUE).102
4.5.14 Cálculo de la forma β-normal correspondiente al término ((AND FALSE)TRUE).103
4.5.15 Cálculo de la forma β-normal correspondiente al término ((AND $(\lambda z.z)$)$(\lambda x.((xx)y))$).104

4.6.1 Cálculo de la forma β-normal correspondiente al término (SIG $\mathbb{2}$). 111
4.6.2 Cálculo de la forma β-normal correspondiente al término ((SUMAR $\mathbb{2}$)$\mathbb{3}$). . . . 114
4.6.3 Cálculo de la forma β-normal correspondiente al término ((SUMAR2 $\mathbb{2}$)$\mathbb{3}$). . . . 115
4.6.4 Cálculo de la forma β-normal correspondiente al término ((MULT $\mathbb{2}$)$\mathbb{3}$). 118
4.6.5 Cálculo de la forma β-normal correspondiente al término ((MULT2 $\mathbb{2}$)$\mathbb{3}$). 120
4.6.6 Primera parte del cálculo de la forma β-normal correspondiente al término ((EXP $\mathbb{2}$)$\mathbb{3}$). 122
4.6.7 Segunda parte del cálculo de la forma β-normal correspondiente al término ((EXP $\mathbb{2}$)$\mathbb{3}$). 123
4.6.8 Tercera parte del cálculo de la forma β-normal correspondiente al término ((EXP $\mathbb{2}$)$\mathbb{3}$).124
4.6.9 Primera parte del cálculo de la forma β-normal correspondiente al término (PRED $\mathbb{3}$). 127

4.6.10 Segunda parte del cálculo de la forma β-normal correspondiente al término ($\textsc{pred}\ \mathbb{3}$). 128
4.6.11 Tercera parte del cálculo de la forma β-normal correspondiente al término ($\textsc{pred}\ \mathbb{3}$). 129

4.7.1 Cálculo de la forma β-normal correspondiente al término ($\mathsf{Y}\ R$)($\mathsf{Y}\ R$). 137

4.9.1 Cálculo de la forma β-normal correspondiente al término ($\textsc{es\_vacia}\ \textsc{vacia}$). . 149
4.9.2 Cálculo de la forma β-normal correspondiente al término ($\textsc{es\_vacia}\ ((\textsc{par}\ \mathbb{5})((\textsc{par}\ \mathbb{9})((\textsc{par}\ \mathbb{8})\textsc{vac}$
4.9.3 El primero de la lista: cálculo de la forma β-normal correspondiente al término (((par 5)((par 9)((par 8)vacia)))true). 151
4.9.4 Resto de la lista: Primera parte del cálculo de la forma β-normal correspondiente al término (((par 5)((par 9)((par 8)vacia)))false). 152
4.9.5 Resto de la lista: Segunda parte del cálculo de la forma β-normal correspondiente al término (((par 5)((par 9)((par 8)vacia)))false). 153

4.11.1 Resumen: Términos del λ-cálculo y clasificación de las variables. 164
4.11.2 Resumen: definición de la función Y; comportamiento recursivo del término ($\mathsf{Y}\ R$). 165
4.11.3 Resumen: definición y modo de uso de la función par. 165
4.11.4 Resumen: algunas operaciones relacionadas con listas. 166

Índice de tablas

4.4.1 Primera versión de la α-conversión; está basada en el renombramiento 39
4.4.2 Segunda versión de la α-conversión; está basada en la sustitución de variables libres 39
4.4.3 La regla de la β-reducción. 64
4.4.4 La regla de la η-reducción. 64

4.5.1 Leyes algebraicas de Boole. 69
4.5.2 Algunas equivalencias relacionadas con los operadores $\rightarrow$ y $\leftrightarrow$. 69

4.7.1 Función que calcula la suma de los números naturales que van del 0 al x. Está escrita en Haskell y utiliza recursividad directa. 135
4.7.2 Función que calcula el factorial de un número natural x. Está escrita en Haskell y utiliza recursividad directa. 135

4.9.1 Cálculo de la suma de los elementos de una lista. Funciones escritas en Haskell y utilizando recursividad. 149

4.12.1 Símbolos griegos. 168

Tema 4

λ-cálculo

4.1.

Introducción

El λ-cálculo orginal fue presentado por Alonzo Church en 1934. El objetivo era establecer un sistema formal que sirviese para analizar las propiedades más generales de las funciones matemáticas. Pero en 1935, se demostró que el sistema presentado no era lógicamente consistente y por tanto no servía. En 1936, Alonzo Church consiguió aislar el fragmento que sí era lógicamente consistente y se comprobó que ese fragmento era adecuado para estudiar la computabilidad de las funciones. El mencionado fragmento es el que se utiliza hoy en día y se le denomina **λ-cálculo sin tipos**[1], puesto que al definir y utilizar las funciones, no se tienen en cuenta –ni siquiera se mencionan— los tipos de los datos y de las funciones. En 1940, Church presentó un sistema formal basado en el λ-cálculo sin tipos pero en el que sí se tienen en cuenta los tipos de los datos y de las funciones. Ese sistema recibe el nombre de **λ-cálculo de tipado simple**[2]. Como el propio nombre indica, los tipos permitidos son bastante básicos pero el sistema supone un primer paso para empezar a tratar los tipos de datos. Posteriormente, se han ido formulando más versiones del λ-cálculo que permiten utilizar tipos de datos con características más complejas. Los sistemas que permiten utilizar tipos de datos pueden ser vistos como subsistemas del λ-cálculo sin tipos. Por tanto, el λ-cálculo sin tipos es el más general.

Para dejar clara la terminología que vamos a utilizar a partir de ahora, cuando hablemos de λ-cálculo, nos referimos al λ-cálculo sin tipos. Y cuando queramos hablar del λ-cálculo de tipado simple o de cualquier otro λ-cálculo con tipos, lo indicaremos explícitamente. En concreto, el λ-cálculo de tipado simple lo abreviaremos como $\lambda^{\rightarrow}$*-cálculo*.

En la figura 4.1.1 de la página 14, se muestra gráficamente la relación entre el λ-cálculo (λ-cálculo sin tipos) y el $\lambda^{\rightarrow}$-cálculo (λ-cálculo de tipado simple).

La expresividad del λ-cálculo sin tipos es mayor, pero el tener en cuenta los tipos permite expresar y probar con más facilidad algunas propiedades de las funciones y, además, los tipos pueden ser utilizados para clasificar funciones.

[1] En inglés, *Untyped λ-calculus*.
[2] En inglés, *Simply typed λ-calculus*.

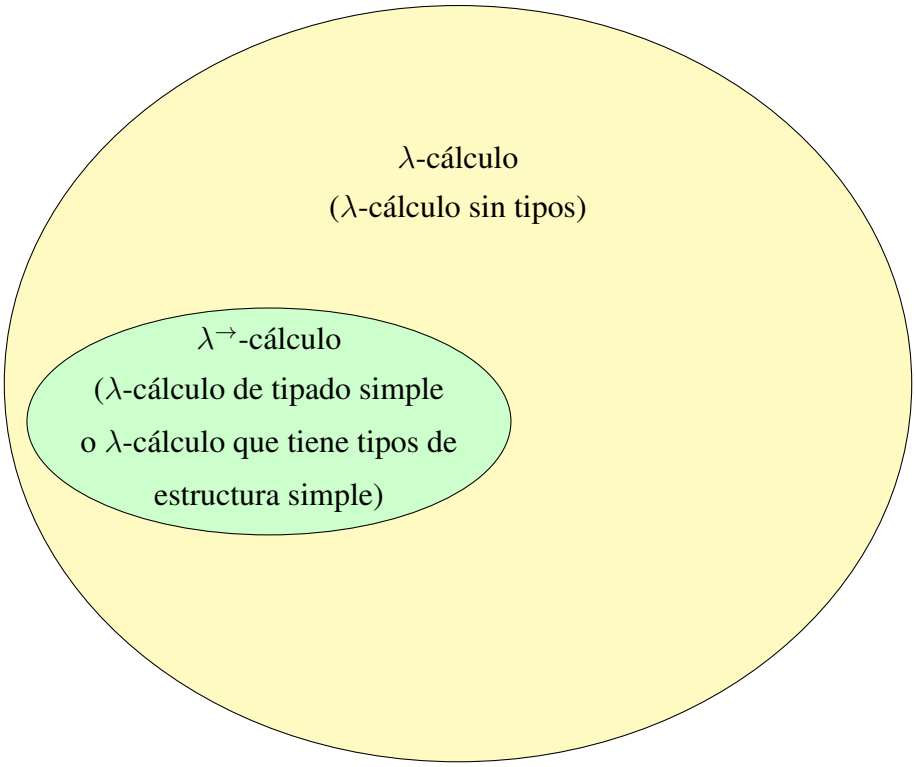

Figura 4.1.1. λ-cálculo y $\lambda^{\rightarrow}$-cálculo: λ-cálculo sin tipos y λ-cálculo de tipado simple.

Hoy en día, el λ-cálculo se utiliza a modo de modelo computacional matemático abstracto. Una de las aportaciones más destacables del λ-cálculo es el haber servido de fundamento teórico para la definición de los lenguajes de programación basados en funciones —o lenguajes de programación funcionales. De hecho, el propio λ-cálculo es considerado como el primer lenguaje de programación funcional. También se ha dicho que es el lenguaje de programación más pequeño y ha sido calificado como un lenguaje de programación minimalista. El resto de lenguajes de programación funcionales se obtienen añadiendo facilidades sintácticas al λ-cálculo. Esas facilidades sintácticas reciben el nombre de "azúcar sintáctico" [3]. El azúcar sintáctico no aporta más capacidad de cálculo pero facilita al programador la manera de escribir programas. Lisp, ML, Miranda y Haskell son algunos de los lenguajes de programación más clásicos, pero hay más.

El λ-cálculo, aparte de ser el fundamento teórico de la programación funcional, también se utiliza para establecer y analizar las características de otros lenguajes de programación no funcionales. La utilidad del λ-cálculo no se limita a las matemáticas y a la ciencia de la computación: se utiliza en áreas como lingüistica y filosofía entre otras.

Cualquier función computable puede ser expresada y evaluada mediante el λ-cálculo. Además, el λ-cálculo sirve para averiguar si una función es computable o incomputable. Por todo ello, se dice que es un lenguaje universal. En cuanto a expresividad y capacidad de cálculo, es equivalente al modelo de computación de las máquinas de Turing.

[3] En inglés, "Syntactic sugar".

4.2.

λ-expresiones en el lenguaje de programación Haskell

El lenguaje de programación Haskell permite utilizar expresiones similares a las que se utilizan en el λ-cálculo. Debido al azúcar sintáctico que incorpora Haskell, las λ-expresiones escritas en Haskell pueden resultar más entendibles y, por ello, antes de adentrarse en el λ-cálculo, puede venir bien conocer la manera de definir λ-expresiones en Haskell y el significado de esas λ-expresiones.

4.2.1 λ-expresiones a modo de funciones anónimas en Haskell

En Haskell, las λ-expresiones sirven para definir funciones anónimas. El símbolo λ se representa mediante el símbolo $\backslash$.

He aquí algunos ejemplos escritos en Haskell:

- $\backslash x \mathrel{->} (x + 1)$
  Función que, dado un número x, devuelve el valor $x + 1$. Para aplicar esa función al número 5, hay que escribir lo siguiente:

  $$(\backslash x \mathrel{->} (x + 1))\ 5$$

  El resultado será 6.

- $\backslash x\ y \mathrel{->} (x + y)$
  Función que, dados dos números x e y, devuelve el valor $x + y$. Para aplicar esa función a los números 5 y 3, hay que escribir lo siguiente:

  $$(\backslash x\ y \mathrel{->} (x + y))\ 5\ 3$$

  El resultado será 8.

- $\backslash x\ y\ z -> (x + (y * z))$
  Función que, dados tres números x, y y z, devuelve el valor $x + (y * z)$. Para aplicar esa función a los números 8, 5 y 2, hay que escribir lo siguiente:

  $$(\backslash x\ y\ z -> (x + (y * z)))\ 8\ 5\ 2$$

  El resultado será 18.

- $\backslash x\ y -> (x\ \&\&\ y)$
  Función que, dados dos valores booleanos x e y, devuelve el valor de la conjunción lógica de x e y. Para aplicar esa función a los valores booleanos $True$ y $False$, hay que escribir lo siguiente:

  $$(\backslash x\ y -> (x\ \&\&\ y))\ True\ False$$

  El resultado será $False$.

- $\backslash x\ y -> (if\ x > y\ then\ x + 1\ else\ y + 1)$
  Función que, dados dos números x e y, si x es mayor que y, entonces devuelve $x + 1$ y si no, devuelve $y + 1$. Para aplicar esa función a los números 5 y 3, hay que escribir lo siguiente:

  $$(\backslash x\ y -> (if\ x > y\ then\ x + 1\ else\ y + 1))\ 5\ 3$$

  El resultado será 6, es decir, $5 + 1$.

  La instrucción if de Haskell tiene el formato $if - then - else$. La rama $else$ no puede ser eliminada y tampoco es posible poner más de un $else$. Por tanto, solo se admiten dos opciones. Pero, sí es posible anidar un if en la rama $then$ y en la rama $else$:

  $$\backslash x\ y\ z -> (if\ x > y\ then\ x + 1\ else\ (if\ y > z\ then\ x + 2\ else\ x + 3))$$

  Para aplicar esa función a los números 5, 9 y 10, hay que escribir lo siguiente:

  $$(\backslash x\ y\ z -> (if\ x > y\ then\ x + 1\ else\ (if\ y > z\ then\ x + 2\ else\ x + 3)))\ 5\ 9\ 10$$

  El resultado será 8, es decir, $5 + 3$.

4.2.2 λ-expresiones para definir funciones con nombre en Haskell

En el lenguaje Haskell, también es posible utilizar λ-expresiones para definir funciones con nombre:

- Supongamos que queremos definir la función $resta$ que calcula la resta entre dos números. Esa función puede ser definida mediante una λ-expresión:

  $$resta = (\backslash x\ y -> (x - y))$$

Para aplicar la función $resta$ a los números 12 y 7, hay que escribir lo siguiente:

$$resta\ 12\ 7$$

El resultado será 5.

- Supongamos que queremos definir la función $divisor$ que recibe dos números y decide si el primero es divisor del segundo. Esa función puede ser definida de la siguiente manera mediante una λ-expresión:

$$divisor = (\backslash x\ y -> ((y\ mod\ x) == 0))$$

Para aplicar la función $divisor$ a los números 8 y 15, hay que escribir lo siguiente:

$$divisor\ 8\ 15$$

El resultado será $False$ porque $(15\ mod\ 8) \neq 0$.

- Por último, vamos a utilizar una λ-expresión para definir la función $implicacion$ que, dados dos valores booleanos x e y, devuelve el valor de la implicación $x \rightarrow y$:

$$implicacion = (\backslash x\ y -> ((not\ x)||\ y))$$

Para aplicar la función $implicacion$ a los valores $False$ y $False$, hay que escribir lo siguiente:

$$implicacion\ False\ False$$

El resultado será $True$ porque el valor de la implicación $False \rightarrow False$ es $True$.

4.3.

λ-expresiones o λ-términos en el λ-cálculo

En el λ-cálculo, no hay azúcar sintáctico. El único tipo de datos es el tipo de las funciones. Esto significa que todos los elementos son funciones. No hay ni números, ni booleanos, ni caracteres, ni listas ni ningún otro tipo de dato. Solo el tipo de las funciones.

4.3.1 Variables, abstracciones y aplicaciones: Definición

Hay tres clases de **λ-expresiones** o **λ-términos**:

(1) **Variables**:
Una variable es una λ-expresión.

Las variables serán habitualmente letras: h, g, f, j, v, x etc. También se pueden utilizar subíndices:

$$h_0, h_1, h_2, \ldots, h_k \ldots$$

Y también se pueden utilizar superíndices:

$$h', h'', h''', h^{iv}, \ldots,$$

Cada variable representa una función, aunque no se sabe qué función es.

(2) **Abstracciones**:
Si h es una variable y E es una λ-expresión, entonces $(\lambda h.E)$ es una λ-expresión y recibe el nombre de abstracción.

Una λ-expresión de la forma $(\lambda h.E)$ representa una función. La variable h es el **parámetro** que representa el dato de entrada. Pero hay que recordar que ese parámetro h representa a su vez una función. Es decir, el dato de entrada es una función. Por otra parte, E es el cuerpo de la función y también E será una función. Una abstracción es un mecanismo

para construir o definir una función nueva: la función $(\lambda h.E)$ recibe un dato de entrada h y realiza con h la operación indicada mediante E. Por tanto, en una abstracción de la forma $(\lambda h.E)$, el dato de entrada h será una función, el cuerpo E será también una función y la abstracción entera $(\lambda h.E)$ será también una función.

(3) **Aplicaciones**:
Si E y Q son dos λ-expresiones, entonces (EQ) es una λ-expresión y recibe el nombre de aplicación.

Un λ-término de la forma (EQ) es una función. Mediante ese λ-término se indica que se aplicará la función E a la función Q. El resultado de esa aplicación será una función.

4.3.2 Variables, abstracciones y aplicaciones: Ejemplos

- La **variable** h es una λ-expresión. Las variables son las λ-expresiones más simples.

- Teniendo en cuenta que la variable h es una λ-expresión, podemos generar la **abstracción** $(\lambda h.h)$. En esa abstracción, la variable h es el parámetro y el componente E es también la variable h. Esa abstracción define la función identidad. es decir, la función que recibe como dato de entrada una función h y devuelve como resultado esa misma función h.

- Teniendo en cuenta que las variables h y g son λ-expresiones, podemos generar la **abstracción** $(\lambda h.g)$. En esa abstracción, la variable h es el parámetro y el componente E es la variable g. Esa abstracción define la función que recibe como dato de entrada una función h y devuelve como resultado la función g.

- A partir de las λ-expresiones $(\lambda h.h)$ y g, podemos generar la **aplicación** $((\lambda h.h)g)$. En esa aplicación, el componente E es la abstracción $(\lambda h.h)$ y el componente Q es la variable g. En la λ-expresión que se ha generado, se aplicará la función $(\lambda h.h)$ a la función g.

- Teniendo en cuenta que las variables h y g son λ-expresiones, podemos generar la **aplicación** (hg). En esa aplicación, la variable h hace de E y la variable g hace de Q. En la λ-expresión que se ha generado, se aplicará la función h a la función g.

- A partir de la λ-expresión h, podemos generar la **aplicación** (hh). En esa aplicación, el componente E es la variable h y el componente Q es también la variable h. En la λ-expresión que se ha generado, se aplicará la función h a la función h.

- Teniendo en cuenta que las variables h, g y f son λ-expresiones, podemos generar la **aplicación** $((hg)f)$. En esa aplicación, el componente E es la aplicación (hg) y el componente Q es la variable f. En la λ-expresión que se ha generado, se aplicará la función (hg) a la función f. Por otra parte, la λ-expresión (hg) es la función que aplica h a g.

- Teniendo en cuenta que las variables h, g y f son λ-expresiones, podemos generar también la **aplicación** $(h(gf))$. En esa aplicación, el componente E es la variable h y el componente Q es la aplicación (gf). En la λ-expresión que se ha generado, se aplicará la función h a la función (gf). Por otra parte, la λ-expresión (gf) es la función que aplica g a f. En general, las aplicaciones $((hg)f)$ y $(h(gf))$ son distintas.
- Partiendo de la λ-expresión $((\lambda h.h)g)$, podemos generar la **abstracción** $(\lambda g.((\lambda h.h)g))$. En esa abstracción, el parámetro es g, y el componente E es la aplicación $((\lambda h.h)g)$.
- Partiendo otra vez de la λ-expresión $((\lambda h.h)g)$, podemos generar la **abstracción** $(\lambda f.((\lambda h.h)g))$. En esa abstracción, el parámetro es f, y el componente E es la aplicación $((\lambda h.h)g)$.
- Puesto que la abstracción $(\lambda h.h)$ es una λ-expresión, $(\lambda g.(\lambda h.h))$ es una λ-expresión correcta. En concreto, es una **abstracción**. En esa abstracción, el parámetro es g, y el componente E es la abstracción $(\lambda h.h)$.
- A partir de la abstracción $(\lambda g.((\lambda h.h)g))$ y la abstracción $(\lambda f.((\lambda h.h)g))$, podemos generar la siguiente **aplicación**:

  $$((\lambda g.((\lambda h.h)g))(\lambda f.((\lambda h.h)g)))$$

  En esa aplicación, el componente E es la abstracción $(\lambda g.((\lambda h.h)g))$ y el componente Q es la abstracción $(\lambda f.((\lambda h.h)g))$.

4.3.3 λ-expresiones que definen funciones de más de un parámetro

Si tomamos como punto de partida una λ-expresión E, una opción para definir una función nueva es utilizar el mecanismo de la abstracción. Por ejemplo, $(\lambda h.E)$. Pero si a partir de E queremos definir una nueva función con dos parametros nuevos, tendremos que utilizar el mecanismo de la abstracción dos veces. Por ejemplo, mediante una primera abstracción generamos la λ-expresión $(\lambda h.E)$ y mediante una segunda abstracción generamos la λ-abstracción

$$(\lambda g.(\lambda h.E))$$

Dicha expresión representa una función que tiene dos parámetros: g y h. La función

$$(\lambda g.(\lambda h.E))$$

recibirá dos funciones como datos de entrada y realizará —con esos datos de entrada— el cálculo indicado por E.

Para definir funciones de tres parámetros, se utilizará la abstracción tres veces:

$$(\lambda f.(\lambda g.(\lambda h.E)))$$

En esa λ-expresión que representa una función, los elementos f, g y h son los parámetros.

Se pueden poner tantos parámetros como haga falta, añadiendo una abstracción por parámetro.

4.3.4 Apariciones libres, ligadas y vinculantes de las variables

4.3.4.1 Definición

- Diremos que una aparición de la variable h en una λ-expresión E es **ligada** si h aparece en la componente Q de un subtérmino de la forma $\lambda h.Q$.

- Diremos que una aparición de la variable h en una λ-expresión E es **vinculante** si h aparece en la componente $\lambda h.$ de un subtérmino de la forma $\lambda h.Q$. La denominación de vinculante viene de que esa aparición de h va a hacer que todas las apariciones de h en Q sean ligadas —salvo las apariciones vinculantes.

- Diremos que una aparición de la variable h en una λ-expresión E es **libre** si dicha aparición de h no es ni ligada ni vinculante.

El **conjunto formado por todas las variables** que aparecen en una λ-expresión E, lo denotaremos como $\mathsf{variables}(E)$. Ese conjunto se define inductivamente teniendo en cuenta la estructura de E:

(1) Caso de la variables: $\mathsf{variables}(\delta) = \{\delta\}$, siendo δ una variable (h, g, etc.).

(2) Caso de las abstracciones: $\mathsf{variables}(\lambda\delta.Q) = ((\mathsf{variables}(Q)) \cup \{\delta\})$, siendo δ una variable y Q un λ-término.

(3) Caso de las aplicaciones: $\mathsf{variables}(QR) = ((\mathsf{variables}(Q)) \cup (\mathsf{variables}(R)))$, siendo Q y R dos λ-términos cualesquiera.

El **conjunto formado por todas las variables libres** que aparecen en una λ-expresión E, lo denotaremos como $\mathsf{libres}(E)$. Ese conjunto se define inductivamente teniendo en cuenta la estructura de E:

(1) Caso de la variables: $\mathsf{libres}(\delta) = \{\delta\}$, siendo δ una variable (h, g, etc.).

(2) Caso de las abstracciones: $\mathsf{libres}(\lambda\delta.Q) = ((\mathsf{libres}(Q)) \setminus \{\delta\})$, siendo δ una variable y Q un λ-término.

(3) Caso de las aplicaciones: $\mathsf{libres}(QR) = ((\mathsf{libres}(Q)) \cup (\mathsf{libres}(R)))$ siendo Q y R dos λ-términos cualesquiera.

Puede ocurrir que en una misma λ-expresión E haya apariciones ligadas, apariciones vinculantes y apariciones libres de una misma variable δ.

4.3.4.2 Ejemplos

- En la λ-expresión h, hay una aparición libre de h, no hay ninguna aparición ligada de h y tampoco ninguna aparición vinculante de h. Por tanto, $\mathsf{libres}(h) = \{h\}$.

- En la λ-expresión $(\lambda h.h)$, la primera aparición de h —empezando por la izquierda— es vinculante y la segunda aparición es ligada. Por tanto, $\mathsf{libres}((\lambda h.h)) = \varnothing$.

- En la λ-expresión $(\lambda h.g)$, la única aparición de h es vinculante y la única aparición de g es libre. Por tanto, $\mathsf{libres}((\lambda h.g)) = \{g\}$. En este ejemplo se puede apreciar que una variable puede aparecer como vinculante y no aparecer como ligada.

- En la λ-expresión $((\lambda h.h)g)$, la primera aparición de h —empezando por la izquierda— es vinculante y la segunda aparición es ligada. Por otra parte, la única aparición de g es libre. Por tanto, $\mathsf{libres}(((\lambda h.h)g)) = \{g\}$.

- En la λ-expresión (hg), la única aparición de h es libre y la única aparición de g es también libre. Por tanto, $\mathsf{libres}((hg)) = \{h, g\}$.

- En la λ-expresión (hh), las dos apariciones de h son libres. Por tanto, $\mathsf{libres}((hh)) = \{h\}$.

- En la λ-expresión $((hg)f)$, la única aparición de h es libre, la única aparición de g es también libre y la única aparición de f es también libre. Por tanto, $\mathsf{libres}(((hg)f)) = \{h, g, f\}$.

- En la λ-expresión $(h(gf))$, la única aparición de h es libre, la única aparición de g es también libre y la única aparición de f es también libre. Por tanto, $\mathsf{libres}((h(gf))) = \{h, g, f\}$.

- En la λ-expresión $(\lambda g.((\lambda h.h)g))$, la primera aparición de h —empezando por la izquierda— es vinculante y la segunda aparición es ligada. De la misma forma, la primera aparición de g —empezando por la izquierda— es vinculante y la segunda aparición es ligada. Consecuentemente, $\mathsf{libres}((\lambda g.((\lambda h.h)g))) = \varnothing$.

- En la λ-expresión $(\lambda f.((\lambda h.h)g))$, la primera aparición de h —empezando por la izquierda— es vinculante y la segunda aparición es ligada. Por otra parte, la única aparición de g es libre. Por último, la única aparición de f es vinculante. Por ende, $\mathsf{libres}((\lambda f.((\lambda h.h)g))) = \{g\}$.

- En la λ-expresión $(\lambda g.(\lambda h.h))$, la primera aparición de h —empezando por la izquierda— es vinculante y la segunda aparición es ligada. Por otra parte, la única aparición de g es vinculante. Esto conlleva que $\mathsf{libres}((\lambda g.(\lambda h.h))) = \varnothing$.

- En la λ-expresión $((\lambda g.((\lambda h.h)g))(\lambda f.((\lambda h.h)g)))$, la primera aparición de h y la tercera aparición de h —empezando por la izquierda— son vinculantes. Por otra parte, la segunda aparición y la cuarta aparición de h son ligadas. Por otra parte, la primera aparición de g es vinculante, la segunda es ligada y la tercera es libre. Finalmente, la única aparición

de f es vinculante. En total, tenemos que $\mathsf{libres}(((\lambda g.((\lambda h.h)g))(\lambda f.((\lambda h.h)g)))) = \{g\}$. En este ejemplo se puede apreciar que una variable puede aparecer como vinculante, ligada y libre en el mismo λ-término.

4.4.

Computación en el λ-cálculo

En el λ-cálculo, la computación consiste en ir transformando las λ-expresiones. Hay tres mecanismos o reglas para transformar las λ-expresiones:

(1) α-conversión: sirve para modificar el nombre de las variables vinculantes y las variables ligadas.

(2) β-reducción: sirve para aplicar una función a un argumento.

(3) η-reducción: cuando se tienen un λ-término cuya estructura cumple una determinada condición, la η-reducción permite sustituir ese λ-término por otro más simple pero que hace lo mismo a efectos de computación.

Antes de definir en detalle la α-conversión, la β-reducción y la η-reducción, se van presentar los siguientes conceptos:

- α-equivalencia.
- Renombramiento, en un λ-término, de todas las apariciones de una variable mediante otra variable.
- Sustitución, en un λ-término, de todas las apariciones libres de una variable por un λ-término.

4.4.1 α-equivalencia

Dados dos λ-términos E y Q, se puede decir que son iguales si solo difieren en el nombre de las variables vinculantes y las variables ligadas. Si es así, se dice que E y Q son **α-equivalentes**.

Por ejemplo, el término $(\lambda h.(\lambda g.(hg)))$ y el término $(\lambda e.(\lambda f.(ef)))$ son α-equivalentes. Esos términos solo difieren en los nombres de las variables vinculantes y las variables ligadas: donde en el primer término aparece h, en el segundo aparece e y, donde en el primer término aparece g, en el segundo aparece f.

Las variables libres han de tener el mismo nombre. Por ejemplo, la expresión $(\lambda h.(\lambda g.((hg)d)))$ y la expresión $(\lambda e.(\lambda f.((ef)d)))$ son α-equivalentes: donde en la primera expresión aparece h, en la segunda aparece e y, donde en la primera expresión aparece g, en la segunda aparece f, pero la variable libre d aparece en ambas expresiones. En cambio, la expresión $(\lambda h.(\lambda g.((hg)d)))$ y la expresión $(\lambda e.(\lambda f.((ef)j)))$ no son α-equivalentes porque donde en la primera expresión aparece la variable libre d, en la segunda expresión aparece la variable libre j.

4.4.2 Renombramiento de todas las apariciones de una variable mediante otra variable

Dadas una λ-expresión E y dos variables γ y δ, la operación que consiste en renombrar o sustituir en E todas las apariciones —tanto libres, como vinculantes y ligadas— de la variable γ mediante la variable δ se representa como $(E\{\delta/\gamma\})$. Por tanto, el significado de $(E\{\delta/\gamma\})$ es el siguiente:

> En la expresión E, se pondrá la variable δ en vez de la variable γ, en todas las apariciones de γ.

La **definición inductiva** de la expresión $(E\{\delta/\gamma\})$ es la siguiente:

(B1) Si la expresión E es la variable γ:
$(\gamma\{\delta/\gamma\}) \equiv \delta$.

(B2) Si la expresión E es una variable π distinta a γ:
$(\pi\{\delta/\gamma\}) \equiv \pi$.

(B3) Si la expresión E es una λ-expresión de la forma (QR), siendo Q y R dos λ-términos:
$((QR)\{\delta/\gamma\}) \equiv ((Q\{\delta/\gamma\})(R\{\delta/\gamma\}))$.

(B4) Si la expresión E es una λ-expresión de la forma $(\lambda\gamma.Q)$, siendo Q un λ-término:
$((\lambda\gamma.Q)\{\delta/\gamma\}) \equiv (\lambda\delta.(Q\{\delta/\gamma\}))$.

(B5) Si la expresión E es una λ-expresión de la forma $(\lambda\pi.Q)$, siendo π una variable distinta de γ y siendo Q un λ-término:
$((\lambda\pi.Q)\{\delta/\gamma\}) \equiv (\lambda\pi.(Q\{\delta/\gamma\}))$.

Tras realizar el renombramiento de una variable de una expresión E, puede ocurrir que el término obtenido sea α-equivalente a E o que no sea α-equivalente a E.

Ejemplo 4.4.2.1. *A continuación se indican los pasos a dar para renombrar las apariciones de la variable h con la variable f en la expresión $(\lambda h.(hg))$:*

- $\underbrace{(((\lambda h.(hg)))\{f/h\})}_{\textit{(B4)}}$
- $(\lambda f.\underbrace{((hg)\{f/h\})}_{\textit{(B3)}})$
- $(\lambda f.((h\{f/h\})\,\underbrace{(g\{f/h\})}_{\textit{(B2)}}))$
- $(\lambda f.(\underbrace{(h\{f/h\})}_{\textit{(B1)}}\,g))$
- $(\lambda f.(fg))$

La expresión obtenida es α-equivalente a la expresión inicial $(\lambda h.(hg))$.

Ejemplo 4.4.2.2. *Consideramos otra vez la expresión $(\lambda h.(hg))$. Los pasos a dar para renombrar las apariciones de la variable h con la variable g son los siguientes:*

- $\underbrace{(((\lambda h.(hg)))\{g/h\})}_{\textit{(B4)}}$
- $(\lambda g.\underbrace{((hg)\{g/h\})}_{\textit{(B3)}})$
- $(\lambda g.((h\{g/h\})\,\underbrace{(g\{g/h\})}_{\textit{(B2)}}))$
- $(\lambda g.(\underbrace{(h\{g/h\})}_{\textit{(B1)}}\,g))$
- $(\lambda g.(gg))$

La expresión obtenida no es α-equivalente a la expresión inicial $(\lambda h.(hg))$. El motivo es que en la nueva expresión resultante $(\lambda g.(gg))$, no hay ninguna aparición libre de la variable g pero en la expresión inicial $(\lambda h.(hg))$, hay una aparición libre de la variable g. El renombramiento $((\lambda h.(hg))\{g/h\})$ ha hecho que la aparición libre de g de la expresión inicial se convierta en una aparición ligada.

Ejemplo 4.4.2.3. *Consideremos por tercera vez la expresión* $(\lambda h.(hg))$. *En esta ocasión se muestran los pasos correspondientes al proceso de sustitución —o renombramiento— de todas las apariciones de* g *utilizando la variable* h*:*

- $\underbrace{(((\lambda h.(hg)))\{h/g\})}_{\textit{(B5)}}$
- $(\lambda h.\underbrace{((hg)\{h/g\})}_{\textit{(B3)}})$
- $(\lambda h.((h\{h/g\})\underbrace{(g\{h/g\})}_{\textit{(B1)}}))$
- $(\lambda h.(\underbrace{(h\{h/g\})}_{\textit{(B2)}}\, h))$
- $(\lambda h.(hh))$

La expresión obtenida no es α*-equivalente a la expresión inicial* $(\lambda h.(hg))$. *De nuevo, el motivo es que en la expresión obtenida* $(\lambda h.(hh))$*, no hay ninguna aparición libre de la variable* g *mientras que en la expresión inicial* $(\lambda h.(hg))$*, hay una aparición libre de la variable* g*. El renombramiento* $((\lambda h.(hg))\{h/g\})$ *ha hecho que la aparición libre de* g *de la expresión inicial desaparezca.*

Ejemplo 4.4.2.4. *Retomamos por cuarta vez la expresión* $(\lambda h.(hg))$. *Los pasos a dar para renombrar las apariciones de la variable* g *con la variable* f *son los siguientes:*

- $\underbrace{(((\lambda h.(hg)))\{f/g\})}_{\textit{(B5)}}$
- $(\lambda h.\underbrace{((hg)\{f/g\})}_{\textit{(B3)}})$
- $(\lambda h.((h\{f/g\})\underbrace{(g\{f/g\})}_{\textit{(B1)}}))$
- $(\lambda h.(\underbrace{(h\{f/g\})}_{\textit{(B2)}}\, f))$
- $(\lambda h.(hf))$

La expresión obtenida no es α-equivalente a la expresión inicial $(\lambda h.(hg))$. El motivo es que en la nueva expresión resultante $(\lambda h.(hf))$, no hay ninguna aparición libre de la variable g pero en la expresión inicial $(\lambda h.(hg))$, hay una aparición libre de la variable g. El renombramiento $((\lambda h.(hg))\{f/g\})$ ha hecho que la aparición libre de g desaparezca. En la nueva expresión $(\lambda h.(hf))$, en vez de una aparición libre de la variable g, tenemos una aparición libre de la variable f, pero la α-equivalencia requiere que las variables libres no se modifiquen.

Ejemplo 4.4.2.5. *A continuación se indican los pasos a dar para renombrar las apariciones de la variable h con la variable g en la expresión $(\lambda h.(h(\lambda h.(hf))))$:*

- $\underbrace{((\lambda h.(h(\lambda h.(hf))))\{g/h\})}_{\textit{(B4)}}$
- $(\lambda g.\underbrace{((h(\lambda h.(hf)))\{g/h\})}_{\textit{(B3)}})$
- $(\lambda g.(\underbrace{(h\{g/h\})}_{\textit{(B1)}}\underbrace{((\lambda h.(hf))\{g/h\})}_{\textit{(B4)}}))$
- $(\lambda g.(g(\lambda g.\underbrace{((hf)\{g/h\})}_{\textit{(B3)}})))$
- $(\lambda g.(g(\lambda g.(\underbrace{(h\{g/h\})}_{\textit{(B1)}}\underbrace{(f\{g/h\})}_{\textit{(B2)}}))))$
- $(\lambda g.(g(\lambda g.(gf))))$

La expresión obtenida es α-equivalente a la expresión inicial $(\lambda h.(h(\lambda h.(hf))))$.

Ejemplo 4.4.2.6. *A continuación, se muestra el proceso de renombramiento de la variable h con la variable g en la expresión $(h(\lambda h.(h(\lambda h.(hf)))))$:*

- $\underbrace{((h(\lambda h.(h(\lambda h.(hf)))))\{g/h\})}_{\textit{(B3)}}$
- $(\underbrace{(h\{g/h\})}_{\textit{(B1)}}((\lambda h.(h(\lambda h.(hf))))\{g/h\}))$
- $(g\underbrace{((\lambda h.(h(\lambda h.(hf))))\{g/h\})}_{\textit{(B4)}})$
- $(g(\lambda g.\underbrace{((h(\lambda h.(hf)))\{g/h\})}_{\textit{(B3)}}))$

- $(g(\lambda g.(\underbrace{(h\{g/h\})}_{\textit{(B1)}}\underbrace{((\lambda h.(hf))\{g/h\})}_{\textit{(B4)}})))$
- $(g(\lambda g.(g(\lambda g.\underbrace{((hf)\{g/h\})}_{\textit{(B3)}})))))$
- $(g(\lambda g.(g(\lambda g.(\underbrace{(h\{g/h\})}_{\textit{(B1)}}\underbrace{(f\{g/h\})}_{\textit{(B2)}})))))$
- $(g(\lambda g.(g(\lambda g.(gf)))))$

La expresión obtenida no es α-equivalente a la expresión original $(h(\lambda h.(h(\lambda h.(hf)))))$. El motivo es que en la nueva expresión $(g(\lambda g.(g(\lambda g.(gf)))))$, no hay ninguna aparición libre de la variable h pero en la expresión inicial hay una aparición libre de la variable h.

Ejemplo 4.4.2.7. *En la expresión $((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))$, la variable h aparece libre una vez, en la primera aparición desde la izquierda. Además, aparece dos veces como vinculante y dos veces como ligada. El ámbito de la primera h vinculante es la expresión $((hg)(\lambda h.(((fg)h)g)))$, pero en esa expresión, h vuelve a aparecer como vinculante y el ámbito de esa segunda aparición vinculante de h es $(((fg)h)g)$. En realidad, el significado de la expresión $((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))$ y el significado de la expresión*

$$((jh)(\lambda h_1.((h_1g)(\lambda h_2.(((fg)h_2)g)))))$$

es el mismo. Por tanto, es como si tuviésemos tres variables: h como libre, h_1 como vinculante y h_2 como vinculante.

Se va a renombrar la variable h con la variable v:

- $\underbrace{(((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))\{v/h\})}_{\textit{(B3)}}$
- $(\underbrace{((jh)\{v/h\})}_{\textit{(B3)}}\underbrace{((\lambda h.((hg)(\lambda h.(((fg)h)g)))))\{v/h\})}_{\textit{(B4)}})$
- $((\underbrace{(j\{v/h\})}_{\textit{(B2)}}\underbrace{(h\{v/h\})}_{\textit{(B1)}})(\lambda v.\underbrace{(((hg)(\lambda h.(((fg)h)g))))\{v/h\})}_{\textit{(B3)}}))$
- $((jv)(\lambda v.((\underbrace{(hg)\{v/h\})}_{\textit{(B3)}}\underbrace{((\lambda h.(((fg)h)g))\{v/h\})}_{\textit{(B4)}})))$
- $((jv)(\lambda v.((\underbrace{(h\{v/h\})}_{\textit{(B1)}}\underbrace{(g\{v/h\})}_{\textit{(B2)}})\underbrace{((\lambda h.(((fg)h)g)))\{v/h\})}_{\textit{(B4)}})))$

- $((jv)(\lambda v.((vg)((\lambda v.\underbrace{(((fg)h)g)))}_{\textit{(B3)}}\{v/h\}))))$
- $((jv)(\lambda v.((vg)(\lambda v.(\underbrace{(((fg)h)\{v/h\})}_{\textit{(B3)}}\underbrace{(g\{v/h\})}_{\textit{(B2)}}))))))$
- $((jv)(\lambda v.((vg)(\lambda v.(\underbrace{(((fg)\{v/h\})}_{\textit{(B3)}}\underbrace{(h\{v/h\})}_{\textit{(B1)}})g)))))$
- $((jv)(\lambda v.((vg)(\lambda v.((\underbrace{((f\{v/h\})}_{\textit{(B2)}}\underbrace{(g\{v/h\}))}_{\textit{(B2)}}v)g)))))$
- $((jv)(\lambda v.((vg)(\lambda v.(((fg)v)g)))))$

La expresión obtenida no es α-equivalente a la expresión original

$$((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))$$

porque en el nuevo término, la variable h no aparece libre. En el sitio donde debiera aparecer h como libre, aparece v como libre.

Ejemplo 4.4.2.8. *Consideramos otra vez la expresión $((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))$. En esa expresión, las tres apariciones de g son libres y la única aparición de f es también libre.*

A continuación, se muestra el proceso de renombramiento de la variable g con la variable f en la expresión $((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))$:

- $\underbrace{(((jh)(\lambda h.((hg)(\lambda h.(((fg)h)g)))))\{f/g\})}_{\textit{(B3)}}$
- $(\underbrace{((jh)\{f/g\})}_{\textit{(B3)}}\underbrace{((\lambda h.((hg)(\lambda h.(((fg)h)g)))))\{f/g\})}_{\textit{(B5)}})$
- $((\underbrace{(j\{f/g\})}_{\textit{(B2)}}\underbrace{(h\{f/g\})}_{\textit{(B2)}})(\lambda h.\underbrace{(((hg)(\lambda h.(((fg)h)g))))\{f/g\})}_{\textit{(B3)}}))$
- $((jh)(\lambda h.(\underbrace{((hg)\{f/g\})}_{\textit{(B3)}}\underbrace{((\lambda h.(((fg)h)g))\{f/g\})}_{\textit{(B5)}})))$
- $((jh)(\lambda h.((\underbrace{(h\{f/g\})}_{\textit{(B2)}}\underbrace{(g\{f/g\})}_{\textit{(B1)}})((\lambda h.\underbrace{(((fg)h)g)\{f/g\})}_{\textit{(B3)}})))))$
- $((jh)(\lambda h.((hf)((\lambda h.\underbrace{(((fg)h)g))))\{f/g\})}_{\textit{(B3)}})))$

- $((jh)(\lambda h.((hf)(\lambda h.(\underbrace{(((fg)h)\{f/g\})}_{\textit{(B3)}}\underbrace{(g\{f/g\})}_{\textit{(B1)}})))))$
- $((jh)(\lambda h.((hf)(\lambda h.(\underbrace{(((fg)\{f/g\})}_{\textit{(B3)}}\underbrace{(h\{f/g\})}_{\textit{(B2)}}f)))))$
- $((jh)(\lambda h.((hf)(\lambda h.((\underbrace{(f\{f/g\})}_{\textit{(B2)}}\underbrace{(g\{f/g\})}_{\textit{(B1)}}h)f)))))$
- $((jh)(\lambda h.((hf)(\lambda h.(((ff)h)f)))))$

La expresión obtenida no es α-equivalente a la expresión original porque en el nuevo término, g no aparece libre y hay más apariciones libres de f. La α-equivalencia requiere no modificar las variables libres.

4.4.3 Sustitución de todas las apariciones libres de una variable γ por un λ-término S

Dadas dos λ-expresiones E y S y una variable γ, la operación que consiste en sustituir en E todas las apariciones libres de la variable γ por la expresión S se representa como $(E[S/\gamma])$. En esa sustitución, las variables que son libres en S han de seguir siendo libres en los sitios donde se haya colocado S dentro de E. Por tanto, el significado de $(E[S/\gamma])$ es el siguiente:

> En la expresión E, se pondrá el término S en vez de cada aparición libre de la variable γ pero, además, las variables libres de S han de seguir siendo libres en aquellos puntos de E en los que se ha insertado S.

Este tipo de sustitución solo afecta a las apariciones libres de la variable γ en cuestión. Las apariciones no libres de γ se mantienen tal cual. Recordemos que en la operación de renombramiento —presentada en el apartado anterior— todas las apariciones son modificadas.

La **definición inductiva** de la expresión $(E[S/\gamma])$ es la siguiente:

(O1) Si la expresión E es la variable γ:
$(\gamma[S/\gamma]) \equiv S$.

(O2) Si la expresión E es una variable π distinta a γ:
$(\pi[S/\gamma]) \equiv \pi$.

(O3) Si la expresión E es una λ-expresión de la forma (QR), siendo Q y R dos λ-términos:
$((QR)[S/\gamma]) \equiv ((Q[S/\gamma])(R[S/\gamma]))$.

(O4) Si la expresión E es una λ-expresión de la forma $(\lambda\gamma.Q)$, siendo Q un λ-término:
$((\lambda\gamma.Q)[S/\gamma]) \equiv (\lambda\gamma.Q)$.

(O5) Si la expresión E es una λ-expresión de la forma $(\lambda\pi.Q)$, siendo π una variable distinta de γ y siendo Q un λ-término:
$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\pi.(Q[S/\gamma]))$,
Si se cumple $\pi \notin \mathsf{libres}(S)$ o se cumple $\gamma \notin \mathsf{libres}(Q)$.

(O6) Si la expresión E es una λ-expresión de la forma $(\lambda\pi.Q)$, siendo π una variable distinta de γ y siendo Q un λ-término y siendo ρ una variable,
$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\rho.((Q\{\rho/\pi\})[S/\gamma]))$,
si se cumple $\pi \in \mathsf{libres}(S)$ y se cumple $\gamma \in \mathsf{libres}(Q)$ y se cumple $\rho \notin \mathsf{variables}(S)$ y se cumple $\rho \notin \mathsf{variables}(Q)$.

La regla (O6) puede ser formulada utilizando la operación de sustitución de una variable libre por un término:

(O6') Si la expresión E es una λ-expresión de la forma $(\lambda\pi.Q)$, siendo π una variable distinta de γ y siendo Q un λ-término y siendo ρ una variable,
$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\rho.((Q[\rho/\pi])[S/\gamma]))$,
si se cumple $\pi \in \mathsf{libres}(S)$ y se cumple $\gamma \in \mathsf{libres}(Q)$ y se cumple $\rho \notin \mathsf{libres}(S)$ y se cumple $\rho \notin \mathsf{libres}(Q)$.

Observación 4.4.3.1. *Los siguientes dos ejemplos ilustran la necesidad de la condición que se ha puesto al final del punto (O5) y la necesidad del punto (O6) —o del punto (O6'):*

- *Sea la función $(\lambda g.h)$. En esa función, g es el parámetro y h es el resultado. La variable h es libre.*

  Si calculamos el valor de la expresión $((\lambda g.h)[f/h])$ teniendo en cuenta solo la parte

  $$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\pi.(Q[S/\gamma]))$$

  de la regla (O5), obtendremos la expresión $(\lambda g.f)$. Al aplicar esa regla, consideramos $\pi = g$ y $S = f$. En la expresión que se ha obtenido mediante la sustitución de h con f, la variable f es libre y eso es lo que exige la operación de sustitución: que las apariciones libres de las variables de S sigan siendo libres en la nueva expresión.

  En cambio, si calculamos el valor de la expresión $((\lambda g.h)[g/h])$ considerando únicamente la parte

  $$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\pi.(Q[S/\gamma]))$$

  de la regla (O5), el resultado será $(\lambda g.g)$. Al aplicar esa regla, tenemos que $\pi = g$ y $S = g$. En la expresión obtenida como resultado, la variable g que aparece libre en S ya no es libre en el punto donde se ha colocado S dentro de $(\lambda g.h)$. Por tanto, no se cumple

el requerimiento de la operación de sustitución de variables libres por términos. Puesto que g es libre en S, g ha de ser libre allá donde se haya insertado S.

Pero si en la regla (O5) se exige la condición de que se cumpla $\pi \notin \mathsf{libres}(S)$ o se cumpla $\gamma \notin \mathsf{libres}(Q)$, entonces al calcular $((\lambda g.h)[g/h])$ no se puede aplicar la regla (O5). Se ha de aplicar la regla (O6). La regla (O6) viene a decir que para resolver el problema surgido, se ha de renombrar primero la variable g con una variable que no aparezca en el término inicial y, a continuación, realizar la sustitución de las apariciones libres de h por g. Si elegimos j como nueva variable, se ha de calcular lo siguiente: $(\lambda j.((h\{j/g\})[g/h]))$. Tras el renombramiento tendremos la expresión $(\lambda j.(h[g/h]))$. Y a partir de esa expresión, la sustitución genera el término $(\lambda j.g)$. Ahora sí se cumple el requerimiento de la operación de sustitución de variables libres por términos: las apariciones libres de g en S, siguen siendo libres allá donde se haya insertado S.

- *Consideramos ahora la función $(\lambda g.(f\,g))$.*

  Si calculamos el valor de la expresión $((\lambda g.(f\,g))[(\lambda j.(g\,j))/f])$ teniendo en cuenta solo la parte

  $$((\lambda\pi.Q)[S/\gamma]) \equiv (\lambda\pi.(Q[S/\gamma]))$$

  de la regla (O5), obtendremos la expresión $(\lambda g.((\lambda j.(g\,j))g))$. Al aplicar esa regla, consideramos $\pi = g$, $\gamma = f$ y $S = (\lambda j.(g\,j))$. En la expresión que se ha obtenido mediante la sustitución de f por $(\lambda j.(g\,j))$, la variable g que aparece libre en S ya no es libre en el punto donde se ha colocado S dentro de $(\lambda g.(f\,g))$. Por tanto, no se cumple el requerimiento de la operación de sustitución de variables libres por términos. Puesto que g es libre en S, g ha de ser libre allá donde se haya insertado S.

  Pero si en la regla (O5) se exige la condición de que se cumpla $\pi \notin \mathsf{libres}(S)$ o se cumpla $\gamma \notin \mathsf{libres}(Q)$, entonces al calcular $((\lambda g.(f\,g))[(\lambda j.(g\,j))/f])$ no se puede aplicar la regla (O5). Se ha de aplicar la regla (O6). La regla (O6) viene a decir que para resolver el problema surgido, se ha de renombrar primero la variable g con una variable que no aparezca en el término inicial y, a continuación, realizar la sustitución de las apariciones libres de f por $(\lambda j.(g\,j))$. Si elegimos v como nueva variable, se ha de calcular lo siguiente: $(\lambda v.(((f\,g)\{v/g\})[(\lambda j.(g\,j))/f]))$. Tras el renombramiento tendremos la expresión $(\lambda v.((f\,v)[(\lambda j.(g\,j))/f]))$. Y a partir de esa expresión, la sustitución genera el término $(\lambda v.((\lambda j.(g\,j))v))$. Ahora sí se cumple el requerimiento de la operación de sustitución de variables libres por términos: las apariciones libres de g en S, siguen siendo libres allá donde se haya insertado S.

Ejemplo 4.4.3.2. *A continuación, se muestran los pasos necesarios para sustituir las apariciones libres de la variable h por el término $(\lambda b.a)$ en el término $(\lambda g.(h(\lambda h.((h\,f)g))))$:*

- $\underbrace{((\lambda g.(h(\lambda h.((h\,f)g))))[(\lambda b.a)/h])}_{\textit{(O5)}}$

- $(\lambda g.\underbrace{((h(\lambda h.((hf)g)))[(\lambda b.a)/h])}_{(O3)})$
- $(\lambda g.(\underbrace{(h[(\lambda b.a)/h])}_{(O1)}\underbrace{((\lambda h.((hf)g))[(\lambda b.a)/h])}_{(O4)}))$
- $(\lambda g.((\lambda b.a)(\lambda h.((hf)g))))$

Ejemplo 4.4.3.3. *Consideremos el término* $(\lambda g.(h(\lambda h.((hf)g))))$. *Los pasos correspondientes a la sustitución de las apariciones libres de la variable* h *por el término* $(\lambda b.g)$ *son los siguientes:*

- $\underbrace{((\lambda g.(h(\lambda h.((hf)g))))[(\lambda b.g)/h])}_{(O6)}$
- $(\lambda j.(\underbrace{((h(\lambda h.((hf)g)))\{j/g\})}_{(B3)}[(\lambda b.g)/h]))$
- $(\lambda j.((\underbrace{(h\{j/g\})}_{(B1)}\underbrace{((\lambda h.((hf)g))\{j/g\})}_{(B4)})[(\lambda b.g)/h]))$
- $(\lambda j.((h(\lambda h.\underbrace{(((hf)g)\{j/g\})}_{(B3)}))[(\lambda b.g)/h]))$
- $(\lambda j.((h(\lambda h.(\underbrace{((hf)\{j/g\})}_{(B3)}\underbrace{(g\{j/g\})}_{(B1)})))[(\lambda b.g)/h]))$
- $(\lambda j.((h(\lambda h.(\underbrace{(h\{j/g\})}_{(B2)}\underbrace{(f\{j/g\})}_{(B2)})j))[(\lambda b.g)/h]))$
- $(\lambda j.\underbrace{((h(\lambda h.((hf)j)))[(\lambda b.g)/h])}_{(O3)})$
- $(\lambda j.(\underbrace{(h[(\lambda b.g)/h])}_{(O1)}\underbrace{((\lambda h.((hf)j))[(\lambda b.g)/h])}_{(O4)}))$
- $(\lambda j.((\lambda b.g)(\lambda h.((hf)j))))$

4.4.4 α-conversión

Dada una λ-expresión E, a veces nos puede interesar trabajar con una λ-expresión distinta pero que sea α-equivalente a E. Es decir, nos puede interesar modificar el nombre de las variables vinculantes y las variables ligadas. Dicha modificación se puede realizar mediante la

α-conversión.

La α-conversión se aplica a abstracciones de la forma $(\lambda\gamma.Q)$ con el objetivo de poner otra variable en el lugar de la variable γ, pero sin modificar el significado del término $(\lambda\gamma.Q)$.

A continuación se presentan dos maneras de modificar el nombre de las variables vinculantes y las variables ligadas. Son dos versiones de la α-conversión.

4.4.4.1 α-conversión: primera versión

La primera versión de la α-conversión se basa en el renombramiento. En la tabla 4.4.1 de la página 39 se muestra dicha primera versión de la α-conversión. Tal como se puede apreciar en esa tabla, si tenemos un término de la forma $(\lambda\gamma.Q)$, podemos renombrar la variable γ con otra variable δ:

$$(\lambda\delta.(Q\{\delta/\gamma\}))$$

Pero para garantizar que el nuevo término tiene el mismo significado que el término de partida, la variable δ ha de ser una variable que no aparece en el término Q de partida.

Para expresar que tras aplicar al término $(\lambda\gamma.Q)$ la primera versión de la α-conversión, nos ha quedado el término $(\lambda\delta.(Q\{\delta/\gamma\}))$, escribiremos lo siguiente:

$$(\lambda\gamma.Q) \ \rightarrow_\alpha (\lambda\delta.(Q\{\delta/\gamma\}))$$

4.4.4.2 α-conversión: segunda versión

La segunda versión de la α-conversión se basa en la sustitución de variables libres. En la tabla 4.4.2 de la página 39 se muestra dicha segunda versión de la α-conversión. Tal como se puede apreciar en esa tabla, si tenemos un término de la forma $(\lambda\gamma.Q)$, podemos sustituir las apariciones libres de la variable γ en Q con otra variable δ y poner $(\lambda\delta.$ en vez de $(\lambda\gamma.$:

$$(\lambda\delta.(Q[\delta/\gamma]))$$

Pero para garantizar que el nuevo término tenga el mismo significado que el término de partida, la variable δ ha de ser una variable que no aparece libre en el término Q de partida.

Para expresar que tras aplicar al término $(\lambda\gamma.Q)$ la segunda versión de la α-conversión, nos ha quedado el término $(\lambda\delta.(Q[\delta/\gamma]))$, escribiremos lo siguiente:

$$(\lambda\gamma.Q) \ \rightarrow_\alpha (\lambda\delta.(Q[\delta/\gamma]))$$

α-conversión: primera versión, basada en el renombramiento
$(\lambda\gamma.Q) \ \rightarrow_\alpha (\lambda\delta.(Q\{\delta/\gamma\}))$ siendo δ una variable que no aparece en el λ-término Q, es decir, $\delta \notin \mathsf{variables}(Q)$.

Tabla 4.4.1. Primera versión de la α-conversión; está basada en el renombramiento

α-conversión: segunda versión, basada en la sustitución de variables libres
$(\lambda\gamma.Q) \ \rightarrow_\alpha (\lambda\delta.(Q[\delta/\gamma]))$ siendo δ una variable que no aparece libre en el λ-término Q, es decir, $\delta \notin \mathsf{libres}(Q)$.

Tabla 4.4.2. Segunda versión de la α-conversión; está basada en la sustitución de variables libres

4.4.4.3 α-conversión: comparación de las dos versiones

La primera versión de la α-conversión está basada en el renombramiento, mientras que la segunda versión está basada en la sustitución de variables libres. Independientemente de cuál de las dos versiones se utilice, el término obtenido será α-equivalente al término de partida. De todas formas, puede que el aplicar una versión u otra, nos lleve a tener términos que no sean iguales caracter a caracter, aunque sí serán α-equivalentes.

Si al término $(\lambda h.(h(\lambda h.(hf))))$ se le aplica la primera versión de la α-conversión utilizando g como variable nueva, se procederá a calcular el siguiente renombramiento:

$$((\lambda h.(h(\lambda h.(hf))))\{g/h\})$$

El término resultante será $(\lambda g.(g(\lambda g.(gf))))$. Por tanto, han desaparecido todas las apariciones de h:

$$(\lambda h.(h(\lambda h.(hf)))) \rightarrow_\alpha ((\lambda h.(h(\lambda h.(hf))))\{g/h\}) \equiv (\lambda g.(g(\lambda g.(gf))))$$

Por otro lado, si al término $(\lambda h.(h(\lambda h.(hf))))$ se le aplica la segunda versión de la α-conversión utilizando g como variable nueva, se procederá a calcular la siguiente sustitución de las apariciones libres de h en el subtérmino $(h(\lambda h.(hf)))$:

$$(\lambda g.((h(\lambda h.(hf)))[g/h]))$$

El término resultante será $(\lambda g.(g(\lambda h.(hf))))$. Solo las apariciones de h ligadas a la primera h vinculante han sido sustituidas:

$$(\lambda h.(h(\lambda h.(hf)))) \rightarrow_\alpha (\lambda g.((h(\lambda h.(hf)))[g/h])) \equiv (\lambda g.(g(\lambda h.(hf))))$$

Lo que ha ocurrido al aplicar la segunda versión, nos indica que en el término $(\lambda h.(h(\lambda h.(hf))))$ no todas las apariciones de la variable h representan un mismo valor. En realidad, hay dos valores distintos. Puesto que todas las apariciones de h son vinculantes o ligadas, pueden ser renombradas manteniendo la α-equivalencia: $(\lambda h_1.(h_1(\lambda h_2.(h_2 f))))$.

De la misma forma, en el término $(h(\lambda h.((h(\lambda h.(hf)))h))h)$, las distintas apariciones de h no representan un mismo valor. Hay tres valores distintos representados por las distintas apariciones de h. Las apariciones vinculantes y ligadas de h pueden ser renombradas pero las apariciones libres de h no pueden ser renombradas si queremos conservar la α-equivalencia:

$$(h(\lambda h_1.((h_1(\lambda h_2.(h_2 f)))h_1))h)$$

Todas las apariciones libres de h representan un mismo valor y el nombre de h no puede ser modificado. En el caso de las apariciones no libres de h, cada aparición vinculante de h tiene su propio ámbito y liga las apariciones de h que están en ese ámbito. Las apariciones vinculantes y ligadas de h pueden ser renombradas: en el ejemplo se han utilizado las variables h_1 y h_2.

4.4.4.4 La α-conversión puede ser aplicada a subtérminos

Dado un λ-término E que contiene un subtérmino de la forma $(\lambda\gamma.Q)$, es posible aplicar la α-conversión a ese subtérmino $(\lambda\gamma.Q)$.

Dado el término $(\lambda h.(h\,\underbrace{(\lambda h.(hf))}))$, si aplicamos la primera versión de la α-conversión al subtérmino $(\lambda h.(hf))$ con g como nueva variable, se calculará el siguiente renombramiento:

$$((\lambda h.(h\,\underbrace{((\lambda h.(hf))\{g/h\})}))$$

y se obtendrá el siguiente término:

$$(\lambda h.(h\,\underbrace{(\lambda g.(gf))}))$$

Por tanto, desaparecen las apariciones de h correspondientes al subtérmino $(\lambda h.(hf))$, pero las aparciones de h que están fuera de ese subtérmino se mantienen:

$$(\lambda h.(h\,\underbrace{(\lambda h.(hf))}))\ \rightarrow_\alpha ((\lambda h.(h\,\underbrace{((\lambda h.(hf))\{g/h\})}))) \equiv (\lambda h.(h\,\underbrace{(\lambda g.(gf))}))$$

Por otro lado, si en el término $(\lambda h.(h\,\underbrace{(\lambda h.(hf))}))$ aplicamos la segunda versión de la α-conversión al subtérmino $(\lambda h.(hf))$ con g como nueva variable, se calculará la siguiente sustitución:

$$((\lambda h.(h\,\underbrace{((\lambda h.(hf))[g/h])})))$$

y se obtendrá el siguiente término:

$$(\lambda h.(h\,\underbrace{(\lambda g.(gf))}))$$

Por tanto, desaparecen las apariciones de h correspondientes al subtérmino $(\lambda h.(hf))$, pero las aparciones de h que están fuera de ese subtérmino se mantienen:

$$(\lambda h.(h\,\underbrace{(\lambda h.(hf))}))\ \rightarrow_\alpha ((\lambda h.(h\,\underbrace{((\lambda h.(hf))[g/h])}))) \equiv (\lambda h.(h\,\underbrace{(\lambda g.(gf))}))$$

En este ejemplo concreto, la primera versión y la segunda versión de la α-conversión han generado el mismo término, pero, en general, pueden dar lugar a términos diferentes pero α-equivalentes, tal como se ha indicado en el apartado anterior.

4.4.5 β-reducción

La regla de la β-reducción especifíca qué termino se obtendrá al aplicar una abstracción de la forma $(\lambda\gamma.Q)$ a un λ-término R.

4.4.5.1 Regla de la β-reducción

En la tabla 4.4.3 de la página 64 se muestra la regla de la β-reducción. Tal como se puede apreciar en esa tabla, si tenemos un término de la forma $((\lambda\gamma.Q)R)$ —es decir, una aplicación— la regla de la β-reducción indica que ese término representa el término que se obtiene sustituyendo las apariciones libres de la variable γ por el término R en Q:

$$(Q[R/\gamma]))$$

Por consiguiente, si aplicamos β-reducción al término $((\lambda\gamma.Q)R)$, nos quedará el término $(Q[R/\gamma]))$.

Es importante recordar las características de la operación de sustitución de una variable libre por un término: se sustituirán solo las apariciones libres de de γ en Q y, además, si surge algún problema de colisión —o captura— entre variables de Q y R, se realizarán renombramientos. Pero todo eso ya está contemplado en la definición de la operación de sustitución de una variable libre por un término.

Para indicar que tras aplicar la regala de la β-reducción al término $((\lambda\gamma.Q)R)$ nos ha quedado el término $(Q[R/\gamma]))$ escribiremos lo siguiente:

$$((\lambda\gamma.Q)R) \ \rightarrow_\beta (Q[R/\gamma]))$$

4.4.5.2 Ejemplos de β-reducción

A continuación se muestran algunos ejemplos de β-reducción. Teniendo en cuenta que los términos de partida tienen la estructura $((\lambda\gamma.Q)R)$, se ha marcado, en amarillo, el componente R:

(1) $((\lambda h.h)\ g\ ) \rightarrow_\beta (h[\ g\ /h]) \equiv\ g\$

En el término h, la aparición libre de h ha sido sustituida por g.

(2) $((\lambda h.h)\ (\lambda g.((fg)f))\ ) \rightarrow_\beta (h[\ (\lambda g.((fg)f))\ /h]) \equiv\ (\lambda g.((fg)f))$

En el término h, la aparición libre de h ha sido sustituda por $(\lambda g.((fg)f))$.

(3) $((\lambda h.f)\ g\ ) \rightarrow_\beta (f[\ g\ /h]) \equiv f$

En el término f, las apariciones libres de h han sido sustituidas por g. Pero, puesto que en la expresión f no hay ninguna aparición libre de h, no ha cambiado nada, y ha quedado f.

(4) $((\lambda h.(h(hg)))\ (\lambda g.g)\ )) \rightarrow_\beta ((h(hg))[\ (\lambda g.g)\ /h]) \equiv (\ (\lambda g.g)\ (\ (\lambda g.g)\ g))$

En el término $(h(hg))$, las apariciones libres de h han sido sustituidas por $(\lambda g.g)$. En la expresión resultante, g aparece cinco veces, pero esas cinco apariciones de g no representan un mismo valor. Podemos renombrar las apariciones vinculantes y ligadas de g manteniendo la α-equivalencia: $(\ (\lambda g_1.g_1)\ (\ (\lambda g_2.g_2)\ g))$. Por tanto, se puede decir que en el término $(\ (\lambda g.g)\ (\ (\lambda g.g)\ g))$ hay tres variables g distintas: g_1, g_2 y g. Recordemos que las apariciones libres no pueden ser renombradas si queremos conservar la α-equivalencia.

(5) $((\lambda h.((\lambda g.(gh))f))\ v\ ) \rightarrow_\beta (((\lambda g.(gh))f)[\ v\ /h]) \equiv ((\lambda g.(g\ v\ ))f)$

En el término $((\lambda g.(gh))f)$, las apariciones libres de h han sido sustituidas por v.

(6) $((\lambda h.(\lambda g.(hg)))\ (gf)\ ) \rightarrow_\beta ((\lambda g.(hg))[\ (gf)\ )/h]) \equiv (\lambda j.(\ (gf)\ j))$

Al aplicar la operación de sustitución, se ha tenido que recurrir a la regla (O6) y, consecuentemente, se ha renombrado la variable g de $(\lambda g.(hg))$ con la variable j y se ha obtenido la expresión $((\lambda j.(hj))[(gf))/h])$. Una vez realizado el renombramiento de g con j, en el término (hj), las apariciones libres de h han sido sustituidas por (gf).

En este ejemplo se puede apreciar que si al realizar una β-reducción se tienen apariciones libres y no libres de una misma variable, es imprescindible que las apariciones libres no pierdan su condición de libres. Para conseguir eso, la solución es renombrar las apariciones no libres. Pero la operación de sustitución de una variable libre por un término ya resuelve esa problemática.

(7) $((\lambda h.(hh))\ (\lambda h.(hh))\ ) \rightarrow_\beta ((hh)[\ (\lambda h.(hh))\ /h]) \equiv (\ (\lambda h.(hh))\ \ (\lambda h.(hh))\ )$

En el término (hh), las apariciones libres de h han sido sustituidas por $(\lambda h.(hh))$. La expresión resultante coincide con la expresión inicial.

(8)

$$((\lambda h.((hh)g))\ (\lambda h.((hh)g))\ ) \rightarrow_\beta \quad (((hh)g)[\ (\lambda h.((hh)g))\ /h]) \equiv$$

$$\equiv ((\ (\lambda h.((hh)g))\ \ (\lambda h.((hh)g))\ )g)$$

En el término $((hh)g)$, las apariciones libres de h han sido sustituidas por $(\lambda h.((hh)g))$. La expresión resultante no coincide con la expresión inicial. En la expresión resultante se tiene una aparición de g que no estaba en la expresión inicial.

4.4.5.3 La β-reducción puede ser aplicada a subtérminos

Dado un λ-término E, la β-reducción puede ser aplicada a cualquier subtérmino de E que tenga la forma $((\lambda\gamma.Q)R)$.

En el punto (5) del apartado anterior, tenemos el término $((\lambda h.((\lambda g.(gh))f))v)$ y al aplicar la β-reducción se ha considerado que Q es $((\lambda g.(gh))f)$ y que R es v:

$$((\lambda h.\underbrace{((\lambda g.(gh))f)}_{Q})\underbrace{v}_{R})$$

El paso de β-reducción ha sido el siguiente:

$$((\lambda h.\underbrace{((\lambda g.(gh))f)}_{Q})\underbrace{v}_{R}) \rightarrow_\beta (((\lambda g.(gh))f)[v/h]) \equiv ((\lambda g.(gv))f)$$

Pero en ese punto (5), en el término $((\lambda h.((\lambda g.(gh))f))v)$, hay otra opción para aplicar β-reducción. Para ello, tendríamos que considerar que Q es (gh) y que R es f:

$$((\lambda h.((\lambda g.\underbrace{(gh)}_{Q})\underbrace{f}_{R}))v)$$

El paso de β-reducción sería el siguiente:

$$((\lambda h.((\lambda g.\underbrace{(gh)}_{Q})\underbrace{f}_{R}))v) \rightarrow_\beta ((\lambda h.(((gh)[f/g]))v) \equiv ((\lambda h.(fh))v)$$

En el término (gh), las apariciones libres de g han sido sustituidas por f.

4.4.5.4 La β-reducción en términos con dos o más parámetros

A continuación, se analizan tres ejemplos:

(1) Consideremos el término $(\lambda h.(\lambda g.((fh)(fg))))$. Ese término es una abstracción que tiene dos parámetros: h y g.

A partir de esa abstracción, podemos construir el siguiente término:

$$(((\lambda h.(\lambda g.((fh)(fg))))a)b)$$

Este nuevo término es una aplicación y la idea es que las apariciones libres de h en $(\lambda g.((fh)(fg)))$ sean sustituidas por a y que en el término resultante de esa sustitución, es decir, en $(\lambda g.((fa)(fg)))$, las apariciones libres de g en el subtérmino $((fa)(fg))$ sean sustituidas por b. Eso requiere dos aplicaciones de la β-reducción:

$$(((\lambda h.\underbrace{(\lambda g.((fh)(fg))))}_{Q}\underbrace{a}_{R})b) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por a

$$\rightarrow_\beta ((\lambda g.\underbrace{((fa)(fg))}_{Q}\underbrace{b}_{R}) \rightarrow_\beta$$

las apariciones libres de g en Q serán sustituidas por b

$$\rightarrow_\beta ((fa)(fb))$$

Aunque una abstracción tenga dos parámetros, no es obligatorio pasarle dos datos para dar lugar a una aplicación. Es posible generar una aplicación pasándole solo un dato. Ese dato corresponderá al primer parámetro:

$$((\lambda h.\underbrace{(\lambda g.((fh)(fg))))}_{Q}\underbrace{a}_{R}) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por a

$$\rightarrow_\beta (\lambda g.((fa)(fg)))$$

(2) Consideremos el término $(\lambda g.(\lambda h.(hg)))$. Ese término es una abstracción que tiene dos parámetros: h y g.

A partir de esa abstracción, podemos construir el siguiente término:

$$(((\lambda g.(\lambda h.(hg)))f)j)$$

Este nuevo término es una aplicación y la idea es que las apariciones libres de g en $(\lambda h.(hg))$ sean sustituidas por f y que en el término resultante de esa sustitución, es decir, en $(\lambda h.(hf))$, las apariciones libres de h en el subtérmino (hf) sean sustituidas por j. Eso requiere dos aplicaciones de la β-reducción:

$$(((\lambda g.\underbrace{(\lambda h.(hg))}_{Q})\underbrace{f}_{R})j) \rightarrow_\beta$$

las apariciones libres de g en Q serán sustituidas por f

$$\rightarrow_\beta ((\lambda h.\underbrace{(hf)}_{Q})\underbrace{j}_{R}) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por j

$$\rightarrow_\beta (jf)$$

Aunque una abstracción tenga dos parámetros, no es obligatorio pasarle dos datos para dar lugar a una aplicación. Es posible generar una aplicación pasándole solo un dato. Ese dato corresponderá al primer parámetro:

$$((\lambda g.\underbrace{(\lambda h.(hg))}_{Q})\underbrace{f}_{R}) \rightarrow_\beta$$

las apariciones libres de g en Q serán sustituidas por f

$$\rightarrow_\beta (\lambda h.(hf))$$

(3) Consideremos el término $(\lambda h.(\lambda h.(hg)))$. Ese término es una abstracción que tiene dos parámetros: h y h.

En el término $(\lambda h.(\lambda h.(hg)))$ los dos parámetros tienen el mismo nombre: h. De todas formas, es α-equivalente al término $(\lambda h_1.(\lambda h_2.(h_2 g)))$. Por tanto, la variable h_1 no aparece libre en el subtérmino $(\lambda h_2.(h_2 g))$.

A partir de esa abstracción, podemos construir el siguiente término:

$$(((\lambda h.(\lambda h.(hg)))f)j)$$

Este nuevo término es una aplicación y la idea es que las apariciones libres de h en $(\lambda h.(hg))$ sean sustituidas por f y que en el término resultante de esa sustitución, es decir, en $(\lambda h.(hg))$, las apariciones libres de h en el subtérmino (hg) sean sustituidas por j. Eso requiere dos aplicaciones de la β-reducción:

$$(((\lambda h.\underbrace{(\lambda h.(hg))}_{Q})\underbrace{f}_{R})j) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por f

$$\rightarrow_\beta ((\lambda h.\underbrace{(hg)}_{Q})\underbrace{j}_{R}) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por j

$$\rightarrow_\beta (jg)$$

Aunque una abstracción tenga dos parámetros, no es obligatorio pasarle dos datos para dar lugar a una aplicación. Es posible generar una aplicación pasándole solo un dato. Ese dato corresponderá al primer parámetro:

$$((\lambda h.\underbrace{(\lambda h.(hg))}_{Q})\underbrace{f}_{R}) \rightarrow_\beta$$

las apariciones libres de h en Q serán sustituidas por f

$$\rightarrow_\beta (\lambda h.(hg))$$

4.4.6 Forma β-normal

4.4.6.1 Definición de la forma β-normal

Dada una λ-expresión E, se dice que E está en forma β-normal si no es posible aplicar la regla de β-reducción sobre E.

4.4.6.2 Ejemplos sobre la forma β-normal

- La λ-expresión h está en forma β-normal.
- La λ-expresión (hg) está en forma β-normal.
- La λ-expresión $((hg)h)$ está en forma β-normal.
- La λ-expresión $((hg)(hf))$ está en forma β-normal.

- La λ-expresión $(\lambda h.(hf))$ está en forma β-normal.
- La λ-expresión $(\lambda g.(\lambda h.h))$ está en forma β-normal.
- La λ-expresión $(\lambda g.((\lambda h.h)g))$ no está en forma β-normal.

$$\underbrace{(\lambda g.\,((\lambda h.\underbrace{h}_{Q})\,\underbrace{g}_{R}))}$$

Se puede aplicar la regla de la β-reducción y se obtendrá la expresión $(\lambda g.g)$ que sí está en forma β-normal:

$$\underbrace{(\lambda g.\,((\lambda h.\underbrace{h}_{Q})\,\underbrace{g}_{R}))}\rightarrow_\beta ((\lambda g.\underbrace{(h[g/h])}))\equiv(\lambda g.g)$$

- La λ-expresión $((\lambda h.((\lambda g.(gh))f))v)$ no está en forma β-normal. Además, es posible aplicar la regla de la β-reducción en dos sitios:

(1) Primera opción:

$$\underbrace{((\lambda h.\underbrace{((\lambda g.(gh))f)}_{Q}\,\underbrace{v}_{R})}$$

La aplicación de la regla de la β-reducción generará el término $((\lambda g.(gv))f)$, el cual no está en forma β-normal.

$$\underbrace{((\lambda h.\underbrace{((\lambda g.(gh))f)}_{Q}\,\underbrace{v}_{R})}\rightarrow_\beta (((\lambda g.(gh))f)[v/h])\equiv((\lambda g.(gv))f)$$

El término $((\lambda g.(gv))f)$ no está en forma β-normal:

$$\underbrace{((\lambda g.\underbrace{(gv)}_{Q}\,\underbrace{f}_{R})}$$

Se puede aplicar la regla de la β-reducción otra vez y obtener el término (fv) que sí está en forma β-normal:

$$\underbrace{((\lambda g.\underbrace{(gv)}_{Q}\,\underbrace{f}_{R})}\rightarrow_\beta ((gv)[f/g])\equiv(fv)$$

(2) Segunda opción:

$$\underbrace{((\lambda h.\,((\lambda g.\,\underbrace{(gh)}_{Q})\,\underbrace{f}_{R}))v)}$$

En este caso, la aplicación de la regla de la β-reducción generará el término $((\lambda h.(fh))v)$, el cual tampoco está en forma β-normal.

$$\underbrace{((\lambda h.\,((\lambda g.\,\underbrace{(gh)}_{Q})\,\underbrace{f}_{R}))v)} \rightarrow_\beta ((\lambda h.(((gh)[f/g]))v) \equiv ((\lambda h.(fh))v)$$

El término $((\lambda h.(fh))v)$ no está en forma β-normal:

$$\underbrace{((\lambda h.\,\underbrace{(fh)}_{Q})\,\underbrace{v}_{R})}$$

Se puede aplicar la regla de la β-reducción de nuevo y obtener el término (fv) que sí está en forma β-normal:

$$\underbrace{((\lambda h.\,\underbrace{(fh)}_{Q})\,\underbrace{v}_{R})} \rightarrow_\beta ((fh)[v/h]) \equiv (fv)$$

4.4.7 Cálculo de la forma β-normal

4.4.7.1 Procedimiento para calcular la forma β-normal

Dado un λ-término E, la forma β-normal correspondiente a E será el término E_n que esté en forma β-normal y haya sido obtenido partiendo de E y aplicando la regla de la β-reducción mientras se pueda. Por tanto, para llegar desde E hasta E_n se generará una secuencia de la siguiente forma:

$$E_0 \rightarrow_\beta E_1 \rightarrow_\beta E_2 \rightarrow_\beta \cdots \rightarrow_\beta E_n$$

donde el término E_0 es el término E. Por otra parte, indicar que se cumplirá $n \geq 0$. Es decir, si el término E está en forma β-normal, no se podrá realizar ninguna β-reducción y, en ese caso, la secuencia constará de un único elemento, el término E, es decir, E_0:

$$E_0$$

En cambio, si el término E no está en forma β-normal, habrá que aplicar la regla de la β-reducción al menos una vez y, consecuentemente, la secuencia tendrá más de un elemento:

$$E_0 \rightarrow_\beta \cdots \rightarrow_\beta E_n$$

siendo $n \geq 1$

Por ejemplo, el λ-término $(\lambda g.(\lambda h.h))$ está en forma β-normal. Por tanto, E_0 sería $(\lambda g.(\lambda h.h))$ y la secuencia necesaria para generar la forma β-normal constaría de un único elemento:

$$\underbrace{(\lambda g.(\lambda h.h))}_{E_0}$$

Por otro lado, el λ-término $(\lambda g.((\lambda h.h)g))$ no está en forma β-normal y la secuencia necesaria para generar la forma β-normal constará de dos elementos:

$$\underbrace{(\lambda g.((\lambda h.h)g))}_{E_0} \rightarrow_\beta \underbrace{(\lambda g.g)}_{E_1}$$

El λ-término $((\lambda h.((\lambda g.(gh))f))v)$ tampoco está en forma β-normal y, además, hay dos maneras de calcular la forma β-normal. Las dos secuencias correspondientes a esas dos maneras de calcular la forma β-normal, constan de tres elementos:

(1) Primera opción para calcular la forma β-normal:

$$\underbrace{((\lambda h.((\lambda g.(gh))f))v)}_{E_0} \rightarrow_\beta \underbrace{((\lambda g.(gv))f)}_{E_1} \rightarrow_\beta \underbrace{(fv)}_{E_2}$$

(2) Segunda opción para calcular la forma β-normal:

$$\underbrace{((\lambda h.((\lambda g.(gh))f))v)}_{E_0} \rightarrow_\beta \underbrace{((\lambda h.(fh))v)}_{E_1} \rightarrow_\beta \underbrace{(fv)}_{E_2}$$

4.4.7.2 λ-términos que no tienen forma β-normal

Algunos λ-términos no tienen forma β-normal. Cuando se intenta calcular la forma β-normal de un término que no tiene forma β-normal, tras cada aplicación de la regla de la β-reducción, se obtendrá otro término que tampoco está en forma β-normal y que, por consiguiente, admite la aplicación de la regla de la β-reducción. Por tanto, en el caso de los términos que no tienen forma β-normal, el proceso de cálculo de la forma β-normal será infinito y la secuencia que se irá generando tendrá la siguiente forma:

$$E_0 \rightarrow_\beta E_1 \rightarrow_\beta E_2 \rightarrow_\beta \cdots \rightarrow_\beta E_n \rightarrow_\beta \ldots$$

A continuación se presentan dos ejemplos:

(1) El término $((\lambda h.(hh))(\lambda h.(hh)))$ no tiene forma β-normal:

$$\underbrace{((\lambda h.(hh))(\lambda h.(hh)))}_{E_0} \rightarrow_\beta \underbrace{((\lambda h.(hh))(\lambda h.(hh)))}_{E_1} \rightarrow_\beta \underbrace{((\lambda h.(hh))(\lambda h.(hh)))}_{E_2} \rightarrow_\beta \ldots$$

Todos los términos E_k (siendo $k \geq 0$) son iguales, pero tras obtener cada uno de esos términos, se puede volver a aplicar la regla de la β-reducción. Por consiguiente, nunca se llegará a un término que no acepte β-reducción.

(2) El término $((\lambda h.((hh)g))(\lambda h.((hh)g)))$ no tiene forma β-normal:

$$\underbrace{((\lambda h.((hh)g))(\lambda h.((hh)g)))}_{E_0} \rightarrow_\beta \underbrace{(((\lambda h.((hh)g))(\lambda h.((hh)g)))g)}_{E_1} \rightarrow_\beta$$

$$\rightarrow_\beta \underbrace{((((\lambda h.((hh)g))(\lambda h.((hh)g)))g)g)}_{E_2} \rightarrow_\beta \ldots$$

Todos los términos E_k (siendo $k \geq 0$) son distintos, y tras obtener cada uno de esos términos, se puede volver a aplicar la regla de la β-reducción. Consecuentemente, nunca se llegará a un término que no acepte β-reducción.

4.4.7.3 Todas las formas β-normales de un término E son α-equivalentes

Dado un término E, puede resultar que haya más de una manera de calcular la forma β-normal.

Un ejemplo es el término $((\lambda h.((\lambda g.(gh))f))v)$. Ese término no está en forma β-normal y hay dos maneras de calcular la forma β-normal:

(1) Primera opción para calcular la forma β-normal:

$$\underbrace{((\lambda h.((\lambda g.(gh))f))v)}_{E_0} \rightarrow_\beta \underbrace{((\lambda g.(gv))f)}_{E_1} \rightarrow_\beta \underbrace{(fv)}_{E_2}$$

(2) Segunda opción para calcular la forma β-normal:

$$\underbrace{((\lambda h.((\lambda g.(gh))f))v)}_{E_0} \rightarrow_\beta \underbrace{((\lambda h.(fh))v)}_{E_1} \rightarrow_\beta \underbrace{(fv)}_{E_2}$$

Pero en ambos casos se llega al mismo término: (fv).

En general, podemos decir que si para un término E existe la correspondiente forma β-normal, esa forma β-normal será única. Es decir, un término E no puede tener dos formas β-normales distintas. Pero para completar esta afirmación hay que tener en cuenta un detalle técnico. Para explicar ese detalle técnico a tener en cuenta, vamos a considerar el término $((\lambda h.(\lambda g.(hg)))(gf))$. El proceso de cálculo de la forma β-normal de ese término es el siguiente:

$$((\lambda h.(\lambda g.(hg)))\,(gf)\,) \rightarrow_\beta ((\lambda g.(hg))[\,(gf)\,)/h]) \equiv (\lambda j.(\,(gf)\,j))$$

Al aplicar la operación de sustitución de una variable libre por un término, se ha tenido que aplicar el punto (O6). Consecuentemente, se ha renombrado la variable g del término $(\lambda g.(hg))$ con la variable j. Tras realizar ese renombramiento, las apariciones libres de h en la expresión (hj) han sido reemplazadas por (gf).

La secuencia es la siguiente:

$$\underbrace{((\lambda h.(\lambda g.(hg)))(gf))}_{E_0} \rightarrow_\beta \underbrace{(\lambda j.((gf)j))}_{E_1}$$

Se ha tenido que utilizar una variable que no estaba en el término de partida y se ha elegido la variable j. Pero al elegir una variable que no esté el término de partida, hay mucahs opciones. Por ejemplo, se puede elegir v. En ese caso, la secuencia sería la siguiente:

$$\underbrace{((\lambda h.(\lambda g.(hg)))(gf))}_{E_0} \rightarrow_\beta \underbrace{(\lambda v.((gf)v))}_{E_1}$$

También podríamos haber elegido la variable z y, en ese caso, tendríamos la siguiente secuencia:

$$\underbrace{((\lambda h.(\lambda g.(hg)))(gf))}_{E_0} \rightarrow_\beta \underbrace{(\lambda z.((gf)z))}_{E_1}$$

Tal como se puede apreciar en esas secuencias, parece que la forma β-normal obtenida en cada secuencia es distinta a las formas β-normales obtenidas mediante las otras secuenciasHor ikusten den moduan, aukeratutako aldagaiaren arabera, badirudi forma β-normala desberdina izango dela:

si se ha elegido j:	si se ha elegido v:	si se ha elegido z:
$(\lambda j.((gf)j))$	$(\lambda v.((gf)v))$	$(\lambda z.((gf)z))$

Esos tres términos difieren solo en el nombre de la variable no libre: j, v o z. Pero podemos transformar cada uno de esos términos para obtener cualquiera de los otros dos términos, utilizando para ello la α-conversión:

$(\lambda j.((gf)j)) \ \rightarrow_\alpha (\lambda v.(((gf)j))\{v/j\})) \equiv (\lambda v.((gf)v))$
$(\lambda j.((gf)j)) \ \rightarrow_\alpha (\lambda z.(((gf)j))\{z/j\})) \equiv (\lambda z.((gf)z))$
$(\lambda v.((gf)v)) \ \rightarrow_\alpha (\lambda j.(((gf)v))\{j/v\})) \equiv (\lambda j.((gf)j))$
$(\lambda v.((gf)v)) \ \rightarrow_\alpha (\lambda z.(((gf)v))\{z/v\})) \equiv (\lambda z.((gf)z))$
$(\lambda z.((gf)z)) \ \rightarrow_\alpha (\lambda j.(((gf)z))\{j/z\})) \equiv (\lambda j.((gf)j))$
$(\lambda z.((gf)z)) \ \rightarrow_\alpha (\lambda v.(((gf)z))\{v/z\})) \equiv (\lambda v.((gf)v))$

Por tanto, esos tres términos son α-equivalentes.

La conclusión es la siguiente:

Todas las formas β-normales correspondientes a un término E son α-equivalentes.

Esto significa que los nombres de las variables no libres pueden ser modificados porque los nombres de las variables no libres no son relevantes.

Si la forma β-normal de un término E solo tiene variables libres, entonces será única, tal como se ha visto en el caso del término $((\lambda h.((\lambda g.(gh))f))v)$ analizado al principio de este apartado.

4.4.7.4 Estrategias para calcular la forma β-normal de un λ-término E

Hay más de una manera para calcular la forma β-normal del término $((\lambda h.((\lambda g.(gh))f))v)$. En concreto, hay dos maneras:

(1) Primera manera:

$$\underbrace{((\lambda h.\underbrace{((\lambda g.(gh))f)}_{Q}\underbrace{v}_{R})}_{} \rightarrow_\beta \underbrace{((\lambda g.\underbrace{(gv)}_{Q}\underbrace{f}_{R})}_{} \rightarrow_\beta (fv)$$

(2) Segunda manera:

$$\underbrace{((\lambda h.\,((\lambda g.\,\underbrace{(gh)}_{Q})\,\underbrace{f}_{R}))v)}_{} \rightarrow_\beta \underbrace{((\lambda h.\,\underbrace{(fh)}_{Q})\,\underbrace{v}_{R})}_{} \rightarrow_\beta (fv)$$

Cuando ante un λ-término hay más de una opciń para aplicar la β-reducción, en principio se puede optar por cualquiera de ellas. En el ejemplo con el que hemos empezado este apartado, hay dos opciones para aplicar la β-reducción al término inicial. Independientemente de la opción elegida, se ha llegado al término (fv), aunque generando secuencias distintas.

Sin embargo, no siempre todas las opciones llevan al mismo término final —que estará en forma β-normal. Puede ocurrir que unas opciones generen una secuencia que termine con un término que esté en forma β-normal y que otras opciones den lugar a un proceso infinito en el que nunca se obtendrá un tÂº'ermino en forma β-normal.

Consideremos el término $((\lambda f.j)((\lambda h.((hh)g))(\lambda h.((hh)g))))$. Hay más de una opciń para aplicar la β-reducción:

(1) Primera opción:

$$\underbrace{((\lambda f.\,\underbrace{j}_{Q})\,\underbrace{((\lambda h.((hh)g))(\lambda h.((hh)g)))}_{R})}_{} \rightarrow_\beta j$$

Se ha llegado en un único paso al término j, el cual está en forma β-normal.

(2) Segunda opción:

$$\underbrace{((\lambda f.j)\,((\lambda h.\,\underbrace{((hh)g)}_{Q})\,\underbrace{(\lambda h.((hh)g))))}_{R}}_{} \rightarrow_\beta$$

$$\underbrace{\rightarrow_\beta ((\lambda f.j)(((\lambda h.\,\underbrace{((hh)g)}_{Q})\,\underbrace{(\lambda h.((hh)g)))}_{R}\,g))}_{} \rightarrow_\beta$$

$$\underbrace{\rightarrow_\beta ((\lambda f.j)(((((\lambda h.\,\underbrace{((hh)g)}_{Q})\,\underbrace{(\lambda h.((hh)g)))}_{R}\,g)g))}_{} \rightarrow_\beta$$

$$\rightarrow_\beta \ldots$$

En este caso, se ha optado por aplicar β-reducción al subtérmino $((\lambda h.((hh)g))(\lambda h.((hh)g)))$ en todos los pasos. Esta estrategia nos ha sumergido en un proceso infinito.

(3) Tercera opción:

$$((\lambda f.j)\,\underbrace{((\lambda h.\underbrace{((hh)g)}_{Q})\,\underbrace{(\lambda h.((hh)g))}_{R})}) \rightarrow_\beta$$

$$\rightarrow_\beta ((\lambda f.j)(\underbrace{((\lambda h.\underbrace{((hh)g)}_{Q})\,\underbrace{(\lambda h.((hh)g))}_{R})\,g)}) \rightarrow_\beta$$

$$\rightarrow_\beta \underbrace{((\lambda f.\underbrace{\ j\ }_{Q})\,\underbrace{(((((\lambda h.((hh)g))(\lambda h.((hh)g)))g)g))}_{R}}) \rightarrow_\beta$$

$$\rightarrow_\beta j$$

En los dos primeros pasos, se ha optado por aplicar β-reducción sobre el subtérmino $((\lambda h.((hh)g))(\lambda h.((hh)g)))$. Pero en el tercer paso, en vez de aplicar la β-reducción de nuevo sobre el subtérmino $((\lambda h.((hh)g))(\lambda h.((hh)g)))$, se ha aplicado β-reducción sobre todo el término, y se ha llegado a la forma β-normal.

(4) Siguiendo la idea presentada en la tercera opción, se pueden generar infinitas secuencias distintas. Cada una de esas infinitas secuencias posibles será finita. Para ello hay que aplicar β-reducción sobre el subtérmino $((\lambda h.((hh)g))(\lambda h.((hh)g)))$ tantas veces como queramos y en algún momento, aplicar β-reducción sobre todo el término, lo cual dará como resultado el término j y as-í se pondrá fin a la secuencia.

Tal como se ha mostrado mediante los ejemplos previos, a la hora de dar un paso con el objetivo de calcular la forma β-normal de untérmino E, es posible que tengamos la opción de aplicar la β-reducción a distintos subtérminos. En principio, se puede optar por cualquiera de las opciones disponibles, pero solo hay una estrategia que de manera sistemática garantiza que si existe la forma β-normal correspondiente a E, se llegue a esa forma β-normal. La estrategia consiste en aplicar la β-reducción al subtérmino que esté más a la izquierda.

La conclusión es la siguiente:

> Si se ha de calcular la forma β-normal correspondiente a un término E, en cada paso habrá que optar por aplicar β-reducción sobre el subtérmino que esté más a la izquierda, teniendo en cuenta los subtérminos que admiten β-reducción.

Si la forma β-normal del término E no existe, aunque siempre se opte por el subtérmino que esté más a la izquierda, no se llegará a un término que esté en forma β-normal y el proceso de cálculo será infinito.

Puesto que los subtérminos que admiten β-reducción puede estar anidados unos dentro de otros, el término que está más a la izquierda es el término más externo.

El significado del hecho de que siempre conviene optar por el término que esté más a la izquierda es el siguiente: Dado un término de la forma $((\lambda h.Q)R)$, calcular su forma β normal es calcular su valor. Pero para calcular ese valor, no conviene calcular primero el valor R' de R –es decir, no conviene calcular primero la forma β-normal R' de R— y luego calcular la forma β-normal de $((\lambda h.Q)R')$. Al contrario, lo que conviene es centrarse en sustituir, en Q, las apariciones libres de h por R. El asunto es el siguiente: tal vez R' no existe. Ese ha sido el problema que ha aparecido en el término $((\lambda f.j)((\lambda h.((hh)g))(\lambda h.((hh)g))))$, donde R es el subtérmino $((\lambda h.((hh)g))(\lambda h.((hh)g)))$ y para el cual no hay forma β-normal.

4.4.8 η-reducción

La regla de la η-reducción sirve para simplificar términos que se ajustan a un determinado formato. En concreto, la η-reducción se aplica a los términos que tienen la forma $(\lambda\gamma.(Q\gamma))$ pero con el requerimiento adicional de que γ no aparezca libre en Q.

4.4.8.1 La regla de la η-reducción

La regla de la β-reducción especifíca qué termino se obtendrá al aplicar una abstracción de la forma $(\lambda\gamma.Q)$ a un λ-término R.

4.4.8.2 Regla de la β-reducción

En la tabla 4.4.4 de la página 64, se muestra la regla de la η-reducción. Tal como se puede apreciar en esa tabla, si tenemos un término de la forma $(\lambda\gamma.(Q\gamma))$ y, además, γ no aparezca libre en Q —es decir, $\gamma \notin \mathsf{libres}(Q)$—, entonces en vez del término $(\lambda\gamma.(Q\gamma))$ podemos poner directamente solo Q:

$$Q$$

Por consiguiente, si aplicamos η-reducción al término $(\lambda\gamma.(Q\gamma))$, nos quedará el término Q.

Pero hay que tener en cuenta que es imprescindible que γ no aparezca libre en Q.

Para indicar que tras aplicar la regala de la η-reducción al término $(\lambda\gamma.(Q\gamma))$ nos ha quedado el término Q escribiremos lo siguiente:

$$(\lambda\gamma.(Q\gamma)) \ \rightarrow_\eta Q$$

4.4.8.3 Significado de la η-reducción

Tal como se ha indicado en el apartado anterior, el esquema general de la η-breducción es el siguiente:

$$(\lambda\gamma.(Q\gamma)) \rightarrow_\eta Q$$

En ese esquema, γ es una variable y Q es un λ-término que no contiene ninguna aparición libre de γ.

La regla de la η-reducción nos dice lo siguiente:

> Si en el término Q no hay ninguna aparición libre de la variable γ, entonces la función $(\lambda\gamma.(Q\gamma))$ y la función Q tendrán el mismo comportamiento y darán lugar al mismo resultado cuando se les pase un dato de entrada.

Al fin y al cabo, si a la función $(\lambda\gamma.(Q\gamma))$ se le da como dato de entrada el término R, calcular el resultado consiste en calcular la forma β-normal de $((\lambda\gamma.(Q\gamma))R)$:

$$\underbrace{((\lambda\gamma.(Q\gamma))R)}_{E_0} \rightarrow_\beta \underbrace{(QR)}_{E_1} \rightarrow_\beta \ldots$$

La regla de la η-reducción viene a decir que el primer paso de esa secuencia es evitable partiendo directamente del término (QR):

$$\underbrace{(QR)}_{E_1} \rightarrow_\beta \ldots$$

También se puede poner de la siguiente forma:

$$\underbrace{(\underbrace{(\lambda\gamma.(Q\gamma))}\, R)}_{E_0} \rightarrow_\eta \underbrace{(QR)}_{E_1} \rightarrow_\beta \ldots$$

4.4.8.4 Ejemplos de η-reducción

A continuación se muestran algunos ejemplos de η-reducción:

(1) En el término $(\lambda h.((hf)g))$ no es posible aplicar η-reducción porque el término $((\lambda h.((hf)g))$ no se ajusta al formato $(\lambda\gamma.(Q\gamma))$.

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(hf)}_{\mathrm{Q}} \underbrace{g}_{\delta}))$$

Donde debiera aparecer la variable γ —es decir, la variable h— por segunda vez, aparece la variable g, que ha sido indicada como δ, para remarcar que es distinta de γ.

La η-reducción no puede ser aplicada porque realmente la función $(\lambda h.((hf)g))$ y la función (hf) son distintas. Veamos qué ocurre si les damos el dato j a esas dos funciones:

- $((\lambda h.((hf)g))j) \rightarrow_\beta ((jf)g)$
- $((hf)j)$ Está en forma β-normal desde el principio.

Por tanto, al dar el dato j a la función $(\lambda h.((hf)g))$ y a la función (hf), el resultado es distinto.

(2) En el término $(\lambda h.((hf)h))$ tampoco es posible aplicar η-reducción. Aunque el término $((\lambda h.((hf)h))$ tenga la forma $(\lambda\gamma.(Q\gamma))$, la variable$h$ —es decir, la variable γ— aparece libre en el término (hf) —es decir, en el componente Q.

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(hf)}_{\text{Q}} \underbrace{h}_{\gamma}))$$

La imposibilidad de aplicar la η-reducción significa que la función $(\lambda h.((hf)h))$ y la función (hf) son distintas. Veamos qué ocurre si les damos el dato j a esas dos funciones:

- $((\lambda h.((hf)h))j) \rightarrow_\beta ((jf)j)$
- $((hf)j)$ Está en forma β-normal desde el principio.

Al dar el dato j a la función $(\lambda h.((hf)h))$ y a la función (hf), el resultado es distinto.

(3) En el término $(\lambda h.((gf)h))$ sí se puede aplicar η-reducción. Por una parte, el término $((\lambda h.((gf)h))$ tiene la forma $(\lambda\gamma.(Q\gamma))$ y, además, la variable h —es decir, la variable γ— no aparece libre en el término (gf) —es decir, en el componente Q.

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(gf)}_{\text{Q}} \underbrace{h}_{\gamma}))$$

El hecho de que se pueda aplicar la η-reducción significa que la función $(\lambda h.((gf)h))$ y la función (gf) son iguales. A continuación se muestra el resultado que devuelve cada una de ellas para el dato j:

- $((\lambda h.((gf)h))j) \rightarrow_\beta ((gf)j)$
- $((gf)j)$ Está en forma β-normal desde el principio.

La función $(\lambda h.((gf)h))$ y la función (gf) devuelven el mismo resultado para el dato j.

(4) En el término $(\lambda h.((\lambda g.(hg))h))$ no es posible aplicar η-reducción. El término $(\lambda h.((\lambda g.(hg))h))$ tiene la forma $(\lambda\gamma.(Q\gamma))$, pero la variable$h$ —es decir, la variable γ— aparece libre en el término $(\lambda g.(hg))$ —es decir, en el componente Q. .

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda g.(hg))}_{\text{Q}} \underbrace{h}_{\gamma}))$$

La imposibilidad de aplicar la η-reducción significa que la función $(\lambda h.((\lambda g.(hg))h))$ y la función $(\lambda g.(hg))$ son distintas. Veamos qué ocurre si les damos el dato j a esas dos funciones:

- $((\lambda h.((\lambda g.(hg))h))j) \rightarrow_\beta ((\lambda g.(jg))j) \rightarrow_\beta (jj)$
- $((\lambda g.(hg))j) \rightarrow_\beta (hj)$

Se han obtenido resultados distintos al dar el dato j a la función $(\lambda h.((\lambda g.(hg))h))$ y a la función $(\lambda g.(hg))$.

(5) En el término $(\lambda h.((\lambda g.(fg))h))$ sí se puede aplicar η-reducción. Por una parte, el término $(\lambda h.((\lambda g.(fg))h))$ tiene la forma $(\lambda\gamma.(Q\gamma))$ y, además, la variable h —es decir, la variable γ— no aparece libre en el término $(\lambda g.(fg))$ —es decir, en el componente Q.

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda g.(fg))}_{\text{Q}} \underbrace{h}_{\gamma}))$$

El hecho de que se pueda aplicar la η-reducción significa que la función $(\lambda h.((\lambda g.(fg))h))$ y la función $(\lambda g.(fg))$ son iguales. A continuación se muestra el resultado que devuelve cada una de ellas para el dato j:

- $((\lambda h.((\lambda g.(fg))h))j) \rightarrow_\beta ((\lambda g.(fg))j) \rightarrow_\beta (fj)$
- $((\lambda g.(fg))j) \rightarrow_\beta (fj)$

La función $(\lambda h.((\lambda g.(fg))h))$ y la función $(\lambda g.(fg))$ devuelven el mismo resultado para el dato j.

(6) En el término $(\lambda h.((\lambda h.(hg))h))$ también se puede aplicar η-reducción. Por una parte, el término $(\lambda h.((\lambda h.(hg))h))$ tiene la forma $(\lambda\gamma.(Q\gamma))$ y, además, la variable h —es decir, la variable γ— no aparece libre en el término $(\lambda h.(hg))$ —es decir, en el componente Q.

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda h.(hg))}_{\text{Q}} \underbrace{h}_{\gamma}))$$

El hecho de que se pueda aplicar la η-reducción significa que la función $(\lambda h.((\lambda h.(hg))h))$ y la función $(\lambda h.(hg))$ son iguales. A continuación se muestra el resultado que devuelve cada una de ellas para el dato j:

- $((\lambda h.((\lambda h.(hg))h))j) \rightarrow_\beta ((\lambda h.(hg))j) \rightarrow_\beta (jg)$
- $((\lambda h.(hg))j) \rightarrow_\beta (jg)$

La función $(\lambda h.((\lambda h.(hg))h))$ y la función $(\lambda h.(hg))$ devuelven el mismo resultado para el dato j.

4.4.8.5 Se puede aplicar η-reducción a subtérminos

Dado un λ-término E, se puede aplicar η-reducción a cualquier subtérmino de E que tenga la forma $((\lambda\gamma.Q)\gamma)$ y, además, cumpla el requerimiento de que la variable γ no aparezca libre en Q.

A continuación se analizan dos ejemplos:

- **Primer ejemplo:** En el término $(\lambda h.((\lambda v.((fg)v))g))$, se puede aplicar η-reducción en el subtérmino $(\lambda v.((fg)v))$:

$$(\lambda h.((\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q}\underbrace{v}_{\gamma}))g))$$

  El paso de η-reducción sería el siguiente:

$$(\lambda h.((\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q}\underbrace{v}_{\gamma}))g)) \rightarrow_\eta (\lambda h.(\underbrace{(fg)}_{Q}g))$$

- **Segundo ejemplo:** En el término $(\lambda h.((\lambda v.((fg)v))h))$, se puede aplicar η-reducción en dos subtérminos:

  - Primera opción:

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda v.((fg)v))}_{Q}\underbrace{h}_{\gamma}))$$

  - Segunda opción:

$$(\lambda h.((\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q}\underbrace{v}_{\gamma}))h))$$

  Los pasos de η-reducción en esos dos casos serían los siguientes:

– En la primera opción:

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda v.((fg)v))}_{Q} \underbrace{h}_{\gamma})) \rightarrow_\eta \underbrace{(\lambda v.((fg)v))}_{Q}$$

– En la segunda opción:

$$(\lambda h.((\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q} \underbrace{v}_{\gamma}))h)) \rightarrow_\eta (\lambda h.(\underbrace{(fg)}_{Q} h))$$

Además, en cada caso, tras aplicar una vez la η-reducción, se puede volver a aplicar η-reducción por segunda vez:

– En la primera opción:

$$(\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q} \underbrace{v}_{\gamma})$$

– En la segunda opción:

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(fg)}_{Q} \underbrace{h}_{\gamma})$$

En definitiva, se pueden generar las siguientes dos secuencias:

– En la primera opción:

$$(\lambda \underbrace{h}_{\gamma}.(\underbrace{(\lambda v.((fg)v))}_{\mathbf{Q}} \underbrace{h}_{\gamma})) \rightarrow_\eta \underbrace{(\lambda v.((fg)v))}_{Q} \equiv (\lambda \underbrace{v}_{\gamma'}.(\underbrace{(fg)}_{Q'} \underbrace{v}_{\gamma'}) \rightarrow_\eta \underbrace{(fg)}_{Q'}$$

– En la segunda opción:

$$(\lambda h.((\lambda \underbrace{v}_{\gamma}.(\underbrace{(fg)}_{Q} \underbrace{v}_{\gamma}))h)) \rightarrow_\eta (\lambda h.(\underbrace{(fg)}_{Q} h)) \equiv (\lambda \underbrace{h}_{\gamma'}.(\underbrace{(fg)}_{Q'} \underbrace{h}_{\gamma'}) \rightarrow_\eta \underbrace{(fg)}_{Q'}$$

La primera η-reducción ha producido el término $(\lambda v.((fg)v))$ en la primera opción y el término $(\lambda h.((fg)h))$ en la segunda opción. Esoa dos términos son α-equivalentes: solo difieren en el nombre de las variables no libres; en un término v es la que aparece como variable no libre y en el otro término h es la que aparece como variable no libre. Tras la segunda η-reducción, en ambos casos se ha obtenido el término (fg). En general, más habitual que obtener el mismo término, es obtener términos α-equivalentes.

4.4.9 Forma $\beta\eta$-normal

4.4.9.1 Definición de la forma $\beta\eta$-normal

Dada una λ-expresión E, se dice que E está en forma $\beta\eta$-normal si no es posible aplicar ni la regla de β-reducción ni la regla de η-reducción sobre E.

4.4.10 Cálculo de la forma $\beta\eta$-normal

4.4.10.1 Procedimiento para calcular la forma $\beta\eta$-normal

Dado un λ-término E, la forma $\beta\eta$-normal correspondiente a E será el término E_n que esté en forma $\beta\eta$-normal y haya sido obtenido partiendo de E y aplicando la regla de la β-reducción y la regla de la η-reducción mientras se pueda. Por tanto, para llegar desde E hasta E_n se generará una secuencia de la siguiente forma:

$$E_0 \rightarrow_{\Omega_1} E_1 \rightarrow_{\Omega_2} E_2 \rightarrow_{\Omega_3} \cdots \rightarrow_{\Omega_n} E_n$$

donde cada Ω_k es un elemento del conjunto $\{\beta, \eta\}$ y el término E_0 es el término E. Por otra parte, indicar que se cumplirá $n \geq 0$. Es decir, si el término E está en forma $\beta\eta$-normal, no se podrá realizar ninguna β-reducción ni η-reducción y, en ese caso, la secuencia constará de un único elemento, el término E, es decir, E_0:

$$E_0$$

En cambio, si el término E no está en forma $\beta\eta$-normal, habrá que aplicar la regla de la β-reducción o la regla de la η-reducción al menos una vez y, consecuentemente, la secuencia tendrá más de un elemento:

$$E_0 \rightarrow_{\Omega_1} \cdots \rightarrow_{\Omega_n} E_n$$

siendo $n \geq 1$ y siendo $\Omega_k \in \{\beta, \eta\}$, para cada k que cumpla $1 \leq k \leq n$.

4.4.10.2 λ-términos que no tienen forma $\beta\eta$-normal

En el apartado 4.4.7.2 de la página 50, se ha mostrado que algunos λ-términos no tienen forma β-normali. Aquellos términos que no tengan forma β-normal, tampoco tendrán forma $\beta\eta$-normal. El problema esa asociado a la β-reducción. Puesto que en esos casos que no tienen forma $\beta\eta$-normal, tras cada β-reducción se tendrá un término que admite otra β-reducción. Consecuentemente, en el caso de los términos que no tienen forma $\beta\eta$-normal, el proceso de cálculo de la forma $\beta\eta$-normal será infinito y la secuencia de términos que se irá generando tendrá el siguiente formato:

$$E_0 \rightarrow_{\Omega_1} E_1 \rightarrow_{\Omega_2} E_2 \rightarrow_{\Omega_3} \cdots \rightarrow_{\Omega_n} E_n \rightarrow_{\Omega_{n+1}} \ldots$$

siendo $\Omega_k \in \{\beta, \eta\}$ para cada k que cumpla $1 \leq k$.

4.4.11 Noción de computación en el λ-cálculo

En el λ-cálculo, la computación asociada a un término E es el proceso de cálculo de la forma β-normal correspondiente a E. Consecuentemente, para algunos términos la computación será finita mientras que para otros términos la computación será infinita. Si la computación es finita, dicha computación terminará cuando se haya obtenido un término que esté en forma β-normal y ese último término será el resultado de la computación. En cambio, si la computación es infinita, nunca se llegará a un término que esté en forma β-normal y la computación no generará un resultado; se desarrollará un proceso infinito con la intención de producir un resultado —un término en forma β-normal— pero nunca se llegará a ese resultado buscado.

Puesto que la η-reducción nos ofrece la posibilidad de sinplificar algunos términos, podemos expandir la noción de computabilidad teniendo en cuenta también la regla de la η-reducción. En tal caso, en las computaciones, además de la regla de la β-reducción, podemos utilizar también la regla de la η-reducción. El objetivo será generar un término que esté en forma $\beta\eta$-normal.

La regla de la β-reducción
$((\lambda\gamma.Q)R) \ \rightarrow_\beta \ (Q[R/\gamma])$

Tabla 4.4.3. La regla de la β-reducción.

La regla de la η-reducción
$(\lambda\gamma.(Q\gamma)) \ \rightarrow_\eta \ Q$ si la variable γ no aparece libre en Q, es decir, $\gamma \notin \mathsf{libres}(Q)$.

Tabla 4.4.4. La regla de la η-reducción.

4.5.

Valores booleanos y operaciones sobre booleanos en el λ-cálculo

4.5.1 Valores booleanos y elección entre dos opciones

En la lógica matemática, los valores booleanos son $True$ y $False$ y el valor de las expresiones lógicas es $True$ o $False$. Dicho de otra forma, cada expresión lógica nos conduce al valor $True$ o al valor $False$.

Las expresiones lógicas se suelen utilizar para decidir qué opción elegir entre dos opciones posibles: elegir la primera opción o elegir la segunda opción. La estructura básica que formaliza esa idea es la estructura *if-then-else*:

$$(\textit{if } \rho \textit{ then } \gamma_1 \textit{ else } \gamma_2)$$

donde ρ es una expresión booleana y γ_1 y γ_2 son las dos opciones posibles, de entre las cuales se ha de elegir una. Si el valor de ρ es $True$, entonces se elegirá γ_1, mientras que si el valor de ρ es $False$, entonces se elegirá γ_2.

4.5.2 Elementos y normas para escribir fórmulas de la lógica proposicional

Para indicar qué fórmulas son aceptables en la lógica proposicional, hay establecer el conjunto de **símbolos básicos** y el conjunto de reglas a tener en cuenta para generar fórmulas:

- Símbolos básicos:

  (a) Proposizioak izendatzeko erabiliko diren sinboloez osatutako Conjunto Prop formado por los símbolos que se utilizarán para dar nombre a las proposiciones. Para representar proposiciones, se suelen utilizar letras minúsculas: $p, q, r, s, \ldots$ (si hace

falta, con subíndices, $p_1, p_2, p_3, \ldots, q_1, q_2, \ldots$). Es posible utilizar nombres más largos ($es\_amarillo$, $llueve$, $funciona$, $eleccion\_realizada$, ...). También es posible simplificar el asunto de los nombres de las proposiciones definiendo el conjunto Prop de la siguiente forma: $\mathsf{Prop} = \{p_j \mid j \geq 0\}$. Si se decide hacerlo de esa manera, el conjunto Prop será el conjunto infinito $\{p_0, p_1, p_2, p_3, \ldots\}$. Por tanto, q, r, $llueve$, $funciona$ y otros nombres de ese tipo se suelen utilizar para facilitar la legibilidad y por comodidad, pero también es posible limitarse a utilizar nombres que tengan el formato p_j.

(b) **Símbolos lógicos** u operadores lógicos:

- Los operadores T (cierto) y F (falso), que son operadores constantes, es decir, operadores sin parámetros.
- El operador $\neg$ (negación, no), que es un operador que solo tiene un parámetro.
- Los operadores $\wedge$ (conjunción, y), $\vee$ (disyunción, o), $\rightarrow$ (condición o implicación) y $\leftrightarrow$ (equivalencia lógica o doble implicación). Estos operadores tienen dos parámetros.

(c) **Símbolos auxiliares**: apertura de paréntesis '(' y cierre de paréntesis ')'.

Al fijar los símbolos básicos que se pueden utilizar, se establece el alfabeto del cual se pueden coger elementos para formar fórmulas. Ese alfabeto depende del conjunto Prop que hayamos fijado. Por tanto, representaremos dicho alfabeto como $\mathbb{A}_{\mathsf{Prop}}$.

- Reglas para generar fórmulas correctas:

De todas aquellas cadenas de símbolos que podemos crear utilizando el alfabeto $\mathbb{A}_{\mathsf{Prop}}$, solo son **fórmulas de la lógica proposicional** o **fórmulas proposicionales** las cadenas finitas que se pueden generar teniendo en cuenta las siguientes reglas de formación:

(At) **Fórmulas atómicas**
T y F son fórmulas y cada elemento π del conjunto Prop es también una fórmula.

($\neg$) **Negación**
Si δ es una fórmula, entonces $(\neg\delta)$ es también una fórmula.

($\wedge$) **Conjunción**
Si δ_1 y δ_2 son fórmulas, entonces $(\delta_1 \wedge \delta_2)$ es una fórmula.

($\vee$) **Disyunción**
Si δ_1 y δ_2 son fórmulas, entonces $(\delta_1 \vee \delta_2)$ es una fórmula.

($\rightarrow$) **Implicación**
Si δ_1 y δ_2 son fórmulas, entonces $(\delta_1 \rightarrow \delta_2)$ es una fórmula.

($\leftrightarrow$) **Equivalencia lógica**
Si δ_1 y δ_2 son fórmulas, entonces $(\delta_1 \leftrightarrow \delta_2)$ es una fórmula.

En el caso (At), se ha utilizado el símbolo π para representar los elementos del conjunto Prop, es decir, p, q, r, ..., p_0, p_1, ...

4.5.3 Significado o valor de las fórmulas de la lógica proposicional

El significado de las fórmulas proposicionales presentadas en el apartado anterior es el siguiente:

- Operadores sin parámetros, T eta F:

T	F
$True$	$False$

  Los operadores T y F son operadores constantes. El valor o significado del operador T es siempre $True$ y no depende de ningún parámetro. De la misma forma, el valor o significado del operador F es siempre $False$ y no depende de ningún parámetro.

- Los elementos π del conjunto Prop:

  Para saber si el valor de un elemento π del conjunto Prop es $True$ o $False$, se tendrá una función $\mathcal{V}$ que dará la valoración de cada elemento π del conjunto Prop:

$$\mathcal{V} : \mathsf{Prop} \rightarrow \{True, False\}$$

  Por tanto, la función $\mathcal{V}$ nos indicará, para cada elemento π de Prop, si ese elemento es $True$ o $False$.

- El operador $\neg$, que solo tiene un parámetro:

δ	$\neg\delta$
$False$	$True$
$True$	$False$

  El operador $\neg$ no es constante, no devuelve siempre $True$ o siempre $False$. Depende de un parámetro δ cuyo valor es booleano. Es decir, el valor de la fórmula $\neg\delta$, depende del valor del parámetro δ.

- Los operadores $\wedge$, $\vee$, $\rightarrow$ y $\leftrightarrow$:

δ_1	δ_2	$\delta_1 \wedge \delta_2$	$\delta_1 \vee \delta_2$	$\delta_1 \rightarrow \delta_2$	$\delta_1 \leftrightarrow \delta_2$
$False$	$False$	$False$	$False$	$True$	$True$
$False$	$True$	$False$	$True$	$True$	$False$
$True$	$False$	$False$	$True$	$False$	$False$
$True$	$True$	$True$	$True$	$True$	$True$

Los operadores $\wedge$, $\vee$, $\rightarrow$ y $\leftrightarrow$ no son constantes, no devuelven siempre $True$ o siempre $False$. El valor de las expresiones booleanas $\delta_1 \wedge \delta_2$, $\delta_1 \vee \delta_2$, $\delta_1 \rightarrow \delta_2$ y $\delta_1 \leftrightarrow \delta_2$ depende de dos parámetros δ_1 y δ_2 que también tienen valor booleano.

Tal como se puede apreciar en esas tablas, el número de combinaciones correspondientes a cada operador depende del número de parámetros: en el caso de T y F, el número de combinaciones es $2^0 = 1$; en el caso de $\neg$, el número de combinaciones es $2^1 = 2$ y, en el caso de $\wedge$, $\vee$, $\rightarrow$ y $\leftrightarrow$ el número de combinaciones es $2^2 = 4$.

4.5.4 Equivalencias en lógica proposicional

4.5.4.1 Definición de la noción de equivalencia

Dadas dos fórmulas de la lógica proposicional, φ y ψ, se dice que φ y ψ son equivalentes si las dos fórmulas tienen el mismo valor para cada valoración posible $\mathcal{V}$. Para indicar que φ y ψ son equivalentes, escribiremos $\varphi \equiv \psi$.

4.5.4.2 Leyes algebraicas de Bool

Las leyes algebraicas de Bool establecen equivalencias entre distintas fórmulas lógicas. En la tabla 4.5.1 de la página 69 se muestran dichas leyes.

4.5.4.3 Equivalencias relacionadas con la implicación

En la tabla 4.5.2 de la página 4.5.2 se muestran algunas equivalencias relacionadas con la implicación y la doble implicación, es decir, el operador $\rightarrow$ y el operador $\leftrightarrow$.

4.5.4.4 Operadores lógicos que se requieren para expresar todas las fórmulas lógicas

Tal como se ha mostrado mediante las equivalencias presentadas en el apartado anterior, algunos operadores lógicos pueden ser expresados haciendo uso de otros operadores lógicos.

Por ejemplo, puesto que se cumple $(\varphi \rightarrow \psi) \equiv ((\neg\varphi) \vee \psi)$, el operador $\rightarrow$ puede ser expresado mediante los operadores del conjunto $\{\neg, \vee\}$. Por tanto, si decidimos utilizar los operadores $\neg$ y$\vee$, no es necesario utilizar el operador $\rightarrow$. De todas formas, es habitual utilizar el operador $\rightarrow$, porque la expresión $(\varphi \rightarrow \psi)$ se entiende mejor que la expresión $((\neg\varphi) \vee \psi)$.

Si n es el número de parámetros, se podrán definir $2^{(2^n)}$ operadores lógicos que tienen n parámetros. Es decir, con $n = 0$, se pueden definir $2^{(2^0)} = 2$ operadores de cero parámetros, con $n = 1$, se pueden definir $2^{(2^1)} = 4$ operadores de un parámetro, con $n = 2$, se pueden

Asociatividad	$\varphi \vee (\psi \vee \chi) \equiv (\varphi \vee \psi) \vee \chi$ $\varphi \wedge (\psi \wedge \chi) \equiv (\varphi \wedge \psi) \wedge \chi$
Conmutatividad	$\varphi \vee \psi \equiv \psi \vee \varphi$ $\varphi \wedge \psi \equiv \psi \wedge \varphi$
Distrubutividad	$\varphi \vee (\psi \wedge \chi) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi)$ $\varphi \wedge (\psi \vee \chi) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$
De Morgan	$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$ $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$
Idempotencia	$\varphi \vee \varphi \equiv \varphi$ $\varphi \wedge \varphi \equiv \varphi$
Absorción	$\varphi \vee (\varphi \wedge \psi) \equiv \varphi$ $\varphi \wedge (\varphi \vee \psi) \equiv \varphi$
Elemento neutro	$\varphi \vee \mathsf{F} \equiv \varphi$ $\varphi \wedge \mathsf{T} \equiv \varphi$
Elemento nulo	$\varphi \vee \mathsf{T} \equiv \mathsf{T}$ $\varphi \wedge \mathsf{F} \equiv \mathsf{F}$
Leyes de la negación	$\neg\mathsf{T} \equiv \mathsf{F}$ $\neg\mathsf{F} \equiv \mathsf{T}$ Negación doble: $\neg\neg\varphi \equiv \varphi$ Solo dos opciones: $\varphi \vee \neg\varphi \equiv \mathsf{T}$ Contradicción: $\varphi \wedge \neg\varphi \equiv \mathsf{F}$

Tabla 4.5.1. Leyes algebraicas de Boole.

(imp_1)	$\varphi \to \psi$	$\equiv$	$\neg\varphi \vee \psi$	(inp_7)	$\varphi \wedge \psi$	$\equiv$	$\neg(\varphi \to \neg\psi)$
(imp_2)	$\varphi \to \psi$	$\equiv$	$\neg(\varphi \wedge \neg\psi)$	(imp_8)	$\varphi \wedge \psi$	$\equiv$	$\neg(\psi \to \neg\varphi)$
(imp_3)	$\varphi \to \psi$	$\equiv$	$\neg\psi \to \neg\varphi$	(inp_9)	$\varphi \to \mathsf{F}$	$\equiv$	$\neg\varphi$
(imp_4)	$\varphi \leftrightarrow \psi$	$\equiv$	$(\varphi \to \psi) \wedge (\psi \to \varphi)$	(imp_{10})	$\varphi \to \mathsf{T}$	$\equiv$	T
(imp_5)	$\varphi \vee \psi$	$\equiv$	$\neg\varphi \to \psi$	(imp_{11})	$\mathsf{F} \to \varphi$	$\equiv$	T
(imp_6)	$\varphi \vee \psi$	$\equiv$	$\neg\psi \to \varphi$	(imp_{12})	$\mathsf{T} \to \varphi$	$\equiv$	φ

Tabla 4.5.2. Algunas equivalencias relacionadas con los operadores $\to$ y $\leftrightarrow$.

definir $2^{(2^2)} = 16$ operadores de dos parámetros, etc.

Se dice que un conjunto $\mathcal{C}$ formado por operadores lógicos es funcionalmente completo si cialquier otro operador lógico puede ser expresado mediante los operadores de $\mathcal{C}$. Por ejemplo, los conjuntos $\{\neg, \wedge, \vee\}$, $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\neg, \rightarrow\}$ y $\{\mathsf{F}, \rightarrow\}$ son funcionalmente completos.

A modo de caso concreto, si consideramos el conjunto $\{\mathsf{F}, \rightarrow\}$, los operadores $\neg$ y $\wedge$ se pueden definir de la siguiente manera:

$$(\neg\varphi) \equiv (\varphi \rightarrow \mathsf{F})$$

$$(\varphi \wedge \psi) \equiv ((\varphi \rightarrow (\psi \rightarrow \mathsf{F})) \rightarrow \mathsf{F})$$

Por otro lado, el conjunto $\{\neg, \leftrightarrow\}$ no es funcionalmente completo. En concreto, $\rightarrow$ no puede ser expresado utilizando $\neg$ y $\leftrightarrow$.

4.5.5 Los operadores T y F en el λ-cálculo: TRUE y FALSE

En el λ-cálculo, todos los elementos son funciones y, consecuentemente, también los operadores T y F se representan como funciones.

Retomemos la estructura (*if* ρ *then* γ_1 *else* γ_2). Supongamos que el elemento ρ es T. En ese caso, se seleccionará la primera opción, la opción γ_1. En cambio, si el elemento ρ es F, se seleccionará la segunda opción, la opción γ_2. La definición de T y de F en el λ-cálculo se basa en esa idea: con T se selecciona la primera opción y con F se selecciona la segunda opción.

4.5.5.1 El operador T en el λ-cálculo: TRUE

El esquema correspondiente a la definición de T en el λ-cálculo es el siguiente:

$$(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$$

donde, γ_1 y γ_2 son dos varibles distintas.

El término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ es la definición del operador lógico T en el λ-cálculo. Tal como se puede apreciar en esa definición ese término tiene dos parámetros —γ_1 y γ_2— y devolverá como resultado el primero, es decir, γ_1. Por tanto, si a la función definida mediante el término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ le pasamos dos datos —dos λ-términos—, la función devolverá el primero. La función tiene dos opciones —γ_1 y γ_2— y siempre devolverá el primero, es decir, γ_1.

El comportamiento del término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ se ajusta al significado de la estructura (*if* ρ *then* γ_1 *else* γ_2) para el caso en el que el valor de ρ es $True$.

Para referirnos a la función o término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ sin necesidad de escribir todo el término, vamos a darle el nombre TRUE:

$$\text{TRUE} \equiv (\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$$

Utilizaremos el nombre TRUE como abreviatura y en vez de escribir $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ escribiremos TRUE. Pero TRUE no es parte del lenguaje del λ-cálculo, es solo una abreviatura que utilizaremos nosotros para facilitar la legibilidad.

En los ejemplos concretos, los elementos del esquema general $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$ serán reemplazados con las variables que nos interesen. A modo de muestra, podemos poner h en vez de γ_1 y g en vez de γ_2:

$$\text{TRUE} \equiv (\lambda h.(\lambda g.h))$$

En esa función, la idea es que hay dos opciones: h y g. La función siempre elegirá el elemento h, es decir, el primero por la izquierda.

En vez de utilizar las variables h y g, es posible utilizar otras dos variables cualesquiera. Por ejemplo, x e y:

$$\text{TRUE} \equiv (\lambda x.(\lambda y.x))$$

En esa función, la idea es que hay dos opciones: x e y. La función siempre elegirá el elemento x, es decir, el primero por la izquierda.

El término $(\lambda h.(\lambda g.h))$ y el término $(\lambda x.(\lambda y.x))$ son α-equivalentes. Debido a ello, da igual qué variables utilicemos: h y g, o x e y, o cualquier otro par de variables.

A continuación se muestra, mediante ejemplos concretos, cómo se comporta la función $(\lambda h.(\lambda g.h))$:

(1)

$$(((\lambda h.(\lambda g.h))j)f) \;\rightarrow_\beta\; ((\lambda g.j)f) \;\rightarrow_\beta\; j$$

A la función TRUE se le han pasado los datos j y f. La función ha devuelto como resultado j, es decir, el primero.

Si queremos utilizar la abreviatura, escribiremos $((\text{TRUE}\ j)f)$. El proceso de cálculo del resultado lo representaremos de la siguiente forma:

$$((\text{TRUE}\ j)f) \;\equiv\; (((\lambda h.(\lambda g.h))j)f) \;\rightarrow_\beta\; ((\lambda g.j)f) \;\rightarrow_\beta\; j$$

La abreviatura ha de ser reemplazada por su definición.

(2)

$$(((\lambda h.(\lambda g.h))f)j) \ \rightarrow_\beta \ ((\lambda g.f)j) \ \rightarrow_\beta \ f$$

En este caso, a la función TRUE se le han pasado los datos f y j, en ese orden. La función ha devuelto como resultado f, es decir, el primero.

Si queremos utilizar la abreviatura, escribiremos $((\textsc{true}\ f)j)$.

(3)

$$(((\lambda h.(\lambda g.h))j)j) \ \rightarrow_\beta \ ((\lambda g.j)j) \ \rightarrow_\beta \ j$$

En este caso, a la función TRUE se le han pasado los datos j y j. La función ha devuelto como resultado j, es decir, el primero.

Si queremos utilizar la abreviatura, escribiremos $((\textsc{true}\ j)j)$.

(4)

$$(((\lambda h.(\lambda g.h))h)g) \ \rightarrow_\beta \ ((\lambda g.h)g) \ \rightarrow_\beta \ h$$

En este caso, a la función TRUE se le han pasado los datos h y g. La función ha devuelto como resultado h, es decir, el primero.

La versión abreviada sería $((\textsc{true}\ h)g)$.

(5)

$$(((\lambda h.(\lambda g.h))(\lambda v.(vs)))f) \ \rightarrow_\beta \ ((\lambda g.(\lambda v.(vs)))f) \ \rightarrow_\beta \ (\lambda v.(vs))$$

En este caso, a la función TRUE se le han pasado los datos $(\lambda v.(vs))$ y f. La función ha devuelto como resultado $(\lambda v.(vs))$, es decir, el primero.

La versión abreviada sería $((\textsc{true}\ (\lambda v.(vs)))f)$.

(6)

$$((\lambda h.(\lambda g.h))(\lambda v.(vs))) \ \rightarrow_\beta \ (\lambda g.(\lambda v.(vs)))$$

En este caso, a la función TRUE se le ha pasado solo un dato. Consecuentemente, solo se ha podido dar un paso de β-reducción. Debido a que no le hemos dado un segundo dato, ha quedado la función $(\lambda g.(\lambda v.(vs)))$.

La versión abreviada sería $(\textsc{true}\ (\lambda v.(vs)))$.

Si le damos un dato a la función $(\lambda g.(\lambda v.(vs)))$ que se ha obtenido con un paso de β-reducción, entonces el resultado será $(\lambda v.(vs))$:

$$((\lambda g.(\lambda v.(vs)))r) \rightarrow_\beta (\lambda v.(vs))$$

A la función $(\lambda g.(\lambda v.(vs)))$ le hemos dado el dato r y la función ha devuelto el resultado $(\lambda v.(vs))$.

4.5.5.2 El operador F en el λ-cálculo: FALSE

El esquema correspondiente a la definición de F en el λ-cálculo es el siguiente:

$$(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$$

donde, γ_1 y γ_2 son dos varibles distintas.

El término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ es la definición del operador lógico F en el λ-cálculo. Tal como se puede apreciar en esa definición ese término tiene dos parámetros —γ_1 y γ_2— y devolverá como resultado el segundo, es decir, γ_2. Por tanto, si a la función definida mediante el término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ le pasamos dos datos —dos λ-términos—, la función devolverá el segundo. La función tiene dos opciones —γ_1 y γ_2— y siempre devolverá el segundo, es decir, γ_2.

El comportamiento del término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ se ajusta al significado de la estructura (*if* ρ *then* γ_1 *else* γ_2) para el caso en el que el valor de ρ es $False$.

Para referirnos a la función o término $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ sin necesidad de escribir todo el término, vamos a darle el nombre FALSE:

$$\text{FALSE} \equiv (\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$$

Utilizaremos el nombre FALSE como abreviatura y en vez de escribir $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ escribiremos FALSE. Pero FALSE no es parte del lenguaje del λ-cálculo, es solo una abreviatura que utilizaremos nosotros para facilitar la legibilidad.

En los ejemplos concretos, los elementos del esquema general $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h en vez de γ_1 y g en vez de γ_2:

$$\text{FALSE} \equiv (\lambda h.(\lambda g.g))$$

En esa función, la idea es que hay dos opciones: h y g. La función siempre elegirá el elemento g, es decir, el segundo por la izquierda.

En vez de utilizar las variables h y g, es posible utilizar otras dos variables cualesquiera. Por ejemplo, x e y:

$$\textsc{false} \equiv (\lambda x.(\lambda y.y))$$

En esa función, la idea es que hay dos opciones: x e y. La función siempre elegirá el elemento y, es decir, el segundo por la izquierda.

El término $(\lambda h.(\lambda g.g))$ y el término $(\lambda x.(\lambda y.y))$ son α-equivalentes. Debido a ello, da igual qué variables utilicemos: h y g, o x e y, o cualquier otro par de variables.

A continuación se muestra, mediante ejemplos concretos, cómo se comporta la función $(\lambda h.(\lambda g.g))$:

(1)

$$(((\lambda h.(\lambda g.g))j)f) \ \rightarrow_\beta \ ((\lambda g.g)f) \ \rightarrow_\beta \ j$$

A la función $\textsc{false}$ se le han pasado los datos j y f. La función ha devuelto como resultado f, es decir, el segundo.

Si queremos utilizar la abreviatura, escribiremos $((\textsc{false}\ j)f)$. El proceso de cálculo del resultado lo representaremos de la siguiente forma:

$$((\textsc{false}\ j)f) \ \equiv \ (((\lambda h.(\lambda g.g))j)f) \ \rightarrow_\beta \ ((\lambda g.g)f) \ \rightarrow_\beta \ f$$

La abreviatura ha de ser reemplazada por su definición.

(2)

$$(((\lambda h.(\lambda g.g))f)j) \ \rightarrow_\beta \ ((\lambda g.g)j) \ \rightarrow_\beta \ j$$

En este caso, a la función $\textsc{false}$ se le han pasado los datos f y j, en ese orden. La función ha devuelto como resultado f, es decir, el segundo.

Si queremos utilizar la abreviatura, escribiremos $((\textsc{false}\ f)j)$.

(3)

$$(((\lambda h.(\lambda g.g))j)j) \ \rightarrow_\beta \ ((\lambda g.g)j) \ \rightarrow_\beta \ j$$

En este caso, a la función $\textsc{false}$ se le han pasado los datos j y j. La función ha devuelto como resultado j, es decir, el segundo.

Si queremos utilizar la abreviatura, escribiremos $((\textsc{false}\ j)j)$.

(4)
$$(((\lambda h.(\lambda g.g))h)g) \ \rightarrow_\beta ((\lambda g.g)g) \ \rightarrow_\beta g$$

En este caso, a la función FALSE se le han pasado los datos h y g. La función ha devuelto como resultado g, es decir, el segundo.

La versión abreviada sería $((\text{FALSE}\ h)g)$.

(5)
$$(((\lambda h.(\lambda g.g))(\lambda v.(vs)))f) \ \rightarrow_\beta ((\lambda g.g)f) \ \rightarrow_\beta f$$

En este caso, a la función FALSE se le han pasado los datos $(\lambda v.(vs))$ y f. La función ha devuelto como resultado f, es decir, el segundo.

La versión abreviada sería $((\text{FALSE}\ (\lambda v.(vs)))f)$.

(6)
$$((\lambda h.(\lambda g.g))(\lambda v.(vs))) \ \rightarrow_\beta (\lambda g.g)$$

En este caso, a la función FALSE se le ha pasado solo un dato. Consecuentemente, solo se ha podido dar un paso de β-reducción. Debido a que no le hemos dado un segundo dato, ha quedado la función $(\lambda g.g)$.

La versión abreviada sería $(\text{FALSE}\ (\lambda v.(vs)))$.

Si le damos un dato a la función $(\lambda g.g)$ que se ha obtenido con un paso de β-reducción, entonces el resultado será ese dato:

$$((\lambda g.g)r) \ \rightarrow_\beta r$$

A la función $(\lambda g.g)$ le hemos dado el dato r y la función ha devuelto el resultado r.

4.5.5.3 TRUE2 y FALSE2: relación entre las funciones TRUE y FALSE

Es posible dar la definición del término TRUE utilizando el término FALSE:

$$\text{TRUE2} \equiv (\lambda h.(\lambda g.(\underbrace{((\lambda x.(\lambda y.y))}_{\text{FALSE}}\ g)h)))$$

Dadas dos opciones h y g, la función que implementa el operador T en el λ-cálculo, ha de elegir h. La función FALSE, dadas dos opciones, elegirá el segundo. Por tanto, para conseguir que se devuelva h, habrá que dar los datos g y h —en ese orden— a la función FALSE.

Para no confundir esta nueva definición con la definición TRUE que hemos dado en un apartado previo, a esta nueva definición la denominaremos TRUE2.

En la figura 4.5.1 de la página 77 se muestra un ejemplo en el se utiliza la función TRUE2. A la función TRUE2 se le han dado los datos j y f. Tal como se esperaba, la función TRUE2 ha devuelto el primero como respuesta, es decir, ha devuelto j.

Por otro lado, también es posible dar la definición del término FALSE utilizando el término TRUE:

$$\text{FALSE2} \equiv (\lambda h.(\lambda g.((\underbrace{(\lambda x.(\lambda y.x))}_{\text{TRUE}} g)h)))$$

Dadas dos opciones h y g, la función que implementa el operador F en el λ-cálculo, ha de elegir g. La función TRUE, dadas dos opciones, elegirá el primero. Por tanto, para conseguir que se devuelva g, habrá que dar los datos g y h —en ese orden— a la función TRUE.

Para no confundir esta nueva definición con la definición FALSE que hemos dado en un apartado previo, a esta nueva definición la denominaremos FALSE2.

En la figura 4.5.2 de la página 78 se muestra un ejemplo en el que se utiliza la función FALSE2. A la función FALSE2 se le han dado los datos j y f. Tal como se esperaba, la función FALSE2 ha devuelto el segundo como respuesta, es decir, ha devuelto f.

4.5.6 La estructura if-$then$-$else$ en el λ-cálculo: ITE

4.5.6.1 Definición de ITE

Tal como se ha explicado en el apartado 4.5.1 de la página 4.5.1, las expresiones booleanas suelen ser utilizadas para realizar una elección entre dos opciones posibles: primera opción y segunda opción. La estructura básica que materializa esa idea es la estructura *if-then-else*:

$$(\textit{if } \rho \textit{ then } \gamma_1 \textit{ else } \gamma_2)$$

donde ρ es una expresión booleana. Si el valor de ρ es $True$, entonces se elige la opción γ_1. En cambio, si el valor de ρ es $False$, entonces se elige la opción γ_2.

La definiciones de las funciones TRUE y FALSE están basadas en esa idea:

- La función TRUE, es decir, la función $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_1))$, es una implementación de la estructura $(\textit{if } \rho \textit{ then } \gamma_1 \textit{ else } \gamma_2)$ adaptada al caso en el que el valor de ρ es $True$.

$$\underbrace{(((\lambda h.(\lambda g.(((\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\, g)h)))\, j)f)}_{\textbf{TRUE2}}\ \rightarrow_\beta$$

h será sustituida por j

$$\rightarrow_\beta\ \underbrace{((\lambda g.(((\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\, g)j))\, f)}_{\textbf{FALSE}}\ \rightarrow_\beta$$

g será sustituida por f

$$\rightarrow_\beta\ (((\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\, f)j)\ \rightarrow_\beta$$

x será sustituida por f

$$\rightarrow_\beta\ (\underbrace{(\lambda y.y)}\, j)\ \rightarrow_\beta$$

y será sustituida por j

$$\rightarrow_\beta\ j$$

Figura 4.5.1. Ejemplo de uso de la función **TRUE2**.

$$\underbrace{(((\lambda h.(\lambda g.(((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\, g)h)))\, j)f)}_{\textbf{FALSE2}}\ \rightarrow_\beta$$

h será sustituida por j

$$\rightarrow_\beta\ \underbrace{((\lambda g.(((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\, g)j))\, f)}\ \rightarrow_\beta$$

g será sustituida por f

$$\rightarrow_\beta\ (((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\, f)j)\ \rightarrow_\beta$$

x será sustituida por f

$$\rightarrow_\beta\ (\underbrace{(\lambda y.f)}\, j)\ \rightarrow_\beta$$

y será sustituida por j

$$\rightarrow_\beta\ f$$

Figura 4.5.2. Ejemplo de uso de la función FALSE2.

- La función FALSE, es decir, la función $(\lambda\gamma_1.(\lambda\gamma_2.\gamma_2))$, es una implementación de la estructura (*if* ρ *then* γ_1 *else* γ_2) adaptada al caso en el que el valor de ρ es $False$.

La definición de la estructura (*if* ρ *then* γ_1 *else* γ_2) en el λ-cálculo se obtiene generalizando las definiciones de TRUE y FALSE, incorporando para ello sel elemento ρ:

$$(\lambda\ \rho.(\lambda\ \gamma_1.(\lambda\ \gamma_2.((\rho\ \gamma_1)\gamma_2))))$$

Dado un valor rho cuyo valor será TRUE o FALSE, y dadas dos opciones γ_1 y γ_2, la elección entre γ_1 y γ_2 se dirime mediante el término $((\rho\ \gamma_1)\gamma_2)$. Si el valor de ρ es TRUE, entonces tendremos la expresión $((\text{TRUE}\ \gamma_1)\gamma_2)$ y la función TRUE elegirá γ_1. En cambio, si el valor de ρ es FALSE, entonces tendremos la expresión $((\text{FALSE}\ \gamma_1)\gamma_2)$ y la función FALSE elegirá γ_2.

Identificaremos el término $(\lambda\ \rho.(\lambda\ \gamma_1.(\lambda\ \gamma_2.((\rho\ \gamma_1)\gamma_2))))$ con la abreviatura ITE:

$$\text{ITE} \equiv (\lambda\ \rho.(\lambda\ \gamma_1.(\lambda\ \gamma_2.((\rho\ \gamma_1)\gamma_2))))$$

En los ejemplos concretos, los elementos del esquema general $(\lambda\ \rho.(\lambda\ \gamma_1.(\lambda\ \gamma_2.((\rho\ \gamma_1)\gamma_2))))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v en vez de ρ, h en vez de γ_1 y g en vez de γ_2:

$$\text{ITE} \equiv (\lambda\ v.(\lambda\ h.(\lambda\ g.((v\ h)g))))$$

En esa función, la idea es que hay que elegir entre h y g y la elección que se haga dependerá del valor de v: si el valor de v es TRUE, entonces se elegirá h y si el valor de v es FALSE, entonces se elegirá g.

En vez de utilizar las variables v, h y g, es posible utilizar otras tres variables cualesquiera. Por ejemplo, z, x e y:

$$\text{ITE} \equiv (\lambda\ z.(\lambda\ x.(\lambda\ y.((z\ x)y))))$$

En esa función, la idea es que hay que elegir entre x e y y la elección que se haga dependerá del valor de z: si el valor de z es TRUE, entonces se elegirá x y si el valor de z es FALSE, entonces se elegirá y.

El término $(\lambda\ v.(\lambda\ h.(\lambda\ g.((v\ h)g))))$ y el término $(\lambda\ z.(\lambda\ x.(\lambda\ y.((z\ x)y))))$ son α-equivalentes. Debido a ello, da igual qué variables utilicemos: v, h y g, o z, x e y, u otras tres variables cualesquiera.

4.5.6.2 ITE: ejemplos

En la figura 4.5.3 de la página 81 se muestra un ejemplo de utilización de la función ITE. En ese ejemplo, la función ITE recibe, como datos de entrada, los valores TRUE, j y f. Tal como se espera, la función ha devuelto el valor j, es decir, el primero de entre j y f.

En la figura 4.5.4 de la página 82 se muestra otro ejemplo de utilización de la función ITE. En ese ejemplo, la función ITE recibe, como datos de entrada, los valores FALSE, j y f. Tal como se espera, la función ha devuelto el valor f, es decir, el segundo de entre j y f.

4.5.7 Expresiones lógicas en el λ-cálculo

4.5.7.1 TRUE y FALSE son expresiones booleanas

Las funciones TRUE y FALSE son expresiones booleanas. Son las implementaciones, en el λ-cálculo, de los operadores lógicos T eta F.

4.5.7.2 Las variables serán consideradas expresiones booleanas cuando convenga: la noción de contexto (la función Contexto)

En la lógica proposicional, los elementos atómicos son los operadores lógicos T y F y los símbolos proposicionales: $p, q, r, \ldots, p_0, p_1, p_2, \ldots$.

El valor de cada símbolo proposicional será $True$ o $False$. El valor concreto de cada símbolo vendrá determinado por una función $\mathcal{V}$:

$$\mathcal{V} : \mathsf{Prop} \rightarrow \{True, False\}$$

Por tanto, la función $\mathcal{V}$ —o valoración— indica si el valor es $true$ o $False$ para cadaelemento π del conjunto Prop.

En la lógica proposicional, todos los elementos tienen un valor booleano y, consecuentemente, no hay posibilidad de confusión a la hora de establecer el tipo de dato de cada elemento.

En el λ-cálculo, no se indica el tipo de datos al que pertenece cada elemento y, por consiguiente, no se sabe a qué tipo de datos pertenece el valor de cada variable y cada término.

En el λ-cálculo, si un termino contiene apariciones libres de variables, el valor de esas variables libres será determinado por una función conocida como "contexto". Esa función se denomina Contexto y su formato es el siguiente:

$$\mathsf{Contexto} : \mathsf{VAR} \rightarrow L_{\mathsf{VAR}}$$

$$((((\underbrace{(\lambda\ v.(\lambda\ h.(\lambda\ g.((v\ h)g))))}_{\textbf{ITE}}\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}})j)f)\ \rightarrow_\beta$$

v será sustituida por **TRUE**

$$\rightarrow_\beta \underbrace{(((\lambda\ h.(\lambda\ g.(((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\ h)g))\ j)f)\ \rightarrow_\beta}$$

h será sustituida por j

$$\rightarrow_\beta \underbrace{((\lambda\ g.(((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\ j)g))\ f)\ \rightarrow_\beta}$$

g será sustituida por f

$$\rightarrow_\beta (((\underbrace{(\lambda x.(\lambda y.x))}_{\textbf{TRUE}}\ j)f)\ \rightarrow_\beta$$

x será sustituida por j

$$\rightarrow_\beta (\underbrace{(\lambda y.j)}\ f)\ \rightarrow_\beta$$

y será sustituida por f

$$\rightarrow_\beta j$$

Está en forma β-normal.

Figura 4.5.3. Ejemplo de uso de la función **ITE** para el caso en el que el primer parámetro es **TRUE**.

$$((((\underbrace{(\lambda\ v.(\lambda\ h.(\lambda\ g.((v\ h)g))))}_{\textbf{ITE}}\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}})j)f)\ \rightarrow_\beta$$

v será sustituida por **FALSE**

$$\rightarrow_\beta (\underbrace{((\lambda\ h.(\lambda\ g.((\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\ h)g))\ j)f)}_{\textbf{FALSE}}\ \rightarrow_\beta$$

h será sustituida por j

$$\rightarrow_\beta \underbrace{((\lambda\ g.((\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\ j)g))\ f)}_{\textbf{FALSE}}\ \rightarrow_\beta$$

g será sustituida por f

$$\rightarrow_\beta (\underbrace{((\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\ j)f)\ \rightarrow_\beta$$

x será sustituida por j

$$\rightarrow_\beta (\underbrace{(\lambda y.y)}\ f)\ \rightarrow_\beta$$

y será sustituida por f

$$\rightarrow_\beta f$$

Está en forma β-normal.

Figura 4.5.4. Ejemplo de uso de la función **ITE** para el caso en el que el primer parámetro es **FALSE**.

donde VAR es el conjunto formado por todas las variables, L_{VAR} es el conjunto formado por todos los λ-términos que se pueden generar utilizando variables del conjunto VAR. La función Contexto asigna un valor —es decir, un λ-término— a las variables libres. Consecuentemente, cuando estemos realizando cálculos con expresiones booleanas, supondremos que la función Contexto asociará el valor TRUE o el valor FALSE a las variables libres involucradas en dichos cálculos.

4.5.7.3 El caso de ITE

En la figura 4.5.5 de la página 84, se muestra qué ocurre si a la función ITE en vez de darle tres datos le damos solo un dato. En ese ejemplo, a la función ITE se le ha dado como dato la variable s y se ha obtenido como resultado una función que se parece a las funciones TRUE y FALSE:

$$(\lambda\ h.(\lambda\ g.((s\ h)g)))$$

Recordemos que el término TRUE es $(\lambda\ h.(\lambda\ g.h))$ y que el término FALSE es $(\lambda\ h.(\lambda\ g.g))$.

Con el término $(\lambda\ h.(\lambda\ g.((s\ h)g)))$, dependiendo del valor de s, se elegirá h o g. Es decir, si el valor de la variable libre s es TRUE, entonces se elegirá h y, en cambio, si el valor de la variable libre s es FALSE, entonces se elegirá g. Por otro lado, el valor de la variable libre s depende del contexto, es decir, la función Contexto determinará en cada momento el valor de s. Consecuentemente, si el valor de Contexto(s) es TRUE, entonces se elegirá h y si el valor de Contexto(s) es FALSE, entonces se elegirá g.

En la figura 4.5.6 de la página 84 se muestra el resultado que se obtiene partiendo del término $(\lambda\ h.(\lambda\ g.((s\ h)g)))$ y considerando que el valor de Contexto(s) es TRUE. Teniendo en cuenta lo que se muestra en la figura 4.5.5 de la página 84 y lo que se muestra en la figura 4.5.6 de la página 84, se puede observar que si el valor de Contexto(s) es TRUE, entonces la forma β-normal del término $(\text{ITE}\ s)$ es TRUE.

En la figura 4.5.7 de la página 96 se muestra el resultado que se obtiene partiendo del término $(\lambda\ h.(\lambda\ g.((s\ h)g)))$ y considerando que el valor de Contexto(s) es FALSE. Teniendo en cuenta lo que se muestra en la figura 4.5.5 de la página 84 y lo que se muestra en la figura 4.5.7 de la página 96, se puede observar que si el valor de Contexto(s) es FALSE, entonces la forma β-normal del término $(\text{ITE}\ s)$ es FALSE.

4.5.7.4 Más mecanismos para formular expresiones booleanas: los λ-términos correspondientes a los operadores lógicos

En los siguientes apartados se indicará cómo se suelen definir en el λ-cálculo los operadores lógicos $\neg$, $\wedge$, $\vee$, $\rightarrow$ y $\leftrightarrow$. En concreto, se analizarán los casos $\neg\delta$, $\delta_1 \wedge \delta_2$, $\delta_1 \vee \delta_2$, $\delta_1 \rightarrow \delta_2$ y

$$\underbrace{((\lambda\, v.(\lambda\, h.(\lambda\, g.((v\ h)g))))}_{\mathbf{ITE}}\, s)\ \rightarrow_\beta$$

v será sustituida por s

$\rightarrow_\beta\ (\lambda\, h.(\lambda\, g.((s\ h)g)))$

Está en forma β-normal.

Figura 4.5.5. Ejemplo de uso de la función **ITE** para el caso en el que solo se le da un dato s en vez de tres datos.

$$(\lambda\, h.(\lambda\, g.((\underbrace{\ s\ }\ h)g))) \equiv (\lambda\, h.(\lambda\, g.((\underbrace{\quad \mathbf{TRUE} \quad}_{\mathsf{Contexto}(s)}\ h)g)))\ \equiv$$

$$\equiv (\lambda\, h.(\lambda\, g.(\underbrace{(\underbrace{(\lambda x.(\lambda y.x))}_{\mathbf{TRUE}}\ h)\, g)}))\ \rightarrow_\beta$$

x será sustituida por h

$\rightarrow_\beta\ (\lambda\, h.(\lambda\, g.\ \underbrace{((\lambda y.h)g)}))\ \rightarrow_\beta$

y será sustituida por g

$\rightarrow_\beta\ \underbrace{(\lambda\, h.(\lambda\, g.h))}_{\mathbf{TRUE}}$

Está en forma β-normal.

Figura 4.5.6. Desarrollo del término $(\mathbf{ITE}\ s)$ cuando el valor de $\mathsf{Contexto}(s)$ es **TRUE**.

$\delta_1 \leftrightarrow \delta_2$. En todas esas definiciones, el objetivo es que el resultado obtenido sea TRUE o FALSE.

4.5.8 El operador ¬ en el λ-cálculo: NOT

4.5.8.1 NOT: definición

En el caso de la negación, es decir, en el caso del operador ¬, ese operador recibe como dato una expresión δ cuyo valor es booleano y devuelve el valor booleano opuesto al valor de δ. Es decir, $(\neg\delta)$ devuelve el valor booleano contrario al valor booleano de δ. En el λ-cálculo, el operador ¬ ha de ser definido o expresado como una función. La función o término se denominará NOT. La función NOT tendrá como dato de entrada un valor booleano δ. Si el valor del parámetro δ es TRUE, entonces la función NOT ha de devolver el valor FALSE. En cambio, si el valor del parámetro δ es FALSE, entonces la función NOT ha de devolver el valor TRUE.

El esquema correspondiente a la definición de NOT en el λ-cálculo es el siguiente:

$$\text{NOT} \equiv (\lambda\delta.((\delta\ \text{FALSE})\text{TRUE}))$$

donde δ representa una variable.

El término $(\lambda\delta.((\delta\ \text{FALSE})\text{TRUE}))$ es la definición del operador lógico ¬ en el λ-cálculo. Tal como se puede apreciar en esa definición ese término tiene un parámetro: δ. Para utilizar la función NOT, se le ha de pasar una expresión booleana. Tras aplicar β-reducción todas las veces que haga falta, esa expresión lógica quedará reducida a TRUE o FALSE. Por tanto, podemos entender que δ representa el valor TRUE o el valor FALSE. El significado del término $((\delta\ \text{FALSE})\text{TRUE})$ es el siguiente: Si el valor de δ es TRUE, la función δ elegirá el primero de sus dos argumentos, es decir, FALSE; en cambio, si el valor de δ es FALSE, la función δ elegirá el segundo de sus dos argumentos, es decir, TRUE. La función NOT ha de hacer la elección contraria a la que haría la función δ. Debido a ello, para definir NOT, se llama a la función δ pero pasándole TRUE y FALSE en el orden contrario: FALSE y TRUE

$$((\delta\ \text{FALSE})\text{TRUE})$$

La función δ ha de elegir entre FALSE y TRUE.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.((\delta\ \text{FALSE})\text{TRUE}))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v en vez de δ:

$$\text{NOT} \equiv (\lambda v.((v\ \text{FALSE})\text{TRUE}))$$

Por otro lado, TRUE y FALSE son abreviaturas y, consecuentemente, en vez de FALSE podemos poner $(\lambda h.(\lambda g.g))$ y en vez de TRUE podemos poner $(\lambda x.(\lambda y.x))$:

$$\text{NOT} \equiv (\lambda v.((v\ \underbrace{(\lambda h.(\lambda g.g))}_{\text{FALSE}})\ \underbrace{(\lambda x.(\lambda y.x))}_{\text{TRUE}}))$$

En esa función, el valor de v será TRUE o FALSE y, consecuentemente, v representa una función que sirve para elegir entre dos opciones. Pero las dos dos opciones de entre las cuales hay que elegir una son, a su vez, TRUE y FALSE: hay que elegir una opción entre TRUE y FALSE y esa elección la tiene que hacer v. Además, es importante recalcar que a la función v se le darán los datos TRUE y FALSE en orden inverso: FALSE y TRUE. De esa forma la función NOT terminará eligiendo lo contrario de lo que eligiría v si a v se le dieran los datos TRUE y FALSE en este orden.

En vez de utilizar las variables v, h, g, x e y, es posible utilizar otras cinco variables cualesquiera. Por ejemplo, w, a, b, c y n:

$$\text{NOT} \equiv (\lambda w.((w \underbrace{(\lambda a.(\lambda b.b))}_{\text{FALSE}} \underbrace{(\lambda c.(\lambda n.c))}_{\text{TRUE}}))$$

El término $(\lambda v.((v\ (\lambda h.(\lambda g.g)))(\lambda x.(\lambda y.x))))$ y el término $(\lambda w.((w\ (\lambda a.(\lambda b.b)))(\lambda c.(\lambda n.c))))$ son α-equivalentes. Debido a ello, da igual cuál de ellos utilicemos.

4.5.8.2 Ejemplos de uso de NOT

En este apartado se analizan dos ejemplos que sirven para ilustrar el uso de la función $(\lambda v.((v\ \text{FALSE})\text{TRUE}))$.

A modo de primer ejemplo, consideramos el término $((\lambda v.((v\ \text{FALSE})\text{TRUE}))\text{TRUE})$:

$$(\underbrace{(\lambda v.((v\ \text{FALSE})\text{TRUE}))}_{\text{NOT}}\ \text{TRUE})$$

Por tanto, estamos ante el término (NOT TRUE) y la respuesta esperada es FALSE. En la figura 4.5.8 de la página 97 se muestra el proceso de cálculo de la forma β-normal correspondiente al término (NOT TRUE).

A modo de segundo ejemplo, a la función NOT se le dará como dato de entrada el valor FALSE. Esto significa que hay que calcular la forma β-normal del siguiente término:

$$(\underbrace{(\lambda v.((v\ \text{FALSE})\text{TRUE}))}_{\text{NOT}}\ \text{FALSE})$$

Por tanto, tenemos el término (NOT FALSE) y el resultado esperado es TRUE. En la figura 4.5.9 de la página 98 se muestra el proceso de cálculo de la forma β-normal correspondiente al término (NOT FALSE).

4.5.8.3 Cuando NOT recibe un dato que no es ni TRUE ni FALSE

Primero, recordemos el esquema correspondiente a la definición de la función NOT en el λ-cálculo:

$$\textbf{NOT} \equiv (\lambda\delta.((\delta\ \textbf{FALSE})\textbf{TRUE}))$$

donde δ representa una variable. Habitualmente, en vez de δ se pondrá una variable concreta, por ejemplo, v:

$$\textbf{NOT} \equiv (\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))$$

En el apartado anterior se ha mostrado qué ocurre si a la función $(\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))$ se le pasa el dato TRUE y qué ocurre si se le pasa el dato FALSE. Pero en el λ-cálculo no hay ninguna herramienta para controlar o detectar la idoneicidad de los datos de entrada. Debido a ello, a la función $(\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))$ en vez de pasarle el dato TRUE o el dato FALSE, se le puede pasar cualquier otra función. Desde el punto de vista del λ-cálculo, eso no generará ningún problema y se aplicará la β-reducción sin ningún problema. Pero el resultado que se obtenga puede ser incoherente con respecto a la idea con la que se ha definido la función NOT.

Consideremos el término $((\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))(\lambda j.j))$:

$$(\underbrace{(\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))}_{\textbf{NOT}}\underbrace{(\lambda j.j)}_{\delta})$$

Lo que tenemos, en formato abreviado, es el término $(\textbf{NOT}\ (\lambda j.j))$, donde el dato de entrada $(\lambda j.j)$ no es ni TRUE ni FALSE. En la figura 4.5.10 de la página 99 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(\textbf{NOT}\ (\lambda j.j))$. Se ha obtenido como resultado el término $(\lambda g.\textbf{TRUE})$ que no es ni TRUE ni FALSE. Cuando utilizamos la función NOT lo normal suele ser obtener como resultado TRUE o FALSE.

Consideremos ahora el término $((\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))(\lambda j.r))$:

$$(\underbrace{(\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))}_{\textbf{NOT}}\underbrace{(\lambda j.r)}_{\delta})$$

Estamos ante el término $(\textbf{NOT}\ (\lambda j.r))$, donde el dato de entrada $(\lambda j.r)$ no es ni TRUE ni FALSE. En la figura 4.5.11 de la página 100 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(\textbf{NOT}\ (\lambda j.r))$. Se ha obtenido como resultado el término $(r\ \textbf{TRUE})$ que no es ni TRUE ni FALSE. Como ya se ha dicho en el ejemplo anterior, cuando utilizamos la función NOT lo que esperamos es obtener como resultado TRUE o FALSE.

También puede resultar que el proceso de cálculo de la forma β-normal correspondiente al dato que se le haya dado a la función NOT, sea infinito. Eso es lo que ocurre con el término $((\lambda v.((v\ \textbf{FALSE})\textbf{TRUE}))((\lambda x.xx)(\lambda x.xx)))$:

$$(\underbrace{(\lambda v.((v\ \textsc{false})\textsc{true}))}_{\textsc{not}}\underbrace{((\lambda x.((xx)y))(\lambda x.((xx)y)))}_{\delta})$$

El dato de entrada $((\lambda x.((xx)y))(\lambda x.((xx)y)))$ no es ni TRUE ni FALSE. En la figura 4.5.12 de la página 101 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(\textsc{not}\ ((\lambda x.((xx)y))(\lambda x.((xx)y))))$. Lo que ocurre es que el término

$$(\textsc{not}\ ((\lambda x.((xx)y))(\lambda x.((xx)y))))$$

no tiene forma β-normal y, consecuentemente, el proceso de cálculo de la forma β-normal es infinito.

4.5.9 El operador $\wedge$ en el λ-cálculo: AND

En este apartado se implementará el operador $\wedge$ mediante una función del λ-cálculo. Denominaremos AND a esa función.

4.5.9.1 AND: definición

En el caso de la conjunción, es decir, en el caso del operador $\wedge$, ese operador recibe como datos dos expresiones δ_1 y δ_2 de valor booleano. Si el valor de esas dos expresiones es $True$, entonces el valor de la expresión $(\delta_1 \wedge \delta_2)$ será $True$ y, en caso contrario, el valor de la expresión $(\delta_1 \wedge \delta_2)$ será $False$. En el λ-cálculo, el operador $\wedge$ ha de ser definido o expresado mediante una abstracción. La función o término se denominará AND. La función AND tendrá como datos de entrada dos valores booleanos δ_1 y δ_2. Si el valor del parámetro δ_1 es TRUE y el valor del parámetro δ_2 es también TRUE, entonces la función AND ha de devolver el valor TRUE. En cambio, si el valor del parámetro δ_1 es FALSE o el valor del parámetro δ_2 es FALSE, entonces la función AND ha de devolver el valor FALSE.

El esquema correspondiente a la definición de AND en el λ-cálculo es el siguiente:

$$\textsc{and} \equiv (\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textsc{false})))$$

donde δ_1 y δ_2 representan dos variables.

El término $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textsc{false})))$ es la definición del operador lógico $\wedge$ en el λ-cálculo. Tal como se puede apreciar en esa definición, ese término tiene dos parámetros: δ_1 y δ_2. Para utilizar la función AND, se le han de pasar dos expresiones booleanas. Esas expresiones booleanas sustituirán a los parámetros δ_1 y δ_2. Tras aplicar β-reducción todas las veces que haga falta, esas expresiones lógicas quedarán reducidas a TRUE o FALSE. Por tanto, podemos entender que los parámetros δ_1 y δ_2 representan, cada uno de ellos, el valor TRUE o el valor FALSE. El significado del término $((\delta_1\ \delta_2)\textsc{false})$ es el siguiente: la función δ_1 ha de elegir entre δ_2 y FALSE;

si el valor de δ_1 es TRUE, la función δ_1 elegirá el primero de sus dos argumentos, es decir, δ_2 y δ_2 será el resultado final —recordemos que δ_2 es TRUE o FALSE; en cambio, si el valor de δ_1 es FALSE, la función δ_1 elegirá el segundo de sus dos argumentos, es decir, FALSE:

$$((\delta_1\ \delta_2)\textbf{FALSE})$$

La función δ_1 elegirá entre δ_2 y FALSE.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textbf{FALSE})))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v y w en vez de δ_1 y δ_2:

$$\textbf{AND} \equiv (\lambda v.(\lambda w.((v\ w)\textbf{FALSE})))$$

Puesto que FALSE es una abreviatura, en vez de FALSE podemos poner $(\lambda h.(\lambda g.g))$:

$$\textbf{AND} \equiv (\lambda v.(\lambda w.((v\ w)\underbrace{(\lambda h.(\lambda g.g))}_{\textbf{FALSE}})))$$

En esa función, el valor de v será TRUE o FALSE. Consecuentemente, v elegirá entre dos valores, en concreto, elegirá entre w y FALSE. Si el valor de v es TRUE, entonces elegirá w. Si el valor de v es FALSE, entonces elegirá FALSE. Si la función v ha elegido el primer argumento, es decir, w, el resultado final es w. Recordemos que el valor de w será TRUE o FALSE. Si la función v ha elegido el segundo argumento, es decir, FALSE, entonces el resultado será FALSE. En resumen, si el valor de v es TRUE y el valor de w es TRUE, entonces el resultado final final será TRUE; en cambio, si el valor de v es FALSE o el valor de w es FALSE, entonces el resultado final será FALSE.

En vez de las variables v, w, h y g, se pueden utilizar cuatro variables cualesquiera, por ejemplo, a, b, c y d:

$$\textbf{AND} \equiv (\lambda a.(\lambda b.((a\ b)\underbrace{(\lambda c.(\lambda d.d))}_{\textbf{FALSE}})))$$

El término $(\lambda v.(\lambda w.((v\ \ w)(\lambda h.(\lambda g.g)))))$ y el término $(\lambda a.(\lambda b.((a\ \ b)(\lambda c.(\lambda d.d)))))$ son α-equivalentes. Debido a ello, da igual qué variables se utilicen.

4.5.9.2 Ejemplos de uso de AND

En este apartado se analizan dos ejemplos que sirven para ilustrar el uso de la función $(\lambda v.(\lambda w.((v\ w)\textbf{FALSE})))$.

A modo de primer ejemplo, consideramos el término $(((\lambda v.(\lambda w.((v\ w)\textbf{FALSE})))\textbf{TRUE})\textbf{TRUE})$:

$$((\underbrace{(\lambda v.(\lambda w.((v\ w)\textbf{FALSE})))}_{\textbf{AND}}\ \textbf{TRUE})\textbf{TRUE})$$

Por tanto, estamos ante el término $((\textsc{and}\ \textsc{true})\textsc{true})$ y el resultado debería ser TRUE. En la figura 4.5.13 de la página 102 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\textsc{and}\ \textsc{true})\textsc{true})$.

A modo de segundo ejemplo, consideramos el término $(((\lambda v.(\lambda w.((v\ w)\textsc{false})))\textsc{false})\textsc{true})$:

$$(((\underbrace{(\lambda v.(\lambda w.((v\ w)\textsc{false})))}_{\textsc{and}}\ \textsc{false})\textsc{true})$$

Por tanto, estamos ante el término $((\textsc{and}\ \textsc{false})\textsc{true})$ y el resultado debería ser FALSE. En la figura 4.5.14 de la página 103 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\textsc{and}\ \textsc{false})\textsc{true})$.

4.5.9.3 Cuando AND recibe datos que no son ni TRUE ni FALSE

Primero, recordemos el esquema correspondiente a la definición de la función AND en el λ-cálculo:

$$\textsc{and} \equiv (\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textsc{false})))$$

donde δ_1 y δ_2 representan dos variables. Habitualmente, en vez de δ_1 y δ_2 se pondrán variables concretas, por ejemplo, v y w:

$$\textsc{and} \equiv (\lambda v.(\lambda w.((v\ w)\textsc{false})))$$

En el apartado anterior se ha mostrado qué ocurre si a la función $(\lambda v.(\lambda w.((v\ w)\textsc{false})))$ se le pasan los datos TRUE eta TRUE y qué ocurre si se le pasan los datos FALSE eta TRUE. Pero en el λ-cálculo no hay ninguna herramienta para controlar o detectar la idoneicidad de los datos de entrada. Debido a ello, a la función $(\lambda v.(\lambda w.((v\ w)\textsc{false})))$ en vez de pasarle dos datos del conjunto $\{\textsc{true}, \textsc{false}\}$, se le pueden pasar dos datos cualesquiera. Desde el punto de vista del λ-cálculo, eso no generará ningún problema y se aplicará la β-reducción sin ningún problema. Pero el resultado que se obtenga puede ser incoherente con respecto a la idea con la que se ha definido la función AND.

A modo de ejemplo, consideramos el siguiente término:

$$(((\underbrace{(\lambda v.(\lambda w.((v\ w)\textsc{false})))}_{\textsc{and}}\ \underbrace{(\lambda z.z)}_{\delta_1}\ \underbrace{(\lambda x.((xx)y))}_{\delta_2})$$

En la figura 4.5.15 de la página 104 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\textsc{and}\ (\lambda z.z))(\lambda x.((xx)y)))$. La forma β-normal es y. Por tanto, no se ha obtenido ni TRUE ni FALSE.

4.5.10 El operador $\vee$ en el λ-cálculo: OR

En este apartado se implementará el operador $\vee$ mediante una función del λ-cálculo. Denominaremos OR a esa función.

4.5.10.1 OR: definición

En el caso de la disyunción, es decir, en el caso del operador $\vee$, ese operador recibe como datos dos expresiones δ_1 y δ_2 de valor booleano. Si el valor de alguna de esas dos expresiones es $True$, entonces el valor de la expresión $(\delta_1 \vee \delta_2)$ será $True$ y, en caso contrario, el valor de la expresión $(\delta_1 \vee \delta_2)$ será $False$. En el λ-cálculo, el operador $\vee$ ha de ser definido o expresado mediante una abstracción. La función o término se denominará OR. La función OR tendrá como datos de entrada dos valores booleanos δ_1 y δ_2. Si el valor del parámetro δ_1 es TRUE o el valor del parámetro δ_2 es TRUE, entonces la función OR ha de devolver el valor TRUE. En cambio, si el valor del parámetro δ_1 es FALSE y el valor del parámetro δ_2 es FALSE, entonces la función OR ha de devolver el valor FALSE.

El esquema correspondiente a la definición de OR en el λ-cálculo es el siguiente:

$$\text{OR} \equiv (\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \text{TRUE})\delta_2)))$$

donde δ_1 y δ_2 representan dos variables.

El término $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \text{TRUE})\delta_2)))$ es la definición del operador lógico $\vee$ en el λ-cálculo. Tal como se puede apreciar en esa definición, ese término tiene dos parámetros: δ_1 y δ_2. Para utilizar la función OR, se le han de pasar dos expresiones booleanas. Esas expresiones booleanas sustituirán a los parámetros δ_1 y δ_2. Tras aplicar β-reducción todas las veces que haga falta, esas expresiones lógicas quedarán reducidas a TRUE o FALSE. Por tanto, podemos entender que los parámetros δ_1 y δ_2 representan, cada uno de ellos, el valor TRUE o el valor FALSE. El significado del término $((\delta_1\ \text{TRUE})\delta_2))$ es el siguiente: la función δ_1 ha de elegir entre TRUE y δ_2; si el valor de δ_1 es TRUE, la función δ_1 elegirá el primero de sus dos argumentos, es decir, TRUE y TRUE será el resultado final; en cambio, si el valor de δ_1 es FALSE, la función δ_1 elegirá el segundo de sus dos argumentos, es decir, δ_2 y δ_2 será el resultado final —recordemos que δ_2 es TRUE o FALSE:

$$((\delta_1\ \text{TRUE})\delta_2))$$

La función δ_1 elegirá entre TRUE y δ_2.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \text{TRUE})\delta_2)))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v y w en vez de δ_1 y δ_2:

$$\text{OR} \equiv (\lambda v.(\lambda w.((v\ \text{TRUE})w)))$$

Puesto que TRUE es una abreviatura, en vez de TRUE podemos poner $(\lambda h.(\lambda g.h))$:

$$\text{OR} \equiv (\lambda v.(\lambda w.((v \underbrace{(\lambda h.(\lambda g.h))}_{\text{TRUE}})w)))$$

En esa función, el valor de v será TRUE o FALSE. Consecuentemente, v elegirá entre dos valores, en concreto, elegirá entre TRUE y w. Si el valor de v es TRUE, entonces elegirá TRUE. Si el valor de v es FALSE, entonces elegirá w. Si la función v ha elegido el primer argumento, es decir, TRUE, el resultado final es TRUE. Si la función v ha elegido el segundo argumento, es decir, w, entonces el resultado final es w. Recordemos que el valor de w será TRUE o FALSE. En resumen, si el valor de v es TRUE o el valor de w es TRUE, entonces el resultado final final será TRUE; en cambio, si el valor de v es FALSE y el valor de w es FALSE, entonces el resultado final será FALSE.

En vez de las variables v, w, h y g, se pueden utilizar cuatro variables cualesquiera, por ejemplo, a, b, c y d:

$$\text{OR} \equiv (\lambda a.(\lambda b.((a \underbrace{(\lambda c.(\lambda d.c))}_{\text{TRUE}})w)))$$

El término $(\lambda v.(\lambda w.((v\ \ (\lambda x.(\lambda y.x)))w)))$ y el término $(\lambda a.(\lambda b.((a\ \ (\lambda c.(\lambda d.c)))b)))$ son α-equivalentes. Debido a ello, da igual qué variables se utilicen.

4.5.11 El operador $\rightarrow$ en el λ-cálculo: IMP

En este apartado se implementará el operador $\rightarrow$ mediante una función del λ-cálculo. Denominaremos IMP a esa función.

4.5.11.1 IMP: definición

En el caso de la implicación, es decir, en el caso del operador $\rightarrow$, ese operador recibe como datos dos expresiones δ_1 y δ_2 de valor booleano. Si el valor de la primera expresión es $True$ y el valor de la segunda expresión es $False$, entonces el valor de la expresión $(\delta_1 \rightarrow \delta_2)$ será $False$ y, en caso contrario, el valor de la expresión $(\delta_1 \rightarrow \delta_2)$ será $True$. En el λ-cálculo, el operador $\rightarrow$ ha de ser definido o expresado mediante una abstracción. La función o término se denominará IMP. La función IMP tendrá como datos de entrada dos valores booleanos δ_1 y δ_2. Si el valor del parámetro δ_1 es TRUE y el valor del parámetro δ_2 es FALSE, entonces la función IMP ha de devolver el valor FALSE. En cambio, para cualquier otro caso, la función IMP ha de devolver el valor TRUE.

El esquema correspondiente a la definición de IMP en el λ-cálculo es el siguiente:

$$\text{IMP} \equiv (\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\text{TRUE})))$$

donde δ_1 y δ_2 representan dos variables.

El término $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textsc{true})))$ es la definición del operador lógico $\rightarrow$ en el λ-cálculo. Tal como se puede apreciar en esa definición, ese término tiene dos parámetros: δ_1 y δ_2. Para utilizar la función IMP, se le han de pasar dos expresiones booleanas. Esas expresiones booleanas sustituirán a los parámetros δ_1 y δ_2. Tras aplicar β-reducción todas las veces que haga falta, esas expresiones lógicas quedarán reducidas a TRUE o FALSE. Por tanto, podemos entender que los parámetros δ_1 y δ_2 representan, cada uno de ellos, el valor TRUE o el valor FALSE. El significado del término $((\delta_1\ \delta_2)\textsc{true})$ es el siguiente: la función δ_1 ha de elegir entre δ_2 y TRUE; si el valor de δ_1 es TRUE, la función δ_1 elegirá el primero de sus dos argumentos, es decir, δ_2 y δ_2 será el resultado final —recordemos que δ_2 es TRUE o FALSE; en cambio, si el valor de δ_1 es FALSE, la función δ_1 elegirá el segundo de sus dos argumentos, es decir, TRUE y TRUE será el resultado final:

$$((\delta_1\ \delta_2)\textsc{true})$$

La función δ_1 elegirá entre δ_2 y TRUE.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)\textsc{true})))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v y w en vez de δ_1 y δ_2:

$$(\lambda v.(\lambda w.((v\ w)\textsc{true})))$$

Puesto que TRUE es una abreviatura, en vez de TRUE podemos poner $(\lambda h.(\lambda g.h))$:

$$\textsc{imp} \equiv (\lambda v.(\lambda w.((v\ w)\underbrace{(\lambda h.(\lambda g.h))}_{\textsc{true}})))$$

En esa función, el valor de v será TRUE o FALSE. Consecuentemente, v elegirá entre dos valores, en concreto, elegirá entre w y TRUE. Si el valor de v es TRUE, entonces elegirá w. Si el valor de v es FALSE, entonces elegirá TRUE. Si la función v ha elegido el primer argumento, es decir, w, entonces el resultado final es w. Recordemos que el valor de w será TRUE o FALSE. Si la función v ha elegido el segundo argumento, es decir, TRUE, el resultado final es TRUE. En resumen, si el valor de v es FALSE o el valor de w es TRUE, entonces el resultado final final será TRUE; en cambio, si el valor de v es TRUE2 y el valor de w es FALSE, entonces el resultado final será FALSE.

En vez de las variables v, w, h y g, se pueden utilizar cuatro variables cualesquiera, por ejemplo, a, b, c y d:

$$\textsc{imp} \equiv (\lambda a.(\lambda b.((a\ b)\underbrace{(\lambda c.(\lambda d.c))}_{\textsc{true}})))$$

El término $(\lambda v.(\lambda w.((v\ w)(\lambda h.(\lambda g.h))$ y el término $(\lambda a.(\lambda b.((a\ b)(\lambda c.(\lambda d.c))$ son α-equivalentes. Debido a ello, da igual qué variables se utilicen.

4.5.12 El operador $\leftrightarrow$ en el λ-cálculo: EQUI

En este apartado se implementará el operador $\leftrightarrow$ mediante una función del λ-cálculo. Denominaremos EQUI a esa función.

4.5.12.1 EQUI: definición

En el caso de la equivalencia, es decir, en el caso del operador $\leftrightarrow$, ese operador recibe como datos dos expresiones δ_1 y δ_2 de valor booleano. Si el valor de las dos expresiones es $True$ o el valor de las dos expresiones es $False$, entonces el valor de la expresión $(\delta_1 \leftrightarrow \delta_2)$ será $True$ y, en caso contrario, el valor de la expresión $(\delta_1 \leftrightarrow \delta_2)$ será $False$. En el λ-cálculo, el operador $\leftrightarrow$ ha de ser definido o expresado mediante una abstracción. La función o término se denominará EQUI. La función EQUI tendrá como datos de entrada dos valores booleanos δ_1 y δ_2. Si el valor de los dos parámetros es TRUE o el valor de los dos parámetros es FALSE, entonces la función EQUI ha de devolver el valor TRUE. En cambio, para cualquier otro caso, la función EQUI ha de devolver el valor FALSE.

El esquema correspondiente a la definición de EQUI en el λ-cálculo es el siguiente:

$$\text{EQUI} \equiv (\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)(\text{NOT}\ \delta_2))))$$

donde δ_1 y δ_2 representan dos variables.

El término $(\lambda\delta_1.(\lambda\delta_2.((\delta_1\ \delta_2)(\text{NOT}\ \delta_2))))$ es la definición del operador lógico $\leftrightarrow$ en el λ-cálculo. Tal como se puede apreciar en esa definición, ese término tiene dos parámetros: δ_1 y δ_2. Para utilizar la función EQUI, se le han de pasar dos expresiones booleanas. Esas expresiones booleanas sustituirán a los parámetros δ_1 y δ_2. Tras aplicar β-reducción todas las veces que haga falta, esas expresiones lógicas quedarán reducidas a TRUE o FALSE. Por tanto, podemos entender que los parámetros δ_1 y δ_2 representan, cada uno de ellos, el valor TRUE o el valor FALSE. El significado del término $((\delta_1\ \delta_2)(\text{NOT}\ \delta_2))$ es el siguiente: la función δ_1 ha de elegir entre δ_2 y $(\text{NOT}\ \delta_2)$; si el valor de δ_1 es TRUE, la función δ_1 elegirá el primero de sus dos argumentos, es decir, δ_2 y δ_2 será el resultado final —recordemos que δ_2 es TRUE o FALSE; en cambio, si el valor de δ_1 es FALSE, la función δ_1 elegirá el segundo de sus dos argumentos, es decir, $(\text{NOT}\ \delta_2)$ y el resultado final será $(\text{NOT}\ \delta_2)$, pero recordemos que $(\text{NOT}\ \delta_2)$ es TRUE o FALSE:

$$((\delta_1\ \delta_2)(\text{NOT}\ \delta_2))$$

La función δ_1 elegirá entre δ_2 y $(\text{NOT}\ \delta_2)$.

En los ejemplos concretos, los elementos del esquema general $((\delta_1\ \delta_2)(\text{NOT}\ \delta_2))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner v y w en vez de δ_1 y δ_2:

$$(\lambda v.(\lambda w.((v\ w)(\text{NOT}\ w))))$$

Puesto que NOT es una abreviatura, en vez de NOT podemos poner $(\lambda f.((f\ \textsc{false})\textsc{true}))$:

$$\textsc{equi} \equiv (\lambda v.(\lambda w.((v\ w)(\underbrace{(\lambda f.((f\ \textsc{false})\textsc{true}))}_{\textsc{not}}\ w))))$$

También TRUE y FALSE son abreviaturas, en vez de TRUE podemos poner $(\lambda h.(\lambda g.h))$ y en vez de FALSE podemos poner $(\lambda j.(\lambda n.n))$:

$$\textsc{equi} \equiv (\lambda v.(\lambda w.((v\ w)(\underbrace{(\lambda f.((f\ \underbrace{(\lambda j.(\lambda n.n))}_{\textsc{false}}\ \underbrace{(\lambda h.(\lambda g.h))}_{\textsc{true}}))\ w))}_{\textsc{not}}))))$$

En esa función, el valor de v será TRUE o FALSE. Consecuentemente, v elegirá entre dos valores, en concreto, elegirá entre w y $(\textsc{not}\ w)$. Si el valor de v es TRUE, entonces elegirá w. Si el valor de v es FALSE, entonces elegirá $(\textsc{not}\ w)$. Si la función v ha elegido el primer argumento, es decir, w, entonces el resultado final es w. Recordemos que el valor de w será TRUE o FALSE. Si la función v ha elegido el segundo argumento, es decir, $(\textsc{not}\ w)$, el resultado final es $(\textsc{not}\ w)$. Recordemos que el valor de $(\textsc{not}\ w)$ será TRUE o FALSE. En resumen, si el valor de v y el valor de w son iguales, entonces el resultado final será TRUE; en cambio, si el valor de v y el valor de w son distintos, entonces el resultado final será FALSE.

$$(\lambda\, h.(\lambda\, g.((\underbrace{\ s\ }\ h)g))) \equiv (\lambda\, h.(\lambda\, g.((\ \underbrace{\textbf{FALSE}}_{\mathsf{Contexto}(s)}\ h)g))) \equiv$$

$$\equiv (\lambda\, h.(\lambda\, g.(\underbrace{(\underbrace{(\lambda x.(\lambda y.y))}_{\textbf{FALSE}}\ h)\, g)})))\ \rightarrow_\beta$$

x será sustituida por h

$\rightarrow_\beta\ (\lambda\, h.(\lambda\, g.\underbrace{((\lambda y.y)g)}))\ \rightarrow_\beta$

y será sustituida por g

$$\rightarrow_\beta \underbrace{(\lambda\, h.(\lambda\, g.g))}_{\textbf{FALSE}}$$

Está en forma β-normal.

Figura 4.5.7. Desarrollo del término (ITE s) cuando el valor de $\mathsf{Contexto}(s)$ es **FALSE**.

$$\underbrace{((\lambda v.((v\ \underbrace{\mathbf{FALSE}}_{\mu}) \underbrace{\mathbf{TRUE}}_{\chi}))}_{\mathbf{NOT}} \underbrace{\mathbf{TRUE}}_{\sigma}) \rightarrow_\beta$$

v será sustituida por σ

$$((\underbrace{\mathbf{TRUE}}_{\sigma}\ \underbrace{\mathbf{FALSE}}_{\mu}) \underbrace{\mathbf{TRUE}}_{\chi}) \equiv ((\underbrace{(\lambda h.(\lambda g.h))}_{\mathbf{TRUE}}\ \underbrace{\mathbf{FALSE}}_{\mu}) \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

h será sustituida por μ

$$((\lambda g.\underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

g será sustituida por χ

$$\underbrace{\mathbf{FALSE}}_{\mu}$$

Está en forma β-normal.

Figura 4.5.8. Cálculo de la forma β-normal correspondiente al término (NOT TRUE).

$$\underbrace{((\lambda v.((v\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}))}_{\mathbf{NOT}}\ \underbrace{\mathbf{FALSE}}_{\sigma}) \rightarrow_\beta$$

v será sustituida por σ

$$((\underbrace{\mathbf{FALSE}}_{\sigma}\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \equiv ((\underbrace{(\lambda h.(\lambda g.g))}_{\mathbf{FALSE}}\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

h será sustituida por μ

$$((\lambda g.g)\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

g será sustituida por χ

$$\underbrace{\mathbf{TRUE}}_{\chi}$$

Está en forma β-normal.

Figura 4.5.9. Cálculo de la forma β-normal correspondiente al término (**NOT FALSE**).

$$\underbrace{((\lambda v.((v\ \underbrace{\textbf{FALSE}}_{\mu})\ \underbrace{\textbf{TRUE}}_{\chi}))}_{\textbf{NOT}}\ \underbrace{(\lambda j.j)}_{\sigma})\ \rightarrow_\beta$$

v será sustituida por σ

$$((\underbrace{(\lambda j.j)}_{\sigma}\ \underbrace{\textbf{FALSE}}_{\mu})\ \underbrace{\textbf{TRUE}}_{\chi})\ \rightarrow_\beta$$

j será sustituida por μ

$$(\textbf{FALSE}\ \underbrace{\textbf{TRUE}}_{\chi})\ \equiv (\underbrace{(\lambda h.(\lambda g.h))}_{\textbf{FALSE}}\ \underbrace{\textbf{TRUE}}_{\chi})\ \rightarrow_\beta$$

h será sustituida por χ

$$(\lambda g.\underbrace{\textbf{TRUE}}_{\chi})$$

Está en forma β-normal.

Figura 4.5.10. Cálculo de la forma β-normal correspondiente al término $(\textbf{NOT}\ (\lambda j.j))$.

$$\underbrace{((\lambda v.((v\ \underbrace{\textbf{FALSE}}_{\mu}\ \underbrace{\textbf{TRUE}}_{\chi}))}_{\textbf{NOT}}\ \underbrace{(\lambda j.r)}_{\sigma}) \rightarrow_\beta$$

v será sustituida por σ

$$(((\underbrace{\lambda j.r}_{\sigma})\ \underbrace{\textbf{FALSE}}_{\mu})\ \underbrace{\textbf{TRUE}}_{\chi}) \rightarrow_\beta$$

j será sustituida por μ

$$(r\ \underbrace{\textbf{TRUE}}_{\chi}) \rightarrow_\beta$$

Está en forma β-normal.

Figura 4.5.11. Cálculo de la forma β-normal correspondiente al término $(\textbf{NOT}\ (\lambda j.r))$.

$$\underbrace{((\lambda v.((v\ \underbrace{\mathbf{FALSE}}_{\mu}) \underbrace{\mathbf{TRUE}}_{\chi}))}_{\mathbf{NOT}}\ \underbrace{((\lambda x.((xx)y))(\lambda x.((xx)y)))}_{\sigma}) \rightarrow_\beta$$

v será sustituida por σ

$$(((\underbrace{(\lambda x.((xx)y))(\lambda x.((xx)y))}_{\sigma})\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

x será sustituida por $(\lambda x.((xx)y))$

$$((((\underbrace{(\lambda x.((xx)y))(\lambda x.((xx)y))}_{\sigma})\, y)\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

x será sustituida por $(\lambda x.((xx)y))$

$$(((((\underbrace{(\lambda x.((xx)y))(\lambda x.((xx)y))}_{\sigma})\, y)y)\ \underbrace{\mathbf{FALSE}}_{\mu})\ \underbrace{\mathbf{TRUE}}_{\chi}) \rightarrow_\beta$$

x será sustituida por $(\lambda x.((xx)y))$

...

proceso infinito

Figura 4.5.12. Cálculo de la forma β-normal correspondiente al término $(\mathbf{NOT}\ ((\lambda x.((xx)y))(\lambda x.((xx)y))))$.

$$((\underbrace{(\lambda v.(\lambda w.((v\ w)\mathbf{FALSE})))}_{\mathbf{AND}}\ \underbrace{\mathbf{TRUE}}_{\delta_1})\ \underbrace{\mathbf{TRUE}}_{\delta_2})\ \rightarrow_\beta$$

v será sustituida por δ_1

$$\rightarrow_\beta\ \underbrace{((\lambda w.((\underbrace{\mathbf{TRUE}}_{\delta_1}\ w)\mathbf{FALSE}))}_{\delta_1}\ \underbrace{\mathbf{TRUE}}_{\delta_2})\ \rightarrow_\beta$$

w será sustituida por δ_2

$$\rightarrow_\beta\ ((\underbrace{\mathbf{TRUE}}_{\delta_1}\ \underbrace{\mathbf{TRUE}}_{\delta_2})\mathbf{FALSE}) \equiv ((\underbrace{(\lambda x.(\lambda y.x))}_{\mathbf{TRUE}}\ \underbrace{\mathbf{TRUE}}_{\delta_2})\mathbf{FALSE})\ \rightarrow_\beta$$

x será sustituida por δ_2

$$\rightarrow_\beta\ ((\lambda y.\underbrace{\mathbf{TRUE}}_{\delta_2})\mathbf{FALSE})\ \rightarrow_\beta$$

y será sustituida por **FALSE**

$$\rightarrow_\beta\ \underbrace{\mathbf{TRUE}}_{\delta_2}$$

Está en forma β-normal.

Figura 4.5.13. Cálculo de la forma β-normal correspondiente al término $((\mathbf{AND}\ \mathbf{TRUE})\mathbf{TRUE})$.

$$(((\underbrace{(\lambda v.(\lambda w.((v\ w)\mathbf{FALSE})))}_{\mathbf{AND}})\underbrace{\mathbf{FALSE}}_{\delta_1})\underbrace{\mathbf{TRUE}}_{\delta_2}) \rightarrow_\beta$$

v será sustituida por δ_1

$$\rightarrow_\beta \underbrace{((\lambda w.((\underbrace{\mathbf{FALSE}}_{\delta_1}\ w)\mathbf{FALSE}))}\underbrace{\mathbf{TRUE}}_{\delta_2}) \rightarrow_\beta$$

w será sustituida por δ_2

$$\rightarrow_\beta ((\underbrace{\mathbf{FALSE}}_{\delta_1}\ \underbrace{\mathbf{TRUE}}_{\delta_2})\mathbf{FALSE}) \equiv (((\underbrace{(\lambda x.(\lambda y.y))}_{\mathbf{FALSE}}\ \underbrace{\mathbf{TRUE}}_{\delta_2})\mathbf{FALSE}) \rightarrow_\beta$$

x será sustituida por δ_2

$$\rightarrow_\beta ((\lambda y.y)\mathbf{FALSE}) \rightarrow_\beta$$

y será sustituida por **FALSE**

$$\rightarrow_\beta \mathbf{FALSE}$$

Está en forma β-normal.

Figura 4.5.14. Cálculo de la forma β-normal correspondiente al término $((\mathbf{AND}\ \mathbf{FALSE})\mathbf{TRUE})$.

$$(((\underbrace{(\lambda v.(\lambda w.((v\ w)\mathbf{FALSE})))}_{\mathbf{AND}})\underbrace{(\lambda z.z)}_{\delta_1})\underbrace{(\lambda x.((xx)y))}_{\delta_2}) \rightarrow_\beta$$

v será sustituida por δ_1

$$\rightarrow_\beta ((\underbrace{(\lambda w.(((\underbrace{\lambda z.z}_{\delta_1})\ w)\mathbf{FALSE}))}_{\delta_1})\underbrace{(\lambda x.((xx)y))}_{\delta_2}) \rightarrow_\beta$$

w será sustituida por δ_2

$$\rightarrow_\beta (((\underbrace{\lambda z.z}_{\delta_1})\ \underbrace{(\lambda x.((xx)y))}_{\delta_2})\mathbf{FALSE}) \rightarrow_\beta$$

z será sustituida por δ_2

$$\rightarrow_\beta ((\lambda x.((xx)y))\mathbf{FALSE}) \rightarrow_\beta$$

x será sustituida por **FALSE**

$$\rightarrow_\beta ((\mathbf{FALSE}\ \mathbf{FALSE})y) \equiv (((\underbrace{\lambda a.(\lambda b.b)}_{\mathbf{FALSE}})\ \mathbf{FALSE})y) \rightarrow_\beta$$

a será sustituida por **FALSE**

$$\rightarrow_\beta ((\lambda b.b)\ y) \rightarrow_\beta$$

b será sustituida por y

$$\rightarrow_\beta y$$

Está en forma β-normal.

Figura 4.5.15. Cálculo de la forma β-normal correspondiente al término $((\mathbf{AND}\ (\lambda z.z))(\lambda x.((xx)y)))$.

4.6.

Números naturales y operaciones sobre naturales en el λ-cálculo

4.6.1 Definición algebraica de los números naturales: $Cero$ y S

La definición algebraica de los números naturales se puede formular de la siguuiente manera:

$$
\begin{array}{l}
0 : Cero \\
1 : S(Cero) \\
2 : S(S(Cero)) \\
3 : S(S(S(Cero))) \\
\ldots \\
n : S(n-1) \\
\ldots
\end{array}
$$

Por tanto, teniendo en cuenta esa formulación, los números naturales se representarían como $Cero$, $S(Cero)$, $S(S(Cero))$, $S(S(S(Cero)))$, etc.

El significado del elemento S es "siguiente": el número 1 es el siguiente del 0, el número 2 es el siguiente del siguiente del 0, etc.

Tal como se puede apreciar en esos ejemplos, todos los números naturales se ajustan a una de estas dos estructuras: directamente $Cero$ o, si no, $S(m)$, donde m es un número natural.Por tanto, el número $S(m)$ es el siguiente del número m.

Las operaciones sobre números naturales se pueden definir utilizando recursividad. Por ejemplo, la operación $siguiente(x)$ que calcula el valor que sigue a x, es decir, $x+1$, se define, mediante dos ecuaciones, de la siguiente forma:

$$
\begin{array}{ll}
siguiente(Cero) = S(Cero) & (1^a\ ec.) \\
siguiente(S(m)) = S(S(m)) & (2^a\ ec.)
\end{array}
$$

Teniendo en cuenta esas ecuaciones, el resultado de la expresión $siguiente(S(S(S(Cero))))$ sería espresioaren emaitza $S(S(S(S(Cero))))$. Es decir, se ha calculado el siguiente del número 3 y el resultado obtenido es 4.

La operación $suma(x, y)$ que, dados dos números naturales x e y, calcula la el valor de $x + y$, se define, mediante dos ecuaciones, de la siguiente forma:

$$\begin{array}{ll} suma(Cero, n) = n & (1^a\ ec.) \\ suma(S(m), n) = S(suma(m, n)) & (2^a\ ec.) \end{array}$$

Teniendo en cuenta esas ecuaciones, el proceso para calcular la suma de los números naturales $S(S(Cero))$ y $S(S(S(Cero)))$ es el siguiente:

$$\begin{array}{ll} suma(S(S(Cero)), S(S(S(Cero)))) = & (2^a\ ec.) \\ = S(suma(S(Cero), S(S(S(Cero))))) = & (2^a\ ec.) \\ = S(S(suma(Cero, S(S(S(Cero)))))) = & (1^a\ ec.) \\ = S(S(S(S(S(Cero))))) & \end{array}$$

Se han sumado los números 2 y 3; el resultado es 5.

La operación $prod(x, y)$ que, dados dos números naturales x e y, calcula el valor de $x * y$, es decir, el producto de x e y, se define, mediante dos ecuaciones, de la siguiente forma:

$$\begin{array}{ll} prod(Cero, n) = Cero & (1^a\ ec.) \\ prod(S(m), n) = suma(n, prod(m, n)) & (2^a\ ec.) \end{array}$$

El significado de la segunda ecuación es el siguiente: puesto que multiplicar $H(m)$ por n consiste en sumar el número $n\ m+1$ veces, si lo planteamos recursivamente tenemos que podemos sumar primero el número $n\ m$ veces —calculando así $prod(m, n)$—, y finalmente sumar n una vez más. De esa manera, se habrá sumado el número $n\ m + 1$ veces.

La operación $anterior(x)$ que calcula el valor anterior a x, es decir, $x - 1$, se define, mediante dos ecuaciones, de la siguiente forma:

$$\begin{array}{ll} anterior(Cero) = Cero & (1^a\ ec.) \\ anterior(S(m)) = m & (2^a\ ec.) \end{array}$$

Si el número x es positivo, se calculará el valor $x - 1$, pero si x es cero, se devolverá el propio cero como resultado, puesto que al estar trabajando con números naturales, no es posible utilizar números negativos.

La operación $resta(x, y)$ que, dados dos números naturales x e y, calcula la el valor de $x - y$, se define, mediante dos ecuaciones, de la siguiente forma:

$$\begin{array}{ll} resta(Cero, Cero) = Cero & (1^a\ ec.) \\ resta(Cero, S(n)) = Cero & (2^a\ ec.) \\ resta(S(m), Cero) = S(m) & (3^a\ ec.) \\ resta(S(m), S(n)) = resta(m, n)) & (4^a\ ec.) \end{array}$$

Al calcular $x - y$, si se cumple $x < y$, entonces el resultado será $Cero$, puesto que al estar trabajando con números naturales, no es posible utilizar números negativos.

4.6.2 Definición de los números naturales en el λ-cálculo

En el λ-cálculo, cada número es una función. La definición de los números en el λ-cálculo se asemeja a la definición algebraica, pero la idea subyacente es todavía más general: en vez de $Cero$, se puede poner cualquier función y en vez de S se puede poner también cualquier función.

- El número 0:

  $$(\lambda\delta.(\lambda\gamma.\gamma))$$

  donde δ y γ son dos variables.

  Denotaremos esa función como $\mathbb{0}$:

  $$\mathbb{0} \equiv (\lambda\delta.(\lambda\gamma.\gamma))$$

  El significado de la función $\mathbb{0}$ es el siguiente: dadas dos funciones δ y γ, se devolverá como resultado el término que se obtenga aplicando cero veces la función δ a la función γ. Esto significa que se devolverá directamente γ.

  En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.(\lambda\gamma.\gamma))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h y g en vez de δ y γ:

  $$\mathbb{0} \equiv (\lambda h.(\lambda g.g))$$

- El número 1:

  $$(\lambda\delta.(\lambda\gamma.(\delta\ \gamma)))$$

  donde δ y γ son dos variables.

  Denotaremos esa función como $\mathbb{1}$:

  $$\mathbb{1} \equiv (\lambda\delta.(\lambda\gamma.(\delta\ \gamma)))$$

  El significado de la función $\mathbb{1}$ es el siguiente: dadas dos funciones δ y γ, se devolverá como resultado el término que se obtenga aplicando una vez la función δ a la función γ.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.(\lambda\gamma.(\delta\ \gamma)))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h y g en vez de δ y γ:

$$\mathbb{1} \equiv (\lambda h.(\lambda g.(hg)))$$

- El número 2:

$$(\lambda\delta.(\lambda\gamma.(\delta(\delta\ \gamma))))$$

donde δ y γ son dos variables.

Denotaremos esa función como $\mathbb{2}$:

$$\mathbb{2} \equiv (\lambda\delta.(\lambda\gamma.(\delta(\delta\ \gamma))))$$

El significado de la función $\mathbb{2}$ es el siguiente: dadas dos funciones δ y γ, se devolverá como resultado el término que se obtenga aplicando dos veces la función δ a la función γ.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.(\lambda\gamma.(\delta(\delta\ \gamma))))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h y g en vez de δ y γ:

$$\mathbb{2} \equiv (\lambda h.(\lambda g.(h(h\ g))))$$

- El número 3:

$$(\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta\ \gamma)))))$$

donde δ y γ son dos variables.

Denotaremos esa función como $\mathbb{3}$:

$$\mathbb{3} \equiv (\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta\ \gamma)))))$$

El significado de la función $\mathbb{3}$ es el siguiente: dadas dos funciones δ y γ, se devolverá como resultado el término que se obtenga aplicando tres veces la función δ a la función γ.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta\ \gamma)))))$

serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h y g en vez de δ y γ:

$$\mathbb{3} \equiv (\lambda h.(\lambda g.(h(h(h\ g)))))$$

- El número 4:

$$(\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta(\delta\ \gamma))))))$$

donde δ y γ son dos variables.

Denotaremos esa función como $\mathbb{4}$:

$$\mathbb{4} \equiv (\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta(\delta\ \gamma))))))$$

El significado de la función $\mathbb{4}$ es el siguiente: dadas dos funciones δ y γ, se devolverá como resultado el término que se obtenga aplicando cuatro veces la función δ a la función γ.

En los ejemplos concretos, los elementos del esquema general $(\lambda\delta.(\lambda\gamma.(\delta(\delta(\delta(\delta\ \gamma))))))$ serán reemplazados por las variables que nos interesen. A modo de muestra, podemos poner h y g en vez de δ y γ:

$$\mathbb{4} \equiv (\lambda h.(\lambda g.(h(h(h(h\ g))))))$$

- Y así con todos los números naturales restantes.

4.6.3 El operador lógico F y el número cero son iguales en λ-cálculo: NOT y 𝕆

En el λ-cálculo, la definición del operador lógico F y la definición del número cero coinciden:

$$\text{NOT} \equiv \mathbb{0} \equiv (\lambda\delta.(\lambda\gamma.\gamma))$$

4.6.4 El número siguiente con respecto a un número dado: SIG

4.6.4.1 SIG: definición

La función SIG reuiqere tres datos de entrada:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una función para ocupar el lugar del parámetro δ en la definición de μ. Le llamaremos φ.

(3) Una función para ocupar el lugar del parámetro γ en la definición de μ. Le llamaremos ψ.

El esquema correspondiente a la definición de SIG en el λ-cálculo es el siguiente:

$$\text{SIG} \equiv (\lambda\mu.(\lambda\varphi.(\lambda\psi.(\varphi((\mu\ \varphi)\psi)))))$$

Dados un número μ y dos funciones φ y ψ, el subtérmino $((\mu\ \varphi)\psi)$ expresa lo siguiente: se aplicará μ veces la función φ a la función ψ. Pero delante del subtérmino $((\mu\ \varphi)\psi)$ se ha puesto φ y el subtérmino $(\varphi((\mu\ \varphi)\psi))$ expresa lo siguiente: se aplicará μ veces y vez más la función φ a la función ψ.

En los ejemplos concretos, los elementos del esquema general $(\lambda\mu.(\lambda\varphi.(\lambda\psi.(\varphi((\mu\ \varphi)\psi)))))$ serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{SIG} \equiv (\lambda m.(\lambda v.(\lambda w.(v((m\ v)w)))))$$

4.6.4.2 Ejemplos de uso de SIG

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función $(\lambda m.(\lambda v.(\lambda w.(v((m\ v)w)))))$

A modo de ejemplo, consideramos el siguiente término:

$$((\lambda m.(\lambda v.(\lambda w.(v((m\ v)w)))))(\lambda h.(\lambda g.(h(h\ g)))))$$

Primero identificamos los componentes:

$$(\underbrace{(\lambda m.(\lambda v.(\lambda w.(v((m\ v)w)))))}_{\text{SIG}}\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{2})$$

Por tanto, estamos ante el término $(\text{SIG}\ 2)$ y el resultado esperado sería el término 3. En la figura 4.6.1 de la página 111 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(\text{SIG}\ 2)$.

4.6.5 La suma en el λ-cálculo: SUMAR y SUMAR2

4.6.5.1 Definición de SUMAR

La función SUMAR requiere cuatro datos:

$$(\underbrace{(\lambda m.(\lambda v.(\lambda w.(v((m\ v)w)))))}_{\mathbf{SIG}}\ \underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}})\ \rightarrow_\beta$$

m será sustituida por $\mathbb{2}$

$$\rightarrow_\beta\ (\lambda v.(\lambda w.(v(\underbrace{(\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ v)}w))))\ \rightarrow_\beta$$

h será sustituida por v

$$\rightarrow_\beta\ (\lambda v.(\lambda w.(v(\underbrace{(\lambda g.(v(v\ g)))\ w}))))\ \rightarrow_\beta$$

g será sustituida por v

$$\rightarrow_\beta\ \underbrace{(\lambda v.(\lambda w.(v(v(v\ w)))))}_{\mathbb{3}}$$

Está en forma β-normal.

Figura 4.6.1. Cálculo de la forma β-normal correspondiente al término (**SIG** $\mathbb{2}$).

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(3) Una función para ocupar el lugar del parámetro δ en las definiciones de μ y ρ. Le llamaremos φ.

(4) Una función para ocupar el lugar del parámetro γ en las definiciones de μ y ρ. Le llamaremos ψ.

El esquema correspondiente a la definición de SUMAR en el λ-cálculo es el siguiente:

$$\textbf{SUMAR} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.(\lambda\psi.((\mu\ \varphi)((\rho\ \varphi)\psi))))))$$

Dados dos números μ y ρ y dos funciones φ y ψ, el subtérmino $((\rho\ \varphi)\psi)$ expresa lo siguiente: se aplicará ρ veces la función φ a la función ψ. Pero delante del subtérmino $((\mu\ \varphi)\psi)$ se ha puesto $(\mu\ \varphi)$ y el subtérmino $((\mu\ \varphi)((\rho\ \varphi)\psi))$ expresa lo siguiente: primero se aplicará ρ veces la función φ a la función ψ; al término que se ontenga de esa manera, se le aplicará μ veces la función φ. En total, se aplicará μ veces más ρ veces la función φ a la función ψ.

En los ejemplos concretos, los elementos del esquema general

$$(\lambda\mu.(\lambda\rho.(\lambda\varphi.(\lambda\psi.((\mu\ \varphi)((\rho\ \varphi)\psi))))))$$

serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\textbf{SUMAR} \equiv (\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))$$

4.6.5.2 Ejemplos de uso de SUMAR

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función $(\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n$

A modo de ejemplo, consideramos el siguiente término:

$$(((\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))(\lambda h.(\lambda g.(h(h\ g)))))(\lambda x.(\lambda z.(x(x(x\ z))))))$$

Primero identificamos los componentes:

$$(((\underbrace{\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))}_{\textbf{SUMAR}})\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{2})\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{3})$$

Por tanto, estamos ante el término $((\text{SUMAR } \mathbb{2})\mathbb{3})$ y el resultado esperado sería el término $\mathbb{5}$. En la figura 4.6.2 de la página 114 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\text{SUMAR } \mathbb{2})\mathbb{3})$.

La definición algebraica dada para la función $sumar$ en el apartado 4.6.1 de la página 105, está basada en la recursividad, pero tal como se puede apreciar en la figura 4.6.2 de la página 114, la definición dada para la función $sumar$ en el λ-cálculo no está basada en la recursividad, sino que está basada en la operación de sustitución. El resultado se ha obtenido a base de insertar subtérminos dentro de otros subtérminos.

4.6.5.3 Otra opción para definir la suma: SUMAR2

La función SUMAR2 requiere dos datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

El esquema correspondiente a la definición de SUMAR2 en el λ-cálculo es el siguiente:

$$\text{SUMAR2} \equiv (\lambda\mu.(\lambda\rho.((\mu\ \text{SIG})\rho)))$$

Idea de la definición: se le aplicará μ veces la función SIG al número ρ.

La función SIG, dado un número, devuelve el siguiente número. El esquema correspondiente a la definición de SIG en el λ-cálculo es el siguiente:

$$\text{SIG} \equiv (\lambda\mu.(\lambda\varphi.(\lambda\psi.(\varphi((\mu\ \varphi)\psi)))))$$

La función SIG ha sido presentada y analizada en el apartado 4.6.4 de la página 109.

En la figura 4.6.3 de la página 115 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\text{SUMAR2 } \mathbb{2})\mathbb{3})$.

4.6.6 La multiplicación en el λ-cálculo: MULT, MULT2 y MULT3

4.6.6.1 Definición de MULT

La función MULT requiere tres datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

$$\underbrace{(\underbrace{(\underbrace{(\underbrace{\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))}_{\textbf{SUMAR}}\ \underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}})}\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})}\ \to_\beta$$

m será sustituida por $\mathbb{2}$

$$\to_\beta \underbrace{((\lambda n.(\lambda v.(\lambda w.(((\underbrace{\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ v)((n\ v)w)))))\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})}\ \to_\beta$$

n será sustituida por $\mathbb{3}$

$$\to_\beta (\lambda v.(\lambda w.(\underbrace{(\underbrace{\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ v)}((\underbrace{\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)w)))) \to_\beta$$

h será sustituida por v

$$\to_\beta (\lambda v.(\lambda w.(\underbrace{(\underbrace{\lambda g.(v(v\ g)))}_{\mathbb{2}}\ (((\underbrace{\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)w)))}_{\sigma}) \to_\beta$$

g será sustituida por σ

$$\to_\beta (\lambda v.(\lambda w.(((v(v\ (((\underbrace{\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)\ w)))))))) \to_\beta$$

x será sustituida por v

$$\to_\beta (\lambda v.(\lambda w.(v(v\ \underbrace{((\lambda z.(v(v(v\ z))))w)}))))) \to_\beta$$

z será sustituida por w

$$\to_\beta \underbrace{(\lambda v.(\lambda w.(v(v(v(v(v\ w)))))))}_{\mathbb{5}}$$

Está en forma β-normal.

Figura 4.6.2. Cálculo de la forma β-normal correspondiente al término $((\textsc{sumar}\ \mathbb{2})\mathbb{3})$.

$(((\underbrace{(\lambda m.(\lambda n.((m\ \textbf{SIG})n)))}_{\textbf{SUMAR2}}\ \mathbb{2})\mathbb{3})\ \rightarrow_\beta\ m$ será sustituida por $\mathbb{2}$

$\rightarrow_\beta\ (\underbrace{(\lambda n.((\mathbb{2}\ \textbf{SIG})n))}_{\textbf{SUMAR2}}\ \mathbb{3})\ \rightarrow_\beta\ n$ será sustituida por $\mathbb{3}$

$\rightarrow_\beta\ ((\mathbb{2}\ \textbf{SIG})\mathbb{3})\ \equiv\ \underbrace{(((\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ \textbf{SIG})\ \mathbb{3})\ \rightarrow_\beta}$

h será sustituida por SIG

$\rightarrow_\beta\ \underbrace{((\lambda g.(\textbf{SIG}(\textbf{SIG}\ g)))\mathbb{3})}\ \rightarrow_\beta\ g$ será sustituida por $\mathbb{3}$

$\rightarrow_\beta\ \underbrace{(\textbf{SIG}(\textbf{SIG}\ \mathbb{3}))}\ \equiv\ \underbrace{((\underbrace{(\lambda a.(\lambda b.(\lambda c.(b((a\ b)c)))))}_{\textbf{SIG}}(\textbf{SIG}\ \mathbb{3}))\ \rightarrow_\beta}$

a será sustituida por $(\textbf{HUR}\ \mathbb{3})$

$\rightarrow_\beta\ (\lambda b.(\lambda c.(b(((\textbf{SIG}\ \mathbb{3})\ b)c))))\ \equiv$

$\equiv\ (\lambda b.(\lambda c.(b(\underbrace{((\underbrace{(\lambda d.(\lambda e.(\lambda f.(e((d\ e)f)))))}_{\textbf{SIG}}\ \mathbb{3})\ b)}c))))\ \rightarrow_\beta$

d será sustituida por $\mathbb{3}$

$\rightarrow_\beta\ (\lambda b.(\lambda c.(b(\underbrace{((\lambda e.(\lambda f.(e((\mathbb{3}\ e)f))))\ b)}\ c))))\ \rightarrow_\beta\ e$ será sustituida por b

$\rightarrow_\beta\ (\lambda b.(\lambda c.(b\ \underbrace{((\lambda f.(b((\mathbb{3}\ b)f)))c)})))\ \rightarrow_\beta\ f$ será sustituida por c

$\rightarrow_\beta\ (\lambda b.(\lambda c.(b(b((\mathbb{3}\ b)c)))))\ \equiv\ (\lambda b.(\lambda c.(b(b(\underbrace{((\underbrace{(\lambda x.(\lambda y.(x(x(x\ y)))))}_{\mathbb{3}}\ b)\ c)))))\ \rightarrow_\beta$

x será sustituida por b

$\rightarrow_\beta\ (\lambda b.(\lambda c.(b(b\ \underbrace{((\lambda g.(b(b(b\ g))))c)})))))\ \rightarrow_\beta\ y$ será sustituida por c

$\rightarrow_\beta\ \underbrace{(\lambda b.(\lambda c.(b(b(b(b(b\ c)))))))}_{\mathbb{5}}$ Está en forma β-normal.

Figura 4.6.3. Cálculo de la forma β-normal correspondiente al término $((\textbf{SUMAR2}\ \mathbb{2})\mathbb{3})$.

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(3) Una función para ocupar el lugar del parámetro δ en las definiciones de μ y ρ. Le llamaremos φ.

El esquema correspondiente a la definición de MULT en el λ-cálculo es el siguiente:

$$\text{MULT} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.(\mu\ (\rho\ \varphi)))))$$

Dados dos números μ y ρ y una función φ, el subtérmino $(\rho\ \varphi)$ expresa lo siguiente: se aplicará ρ veces la función φ, lo cual genera ρ copias de φ. Pero delante del subtérmino $(\rho\ \varphi)$ se ha puesto μ y el subtérmino $(\mu\ (\rho\ \varphi))$ expresa lo siguiente: primero se generarán ρ copias de φ; a continuación, se generarán μ copias de ese bloque de ρ copias de φ. En total, se generarán tantas copias de φ como indica el producto de μ y ρ.

En los ejemplos concretos, los elementos del esquema general

$$\text{MULT} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.(\mu\ (\rho\ \varphi)))))$$

serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{MULT} \equiv (\lambda m.(\lambda n.(\lambda v.(m\ (n\ v)))))$$

4.6.6.2 Segunda opción: MULT2

La función MULT2 requiere cuatro datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(3) Una función para ocupar el lugar del parámetro δ en las definiciones de μ y ρ. Le llamaremos φ.

(4) Una función para ocupar el lugar del parámetro γ en las definiciones de μ y ρ. Le llamaremos ψ.

El esquema correspondiente a la definición de MULT2 en el λ-cálculo es el siguiente:

$$\text{MULT2} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.(\lambda\psi.((\mu\ (\rho\ \varphi))\psi)))))$$

Dados dos números μ y ρ y dos funciones φ y ψ, el subtérmino $(\rho\ \varphi)$ genera ρ copias de φ. El subtérmino $(\mu\ (\rho\ \varphi))$ genera μ copias del bloque formado por ρ copias de φ. En total, se generan $\mu * \rho$ copias de φ. El subtérmino $((\mu\ (\rho\ \varphi))\psi)$ aplica $\mu * \rho$ veces φ a ψ:

$$(\quad \underbrace{\varphi(\varphi(\ldots(\varphi}_{\mu * \rho \text{ copias de } \varphi} \quad (\psi))\ldots)))$$

En los ejemplos concretos, los elementos del esquema general

$$(\lambda\mu.(\lambda\rho.(\lambda\varphi.(\lambda\psi.((\mu\ (\rho\ \varphi))\psi)))))$$

serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\textsc{mult2} \equiv (\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ (n\ v))w)))))$$

4.6.6.3 Ejemplo de uso de MULT

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función $(\lambda m.(\lambda n.(\lambda v.(m\ (n\ v)))))$.

A modo de ejemplo, consideramos el siguiente término:

$$(((\lambda m.(\lambda n.(\lambda v.(m\ (n\ v)))))(\lambda h.(\lambda g.(h(h\ g)))))(\lambda x.(\lambda z.(x(x(x\ z))))))$$

Primero identificamos los componentes:

$$((\underbrace{(\lambda m.(\lambda n.(\lambda v.(m\ (n\ v)))))}_{\textsc{mult}}\ \underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{2}\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z))))))}_{3})$$

Por tanto, estamos ante el término $((\textsc{mult}\ 2)3)$ y el resultado esperado sería el término 6. En la figura 4.6.4 de la página 118 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\textsc{mult}\ 2)3)$.

4.6.6.4 Ejemplo de uso de MULT2

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función $(\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ (n\ v))$

A modo de ejemplo, consideramos el siguiente término:

$$(((\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ (n\ v))w))))(\lambda h.(\lambda g.(h(h\ g)))))(\lambda x.(\lambda z.(x(x(x\ z))))))$$

Primero identificamos los componentes:

$$((\underbrace{(\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ (n\ v))w))))}_{\textsc{mult2}}\ \underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{2}\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z))))))}_{3})$$

$$(\underbrace{(\lambda m.(\lambda n.(\lambda v.(m\ (n\ v)))))}_{\textbf{MULT}}\ \underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})\ \rightarrow_\beta$$

m será sustituida por $\mathbb{2}$

$$\rightarrow_\beta ((\lambda n.(\lambda v.(\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ (n\ v))))\ \underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})\ \rightarrow_\beta$$

n será sustituida por $\mathbb{3}$

$$\rightarrow_\beta (\lambda v.(\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\ \underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma}))\ \rightarrow_\beta$$

h será sustituida por σ (en dos sitios: σ_1 y σ_2)

$$\rightarrow_\beta (\lambda v.(\lambda g.(\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_1}\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g))))\ \rightarrow_\beta$$

x será sustituida por v en σ_1

$$\rightarrow_\beta (\lambda v.(\lambda g.(\underbrace{(\lambda z.(v(v(v\ z))))}_{\xi}\ \underbrace{(\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g)}_{\chi_1})))\ \rightarrow_\beta$$

z será sustituida por χ_1 en ξ

$$\rightarrow_\beta (\lambda v.(\lambda g.(v(v(v\ (\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g)}_{\chi_1})))))\ \rightarrow_\beta$$ x será sustituida por v en $\mathbb{3}$

$$\rightarrow_\beta (\lambda v.(\lambda g.(v(v(v(\underbrace{(\lambda z.(v(v(v\ z))))}_{\chi_2}\ g))))))\ \rightarrow_\beta$$ z será sustituida por g en χ_2

$$\rightarrow_\beta \underbrace{(\lambda v.(\lambda g.(v(v(v(v(v(v\ g)))))))))}_{\mathbb{6}}$$ Está en forma β-normal.

Figura 4.6.4. Cálculo de la forma β-normal correspondiente al término $((\textsc{mult}\ 2)3)$.

Por tanto, estamos ante el término $((\text{MULT2}\ \mathbb{2})\mathbb{3})$ y el resultado esperado sería el término $\mathbb{6}$. En la figura 4.6.5 de la página 120 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\text{MULT2}\ \mathbb{2})\mathbb{3})$.

4.6.6.5 Otra opción para definir la multiplicación: MULT3

La función MULT3 requiere dos datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

El esquema correspondiente a la definición de MULT3 en el λ-cálculo es el siguiente:

$$\text{MULT3} \equiv (\lambda\mu.(\lambda\rho.((\mu(\text{SUMAR2}\ \rho))\mathbb{0})))$$

Ideia subyacente: Se suma el número ρ μ veces al número $\mathbb{0}$.

La función SUMAR2 calcula la suma de dos números. El esquema de la definición de SUMAR2 es el siguiente:

$$\text{SUMAR2} \equiv (\lambda\mu.(\lambda\rho.((\mu\ \text{HUR})\rho)))$$

La función SUMAR2 ha sido presentada y analizada en el apartado 4.6.5.3 de la página 113.

La función SIG, dado un número, devuelve el siguiente número. El esquema correspondiente a la definición de SIG en el λ-cálculo es el siguiente:

$$\text{SIG} \equiv (\lambda\mu.(\lambda\varphi.(\lambda\psi.(\varphi((\mu\ \varphi)\psi)))))$$

La función SIG ha sido presentada y analizada en el apartado 4.6.4 de la página 109.

4.6.7 Exponenciación en el λ-cálculo: EXP y EXP2

4.6.7.1 Definición de EXP

La función EXP requiere dos datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

$$(((\underbrace{(\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ (n\ v))w))))}_{\textbf{MULT2}})\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}})\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})\ \rightarrow_\beta$$

m será sustituida por $\mathbb{2}$

$$\rightarrow_\beta ((\lambda n.(\lambda v.(\lambda w.(((\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}})\ (n\ v))w))))\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}})\ \rightarrow_\beta$$

n será sustituida por $\mathbb{3}$

$$\rightarrow_\beta (\lambda v.(\lambda w.(((\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}})\ \underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma})w))\ \rightarrow_\beta$$

h será sustituida por σ (en dos sitios: σ_1 y σ_2)

$$\rightarrow_\beta (\lambda v.(\lambda w.(\lambda g.((\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_1}(\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g)w)))\ \rightarrow_\beta$$

x será sustituida por v en σ_1

$$\rightarrow_\beta (\lambda v.(\lambda w.(\lambda g.(\underbrace{(\lambda z.(v(v(v\ z))))}_{\xi}\ \underbrace{(\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g)}_{\chi_1})w))\ \rightarrow_\beta$$

z será sustituida por χ_1 en ξ

$$\rightarrow_\beta (\lambda v.(\lambda w.(\lambda g.(v(v(v\ (\underbrace{(\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ v)}_{\sigma_2}\ g)}_{\chi_1})w))))\ \rightarrow_\beta$$

x será sustituida por v en $\mathbb{3}$

$$\rightarrow_\beta (\lambda v.(\lambda w.(\lambda g.(v(v(v(\underbrace{(\lambda z.(v(v(v\ z))))}_{\chi_2}\ g)w)))))\ \rightarrow_\beta$$

z será sustituida por g en χ_2

$$\rightarrow_\beta \underbrace{(\lambda v.(\lambda w.(\lambda g.(v(v(v(v(v\ g))))))w)))}$$

g será sustituida por w

$$\rightarrow_\beta \underbrace{(\lambda v.(\lambda w.(v(v(v(v(v\ w)))))))}_{\mathbb{6}}$$

Está en forma β-normal.

Figura 4.6.5. Cálculo de la forma β-normal correspondiente al término $((\textsc{mult2}\ \mathbb{2})\mathbb{3})$.

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

El esquema correspondiente a la definición de **EXP** en el λ-cálculo es el siguiente:

$$\mathbf{EXP} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.((\rho\ \mu)\varphi))$$

Nótese que en el subtérmino $(\rho\ \mu)$ el orden de los parámetros ha cambiado.

4.6.7.2 Ejemplos de uso de EXP

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función

$$(\lambda m.(\lambda n.(\lambda v.((n\ m)v))))$$

A modo de ejemplo, consideramos el siguiente término:

$$(((\lambda m.(\lambda n.(\lambda v.((n\ m)v))))(\lambda h.(\lambda g.(h(h\ g)))))(\lambda x.(\lambda z.(x(x(x\ z))))))$$

Primero identificamos los componentes:

$$((\underbrace{(\lambda m.(\lambda n.(\lambda v.((n\ m)v))))}_{\mathbf{EXP}}\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}\underbrace{(\lambda x.(\lambda z.(x(x(x\ z))))))}_{\mathbb{3}})$$

Por tanto, estamos ante el término $((\mathbf{BER}\ \mathbb{2})\mathbb{3})$ y el resultado esperado sería el término $\mathbb{8}$. En la figura 4.6.6 de la página 122 se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\mathbf{EXP}\ \mathbb{2})\mathbb{3})$.

4.6.7.3 EXP2: otra opción para definir la exponenciación

La función **EXP2** requiere dos datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

(2) Una expresión ρ cuyo valor sea un número. Al ser ρ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$

El esquema correspondiente a la definición de **EXP2** en el λ-cálculo es el siguiente:

$$\mathbf{EXP2} \equiv (\lambda\mu.(\lambda\rho.((\rho(\mathbf{MULT}\ \mu))\mathbb{1})))$$

Idea subyacente: Se multiplica el número $\rho\ \mu$ veces al número $\mathbb{1}$.

$\underbrace{((\underbrace{(\lambda m.(\lambda n.(\lambda v.((n\ m)v))))}_{\mathbf{EXP}}\ \mathbb{2})\ \mathbb{3})}$ $\rightarrow_\beta$ m será sustituida por $\mathbb{2}$

$\rightarrow_\beta \underbrace{((\lambda n.(\lambda v.((n\ \mathbb{2})v))))\mathbb{3})}$ $\rightarrow_\beta$ n será sustituida por $\mathbb{3}$

$\rightarrow_\beta (\lambda v.((\mathbb{3}\ \mathbb{2})v)) \equiv \underbrace{(\lambda v.((\underbrace{(\lambda x.(\lambda z.(x(x(x\ z)))))}_{\mathbb{3}}\ \mathbb{2})\ v))}$ $\rightarrow_\beta$

x será sustituida por $\mathbb{2}$

$\rightarrow_\beta (\lambda v.\underbrace{((\lambda z.(\mathbb{2}(\mathbb{2}(\mathbb{2}\ z))))v)})$ $\rightarrow_\beta$ z será sustituida por v

$\rightarrow_\beta (\lambda v.\underbrace{(\mathbb{2}(\mathbb{2}(\mathbb{2}\ v)))}) \equiv \underbrace{(\lambda v.\ (\underbrace{(\lambda h.(\lambda g.(h(h\ g))))}_{\mathbb{2}}(\mathbb{2}(\mathbb{2}\ v))))}$ $\rightarrow_\beta$

h será sustituida por $(\mathbb{2}(\mathbb{2}\ v))$

$\rightarrow_\beta (\lambda v.(\lambda g.((\mathbb{2}(\mathbb{2}\ v))((\mathbb{2}(\mathbb{2}\ v))\ g)))) \equiv$

$\equiv \underbrace{(\lambda v.(\lambda g.((\underbrace{(\lambda h'.(\lambda g'.(h'(h'\ g'))))}_{\mathbb{2}}(\mathbb{2}\ v))((\mathbb{2}(\mathbb{2}\ v))\ g))))}$ $\rightarrow_\beta$

h' será sustituida por $(\mathbb{2}\ v)$

$\rightarrow_\beta (\lambda v.(\lambda g.\underbrace{((\lambda g'.((\mathbb{2}\ v)((\mathbb{2}\ v)\ g')))((\mathbb{2}(\mathbb{2}\ v))\ g))))}$ $\rightarrow_\beta$ g' será sustituida por $((\mathbb{2}(\mathbb{2}\ v))\ g)$

$\rightarrow_\beta (\lambda v.(\lambda g.((\mathbb{2}\ v)((\mathbb{2}\ v)\ ((\mathbb{2}(\mathbb{2}\ v))\ g))))) \equiv$

$\equiv \underbrace{(\lambda v.(\lambda g.\ (((\underbrace{(\lambda h''.(\lambda g''.(h''(h''\ g''))))}_{\mathbb{2}}\ v)((\mathbb{2}\ v)\ ((\mathbb{2}(\mathbb{2}\ v))\ g)))))}$

h'' será sustituida por v

Continua en la figura 4.6.7 de la página 123....

Figura 4.6.6. Primera parte del cálculo de la forma β-normal correspondiente al término $((\mathbf{EXP}\ \mathbb{2})\mathbb{3})$.

Continuación de la figura 4.6.6 de la página 122....

$$\rightarrow_\beta \ (\lambda v.(\lambda g.\underbrace{((\lambda g''.(v(v\ g'')))((\mathbb{2}\ v)\ ((\mathbb{2}(\mathbb{2}\ v))\ g)))}))\ \rightarrow_\beta$$

g'' será sustituida por $((\mathbb{2}\ v)\ ((\mathbb{2}(\mathbb{2}\ v))\ g))$

$$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v\ ((\mathbb{2}\ v)\ ((\mathbb{2}(\mathbb{2}\ v))\ g))))))\ \equiv$$

$$\equiv \ (\lambda v.(\lambda g.(v(v\ \underbrace{(\underbrace{(\lambda h_3.(\lambda g_3.(h_3(h_3\ g_3))))}_{\mathbb{2}}\ v)}\ ((\mathbb{2}(\mathbb{2}\ v))\ g))))))\ \rightarrow_\beta$$

h_3 será sustituida por v

$$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v\ \underbrace{((\lambda g_3.(v(v\ g_3)))\ ((\mathbb{2}(\mathbb{2}\ v))\ g))}))))\ \rightarrow_\beta$$

g_3 será sustituida por $((\mathbb{2}(\mathbb{2}\ v))\ g)$

$$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v(\underbrace{(v(v((\mathbb{2}(\mathbb{2}\ v))\ g)))}))))))\ \equiv$$

$$\equiv \ (\lambda v.(\lambda g.(v(v((v(v(\underbrace{(\lambda h_4.(\lambda g_4.(h_4(h_4\ g_4))))}_{\mathbb{2}}(\mathbb{2}\ v))\ g)))))))\ \rightarrow_\beta$$

h_2 será sustituida por $(\mathbb{2}\ v)$

$$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v\underbrace{((v(v(\underbrace{(\lambda g_4.((\mathbb{2}\ v)((\mathbb{2}\ v)\ g_4)))}\ g)))))))}))\ \rightarrow_\beta$$

g_4 será sustituida por g

$$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v((v(v((\mathbb{2}\ v)((\mathbb{2}\ v)g)))))))))\ \equiv$$

$$\equiv \ (\lambda v.(\lambda g.(v(v((v(v(\underbrace{(\underbrace{(\lambda h_5.(\lambda g_5.(h_5(h_5\ g_5))))}_{\mathbb{2}}\ v)}((\mathbb{2}\ v)g)))))))))\ \rightarrow_\beta$$

h_5 será sustituida por v

Continua en la figura 4.6.8 de la página 124....

Figura 4.6.7. Segunda parte del cálculo de la forma β-normal correspondiente al término $((\mathbf{EXP}\ \mathbb{2})\mathbb{3})$.

Continuación de la figura 4.6.7 de la página 123....

$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v(v(v\underbrace{((\lambda g_5.(v(v\ g_5)))((\mathbb{2}\ v)g))}))))))\ \rightarrow_\beta$

g_5 será sustituida por $((\mathbb{2}\ v)g)$

$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v(v(v(v(v((\mathbb{2}\ v)g)))))))))\ \equiv$

$\equiv \ (\lambda v.(\lambda g.(v(v(v(v(v(v\underbrace{(\underbrace{(\lambda h_6.(\lambda g_6.(h_6(h_6\ g_6))))}_{\mathbb{2}}\ v)}\ g)))))))))\ \rightarrow_\beta$

h_6 será sustituida por v

$\rightarrow_\beta \ (\lambda v.(\lambda g.(v(v(v(v(v\underbrace{((\lambda g_6.(v(v\ g_6))))g)}))))))))\ \rightarrow_\beta$

g_6 será sustituida por g

$\rightarrow_\beta \ \underbrace{(\lambda v.(\lambda g.(v(v(v(v(v(v(v\ g)))))))))}_{\mathbb{8}}$

Está en forma β-normal.

Figura 4.6.8. Tercera parte del cálculo de la forma β-normal correspondiente al término $((\textbf{EXP}\ \mathbb{2})\mathbb{3})$.

La función MULT calcula el producto de dos números. El esquema de la definición de MULT es el siguiente:

$$\text{MULT} \equiv (\lambda\mu.(\lambda\rho.(\lambda\varphi.(\mu\ (\rho\ \varphi)))))$$

La función MULT ha sido presentada y analizada en el apartado 4.6.6 de la página 113.

4.6.8 El predecesor de un número en λ-cálculo: PRED

4.6.8.1 Definición de PRED

La función PRED requiere dos datos:

(1) Una expresión μ cuyo valor sea un número. Al ser μ un número, tendrá el formato que se muestra a continuación: $\lambda(\delta.(\lambda\gamma.(\delta(\delta(\ldots(\delta(\gamma))\ldots)))))$. Si utilizamos variables concretas, podemos expresarlo de la siguiente forma:

$$(\lambda y.(\lambda z.(y(y(\ldots(y\ z)\ldots)))))$$

(2) Una función para ocupar el lugar del parámetro δ en la definición de μ. Le llamaremos φ.

(3) Una función para ocupar el lugar del parámetro γ en la definición de μ. Le llamaremos ψ.

El esquema correspondiente a la definición de PRED en el λ-cálculo es el siguiente:

$$\text{PRED} \equiv (\lambda\mu.(\lambda\varphi.(\lambda\psi.\ (((\mu\ \underbrace{(\lambda\xi.(\lambda\sigma.(\sigma(\xi\ \varphi))))}_{\chi_1})\ \underbrace{(\lambda\rho.\psi)}_{\chi_2})\ \underbrace{(\lambda\pi.\pi)}_{\chi_3})))))$$

En los ejemplos concretos, los elementos del esquema general serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{PRED} \equiv (\lambda n.(\lambda f.(\lambda x.\ (((n\ \underbrace{(\lambda g.(\lambda h.(h(g\ f))))}_{\chi_1})\ \underbrace{(\lambda v.x)}_{\chi_2})\ \underbrace{(\lambda w.w)}_{\chi_3})))))$$

Dados un número n y dos funciones f y x, hay que definir un término que sirva para aplicar la función f $n-1$ veces a x. Dicho de otra forma, hay que generar un término que represente el número $n-1$. Por tanto, hay que pasar de un término que aplica f n veces a x, a un término que aplica f $n-1$ veces a x. Es decir, hay que eliminar una de las n aplicaciones de f a x. Se utilizarán tres funciones auxiliares: χ_1, χ_2 y χ_3.

4.6.8.2 Ejemplo de uso de PRED

En este apartado se analiza un ejemplo que sirve para ilustrar el uso de la función

$$\textsc{pred} \equiv (\lambda n.(\lambda f.(\lambda x.\ (\underbrace{\underbrace{(n\ \chi_1)}\ \chi_2)\ \chi_3}))))$$

En concreto, aplicaremos la función PRED al número $\mathbb{3}$. Por tanto, se calculará la forma β-normal del término $(\textsc{pred}\ \mathbb{3})$. El resultado que se espera obtener es $\mathbb{2}$. El proceso de cálculo de la forma β-normal correspondiente al término $(\textsc{pred}\ \mathbb{3})$ esta formalizado paso a paso en la figura 4.6.9 de la página 127, en la figura 4.6.10 de la página 128 y en la figura 4.6.11 de la página 129.

Cada χ_1 genera una copia de f y la coloca en el lado derecho para que luego la utilice el χ_1 de su derecha. Cada χ_1 coge la copia de f generada por el χ_1 de su izquierda y la utiliza para dar un paso en la construcción del nuevo número. En total se generan n copias de χ_1 y cada uno de estos términos χ_1 genera una f. Consecuentemente, se generan n copias de f. Se elminará la copia de f generada por el término χ_1 del extremo derecho. Se esa manera, quedarán $n-1$ copias de f.

χ_2 es la función que se utiliza para eliminar una copia de f. Se elimina la última f que se genera. La función χ_3 es la función identidad y no produce ningún cambio, devuelve como resultado lo que se le da como dato. Cada χ_1 necesita la f generada por el χ_1 que estaba a su izquierda, pero para el primer χ_1 no hay ninguna f generada previamente, y es por ello que se pone χ_3, para que el primer χ_1 tenga una función. Pero conviene que esa función no haga nada (es como un 0 en la suma, un 1 en el producto, etc.).

4.6.9 La resta en el λ-cálculo: RESTAR

4.6.9.1 Definición de RESTAR

La función RESTAR requiere dos datos: dos términos μ y ρ cuyo valor sea un número.

El esquema correspondiente a la definición de RESTAR en el λ-cálculo es el siguiente:

$$\textsc{restar} \equiv (\lambda\mu.(\lambda\rho.((\mu\ \textsc{pred})\rho)))$$

En los ejemplos concretos, los elementos del esquema general serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\textsc{restar} \equiv (\lambda m.(\lambda n.((n\ \textsc{pred})m)))$$

Dados nos números m y n, se le aplicará n veces la función PRED al número m.

$$(\textbf{PRED}\ \mathbb{3}) \equiv ((\lambda n.(\lambda f.(\lambda x.(\underbrace{\underbrace{(\underbrace{(n\ \chi_1)}\ \chi_2)}\ \chi_3)}))))\mathbb{3}) \rightarrow_\beta$$

n será sustituida por $\mathbb{3}$

$$\rightarrow_\beta (\lambda f.(\lambda x.(\underbrace{\underbrace{(\underbrace{(\mathbb{3}\ \chi_1)}\ \chi_2)}\ \chi_3}))) \equiv$$

$$\equiv (\lambda f.(\lambda x.(\underbrace{\underbrace{\underbrace{(\underbrace{(\lambda y.(\lambda z.(y(y(y\ z)))))}_{\mathbb{3}}\ \chi_1)\ \chi_2)}\ \chi_3}})) \rightarrow_\beta$$

y será sustituida por χ_1

$$\rightarrow_\beta (\lambda f.(\lambda x.(\underbrace{\underbrace{(\underbrace{(\lambda z.(\chi_1(\chi_1(\chi_1\ z))))}\ \chi_2)}\ \chi_3}))) \rightarrow_\beta$$

z será sustituida por χ_2

$$\rightarrow_\beta (\lambda f.(\lambda x.(\underbrace{\underbrace{(\chi_1(\chi_1(\chi_1\ \chi_2)))}\ \chi_3}))) \equiv$$

$$\equiv (\lambda f.(\lambda x.(\underbrace{\underbrace{(\underbrace{(\lambda g.(\lambda h.(h(g\ f))))}_{\chi_1}(\chi_1(\chi_1\ \chi_2)))\ \chi_3}}))) \rightarrow_\beta$$

g será sustituida por $(\chi_1(\chi_1\ \chi_2))$

$$\rightarrow_\beta (\lambda f.(\lambda x.(\underbrace{(\lambda h.(h((\chi_1(\chi_1\ \chi_2))\ f)))\chi_3)})) \rightarrow_\beta$$

h será sustituida por χ_3

Continua en la figura 4.6.10 de la página 128....

Figura 4.6.9. Primera parte del cálculo de la forma β-normal correspondiente al término $(\textbf{PRED}\ \mathbb{3})$.

Continuación de la figura 4.6.9 de la página 127....

$\rightarrow_\beta (\lambda f.(\lambda x.(\chi_3((\chi_1(\chi_1\ \chi_2))\ f)))) \equiv$

$\equiv (\lambda f.(\lambda x.\underbrace{(\underbrace{(\lambda w.w)}_{\chi_3}((\chi_1(\chi_1\ \chi_2))\ f))))}) \rightarrow_\beta$

w será sustituida por $((\chi_1(\chi_1\ \chi_2))\ f)$

$\rightarrow_\beta (\lambda f.(\lambda x.((\chi_1(\chi_1\ \chi_2))\ f))) \equiv$

$\equiv (\lambda f.(\lambda x.\underbrace{(\underbrace{(\underbrace{(\lambda g.(\lambda h.(h(g\ f))))}_{\chi_1}(\chi_1\ \chi_2))}\ f)}) \rightarrow_\beta$

g será sustituida por $(\chi_1\ \chi_2)$

$\rightarrow_\beta (\lambda f.(\lambda x.\underbrace{(\underbrace{(\lambda h.(h((\chi_1\ \chi_2)\ f)))\ f)}))}) \rightarrow_\beta$

h será sustituida por f

$\rightarrow_\beta (\lambda f.(\lambda x.\underbrace{(f((\chi_1\ \chi_2)\ f))}))) \equiv$

$\equiv (\lambda f.(\lambda x.(f(\underbrace{(\underbrace{(\lambda g.(\lambda h.(h(g\ f))))}_{\chi_1}\ \chi_2)}\ f)))) \rightarrow_\beta$

g será sustituida por χ_2

Continua en la figura 4.6.11 de la página 129....

Figura 4.6.10. Segunda parte del cálculo de la forma β-normal correspondiente al término (PRED $\mathbb{3}$).

Continuación de la figura 4.6.10 de la página 128....

$\rightarrow_\beta (\lambda f.(\lambda x.(f \underbrace{((\lambda h.(h(\chi_2\ f)))\ f)}))) \rightarrow_\beta$

h será sustituida por f

$\rightarrow_\beta (\lambda f.(\lambda x.(f(f(\chi_2\ f))))) \equiv$

$\equiv (\lambda f.(\lambda x.(f(f\ \underbrace{\underbrace{(\lambda v.x)}_{\chi_2}\ f})))) \rightarrow_\beta$

v será sustituida por f

$\rightarrow_\beta \underbrace{(\lambda f.(\lambda x.(f(f\ x))))}_{\mathbb{2}}$

Está en forma β-normal.

Figura 4.6.11. Tercera parte del cálculo de la forma β-normal correspondiente al término (PRED $\mathbb{3}$).

4.6.10 Decidir si un número es cero en el λ-cálculo: ES_CERO

4.6.10.1 Definición de ES_CERO

La función ES_CERO requiere como dato un valor numérico μ y decidirá si ese valor numérico es cero. El resultado será TRUE o FALSE.

El esquema correspondiente a la definición de ES_CERO en el λ-cálculo es el siguiente:

$$\text{ES\_CERO} \equiv (\lambda\mu.((\mu(\lambda\psi.\text{FALSE}))\text{TRUE}))$$

En los ejemplos concretos, los elementos del esquema general serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{ES\_CERO} \equiv (\lambda n.((n(\lambda v.\text{FALSE}))\text{TRUE}))$$

Dado un número n, se le aplicará n veces n la función $(\lambda v.\text{FALSE})$ al valor TRUE. De esa manera, si n es cero, se aplicará cero veces la función $(\lambda v.\text{FALSE})$ al valor TRUE y, por consiguiente, el resultado será el propio TRUE. En cambio, si n es mayor que cero, la función $(\lambda v.\text{FALSE})$ será aplicada al menos una vez y, puesto que esa función siempre devuelve FALSE, el resultado final será FALSE.

4.6.11 La comparación 'menor o igual' en el λ-cálculo: MIG

4.6.11.1 Definición de MIG

La función MIG requiere como datos, dos valores numéricos μ y ρ. Si μ es menor que ρ o μ es igual a ρ, entonces devolverá TRUE y, si no, devolverá FALSE.

El esquema correspondiente a la definición de MIG en el λ-cálculo es el siguiente:

$$\text{MIG} \equiv (\lambda\mu.(\lambda\rho.(\text{ES\_CERO}((\text{RESTAR } \mu)\rho))))$$

En los ejemplos concretos, los elementos del esquema general serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{MIG} \equiv (\lambda m.(\lambda n.(\text{ES\_CERO}((\text{RESTAR } m)n))))$$

Dados dos números m y n, se calculará $m - n$ y, si el resultado es cero, entonces sabremos que m es menor o igual que n. Conviene recordar que si n es mayor que m, la resta devolverá el valor cero porque estamos manejando solo números naturales y, consecuentemente, los números negativos no se pueden utilizar.

4.6.12 La igualdad entre números en el λ-cálculo: IGUAL

4.6.12.1 Definición de IGUAL

La función IGUAL requiere como datos, dos valores numéricos μ y ρ. Si μ y ρ son α-equivalentes, entonces devolverá TRUE y, si no, devolverá FALSE.

El esquema correspondiente a la definición de IGUAL en el λ-cálculo es el siguiente:

$$\text{IGUAL} \equiv (\lambda\mu.(\lambda\rho.((\text{AND}((\text{MIG}\ \mu)\rho))((\text{MIG}\ \rho)\mu))))$$

En los ejemplos concretos, los elementos del esquema general serán reemplazados por las variables que nos interesen. Por ejemplo, de esta manera:

$$\text{IGUAL} \equiv (\lambda m.(\lambda n.((\text{AND}((\text{MIG}\ m)n))((\text{MIG}\ n)m))))$$

Dados dos números m y n, si m es menor o igual que n y, además, n es menor o igual que m, entonces m y n son iguales y la función IGUAL devolverá TRUE. En cambio, si o bien m no es menor o igual que n o bien n no es menor o igual que m, entonces m y n no son iguales y la función IGUAL devolverá FALSE.

4.7.

Recursividad en el λ-cálculo

4.7.1 La recursividad directa no es posible en el λ-cálculo

Para poder utilizar la recursividad de manera directa, las funciones han de tener un nombre. En el λ-cálculo, las funciones no tienen nombre, son anónimas. Recordemos la funciÂº'on que calcula la suma de dos números naturales:

$$\textsc{sumar} \equiv (\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))$$

Ahí, SUMAR no es el nombre de la función. SUMAR es una abreviatura que hemos introducido nosotros para no tener que escribir toda la función en los ejemplos. Pero eso es algo externo al propio λ-cálculo. además, aunque utilicemos abreviaturas para simplificar la escritura de las expresiones y para mejorar su legibilidad, en el momento en el que tengamos que calcular la forma β-normal de una expresión que contenga abreviaturas, antes o después habra que sustituir las abreviaturas por sus verdaderas definiciones. Por ejemplo, en el caso de SUMAR, habrá que poner lo siguiente:

$$(\lambda m.(\lambda n.(\lambda v.(\lambda w.((m\ v)((n\ v)w))))))$$

Por tanto, cuando haya que utilizar la función y haya que operar con ella, será necesario escribir la definición completa.

En los lenguajes de programación de alto nivel, las funciones suelen tener un nombre y para utilizar una función, es suficiente con poner su nombre —y los argumentos requeridos.

En la tabla 4.7.1 de la página 135, se muestra una función escrita en Haskell. Dicha función calcula la suma de los números naturales que van del 0 al x, siendo x mayor o igual que 0.

$$\sum_{k=0}^{x} k$$

La función diseñada calcula ese sumatorio utilizando recursividad. El planteamiento recursivo elegido es el siguiente:

$$\sum_{k=0}^{x} k = x + \sum_{k=0}^{x-1} k$$

En la tabla 4.7.1 de la página 135, tenemos una opción para implementar ese planteamiento recursivo, pero no es la única posibilidad. Por un lado tenemos la función *sumatorio*. Esa función no es propiamente recursiva, puesto que llama a la función *sumatorio_aux*, la cual se encarga de realizar realmente el cálculo del sumatorio. La función *sumatorio_aux* sí es recursiva. El cometido de la función *sumatorio* es calcular la suma de los números naturales que van del 0 al x, pero la función *sumatorio_aux*, aparte del parámetro x, tiene otros dos parámetros: el parámetro i se utilizará para ir pasando los números desde x hasta 0; el parámetro r se utilizará para ir guardando el resultado parcial calculado hasta el momento. Cuando termine la recursividad, el resultado definitivo estará en r. El cometido de la función *sumatorio_aux* es el de sumar a r la suma de los números que van desde i hasta 0. La precondición indica que en r se tiene la suma de los números que van desde x hasta $x + 1$. Por tanto, la función *sumatorio_aux* devolverá en r la suma de los números naturales que van del 0 al x.

En la tabla 4.7.2 de la página 135, se muestra otra función escrita en Haskell. Dicha función calcula el factorial de x, es decir, el producto de los números que van del 1 a x.

$$\prod_{k=1}^{x} k$$

La función diseñada calcula el factorial utilizando recursividad. El planteamiento recursivo elegido es el siguiente:

$$\prod_{k=1}^{x} k = 1 * \sum_{k=2}^{x} k$$

En la tabla 4.7.2 de la página 135, tenemos una opción para implementar ese planteamiento recursivo, pero no es la única posibilidad. Por un lado tenemos la función *factorial*. Esa función no es propiamente recursiva, puesto que llama a la función *factorial_aux*, la cual se encarga de realizar realmente el cálculo del factorial. La función *factorial_aux* sí es recursiva. El cometido de la función *factorial* es calcularel producto de los números naturales que van de 1 a x, pero la función *factorial_aux*, aparte del parámetro x, tiene otros dos parámetros: el parámetro i se utiliza para ir pasando los números desde 1 hasta $x + 1$; el parámetro r se utiliza para ir guardando el resultado parcial calculado hasta el momento. Cuando termine la recursividad, el resultado definitivo estará en r. El cometido de la función *factorial_aux* es el de multiplicar a r el producto de los números que van desde i hasta x. La precondición indica que en r se tiene el producto de los números que van desde 1 hasta $i - 1$. Por tanto, la función *factorial_aux* devolverá en r el producto de los números naturales que van de 1 a x.

Se considera que se cumplen las condiciones establecidas mediante la especificación. Consecuentemente, no hay casos de error.

```
sumatorio :: Integer -> Integer

-- Precondición: x ∈ ℕ

sumatorio x = sumatorio_aux x x 0

-------------------------------------------------------------

sumatorio_aux :: Integer -> Integer -> Integer -> Integer
-- Precondición: x ∈ ℕ ∧ i ∈ ℕ ∧ (0 ≤ i ≤ x) ∧ r = Σ_{k=i+1}^{x} k
sumatorio_aux x i r
  | i == 0        = r
  | otherwise     = sumatorio_aux x (i-1) (r+i)
```

Tabla 4.7.1. Función que calcula la suma de los números naturales que van del 0 al x. Está escrita en Haskell y utiliza recursividad directa.

Se considera que se cumplen las condiciones establecidas mediante la especificación. Consecuentemente, no hay casos de error.

```
factorial :: Integer -> Integer

-- Precondición: x ∈ ℕ

factorial x = factorial_aux x 1 1

-------------------------------------------------------------

factorial_aux :: Integer -> Integer -> Integer -> Integer
-- Precondición: x ∈ ℕ ∧ i ∈ ℕ ∧ (1 ≤ i ≤ x + 1) ∧ r = Π_{k=1}^{i-1} k
factorial_aux x i r
  | i == x+1      = r
  | otherwise     = factorial_aux x (i+1) (r*i)
```

Tabla 4.7.2. Función que calcula el factorial de un número natural x. Está escrita en Haskell y utiliza recursividad directa.

4.7.2 Funciones recursivas en el λ-cálculo: la función Y

En el λ-cálculo, se utiliza la función Y para definir las funciones recursivas. A la función Y se le ha de dar como dato de entrada, una función R —es decir, un λ-término. Al aplicar β-reducción al término $(\mathsf{Y}\ R)$, se obtendrá el término $(R(\mathsf{Y}\ R))$.

La definición de la función Y es la siguiente:

$$\mathsf{Y} \equiv (\lambda h.\ \underbrace{(\underbrace{(\lambda g.(h(gg)))}\ \underbrace{(\lambda g.(h(gg)))}})$$

En la figura 4.7.1 de la página 137, se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(\mathsf{Y}\ R)$. La secuencia generada en esa figura es la siguiente:

$$(\mathsf{Y}\ R)\ \rightarrow_\beta (R(\mathsf{Y}\ R))\ \rightarrow_\beta (R(R(\mathsf{Y}\ R)))\ \rightarrow_\beta (R(R(R(\mathsf{Y}\ R))))\ \rightarrow_\beta \ldots$$

Por tanto, el término R será aplicado una y otra vez y eso es al fin y al cabo la recursividad: aplicar una misma función una y otra vez.

4.7.3 Suma de los números del intervalo $[0..x]$, utilizando la función Y

4.7.3.1 La función *sumatorio_aux*, utilizando la función Y

En la tabla 4.7.1 de la página 135, se muestran las funciones *sumatorio* y *sumatorio_aux*, escritas en Haskell. En este apartado se indicará cómo hay que formular la función *sumatorio_aux* en el λ-cálculo.

Para formular la función *sumatorio_aux*, hay que formular el término R adecuado para este ejemplo:

$$R \equiv (\lambda v.(\lambda x.(\lambda i.(\lambda r.(((\textbf{ES\_CERO}\ i)r)(((v\ x)(\textbf{PRED}\ i))((\textbf{SUMAR}\ r)i)))))))$$

Conviene diferenciar con claridad los componentes de ese término:

$$R \equiv (\lambda v.(\lambda x.(\lambda i.(\lambda r.\ \underbrace{(\underbrace{(\underbrace{(\textbf{ES\_CERO}\ i)}\ \underbrace{r}_{})}\ \underbrace{(((v\ x)(\textbf{PRED}\ i))((\textbf{SUMAR}\ r)i))}})))))$$

El significado de ese término es el siguiente:

- Si el valor de i es 0, entonces el resultado es r,
- En caso contrario, es decir, si el valor de i no es 0, entonces el resultado se obtendrá pasándole a la función v los datos x, $i-1$ y $r+i$.

$$(\mathsf{Y}\ R) \equiv (\underbrace{(\lambda h.((\lambda g.(h(gg)))(\lambda g.(h(gg)))))}_{\mathsf{Y}}\ R) \rightarrow_\beta$$

h será sustituida por R

$$\rightarrow_\beta \underbrace{(\underbrace{(\lambda g.(R(gg)))}_{\delta}\ \underbrace{(\lambda g.(R(gg)))}_{\delta})}_{(\mathsf{Y}\ R)} \rightarrow_\beta$$

en la δ de la izquierda, g será sustituida por la δ de la derecha

$$\rightarrow_\beta \underbrace{(R(\underbrace{(\lambda g.(R(gg)))}_{\delta}\ \underbrace{(\lambda g.(R(gg)))}_{\delta}))}_{(R(\mathsf{Y}\ R))} \rightarrow_\beta$$

en la δ de la izquierda, g será sustituida por la δ de la derecha

$$\rightarrow_\beta \underbrace{(R(R(\underbrace{(\lambda g.(R(gg)))}_{\delta}\ \underbrace{(\lambda g.(R(gg)))}_{\delta})))}_{(R(R(\mathsf{Y}\ R)))} \rightarrow_\beta$$

en la δ de la izquierda, g será sustituida por la δ de la derecha

$$\rightarrow_\beta \underbrace{(R(R(R(\underbrace{(\lambda g.(R(gg)))}_{\delta}\ \underbrace{(\lambda g.(R(gg)))}_{\delta}))))}_{(R(R(R(\mathsf{Y}\ R))))} \rightarrow_\beta$$

$\ldots$

Que el proceso termine o no termine dependerá del término R.

Cuando en el lugar de R se ponga un término concreto, se sabrá

si el proceso será finito o no.

Figura 4.7.1. Cálculo de la forma β-normal correspondiente al término $(\mathsf{Y}\ R)(\mathsf{Y}\ R)$.

En la definición de R, todavía no se puede ver qué función va a ocupar el sitio de v, pero al dar la definición de *sumatorio_aux* se verá que v es el término $(\mathsf{Y}\ R)$. Y ese término es el que permite simular la recursividad en el λ-cálculo.

A continuación, se muestra la formulación de la función *sumatorio_aux*:

$$(\lambda x.(\lambda i.(\lambda r.((((\mathsf{Y}\ R)x)i)r))))$$

A modo de ejemplo, para calcular la suma de los números del intervalo $[0..3]$, hay que calcular la forma β-normal del siguiente término:

$$((((\lambda x.(\lambda i.(\lambda r.((((\mathsf{Y}\ R)x)i)r))))\mathbb{3})\mathbb{3})\mathbb{0})$$

Puesto de otra forma, hay que calcular la forma β-normal del término $((((\mathsf{Y}\ R)\mathbb{3})\mathbb{3})\mathbb{0})$. Y eso es lo que se hará en este apartado.

El primer paso de β-reducción es el siguiente:

$$((((\mathsf{Y}\ R)\mathbb{3})\mathbb{3})\mathbb{0}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{3})\mathbb{3})\mathbb{0})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{3}$, en vez de i hay que poner $\mathbb{3}$ y en vez de r hay que poner $\mathbb{0}$:

$$(((\textsc{es\_cero}\ \mathbb{3})\mathbb{0})((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{3}))((\textsc{sumar}\ \mathbb{0})\mathbb{3})))$$

Puesto que el valor del término $(\textsc{zero\_da}\ \mathbb{3})$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{3}))((\textsc{sumar}\ \mathbb{0})\mathbb{3}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \mathbb{3})\mathbb{2})\mathbb{3})$:

$$((((\mathsf{Y}\ R)\mathbb{3})\mathbb{2})\mathbb{3}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{3})\mathbb{2})\mathbb{3})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{3}$, en vez de i hay que poner $\mathbb{2}$ y en vez de r hay que poner $\mathbb{3}$:

$$(((\textsc{es\_cero}\ \mathbb{2})\mathbb{3})((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{2}))((\textsc{sumar}\ \mathbb{3})\mathbb{2})))$$

Puesto que el valor del término $(\textsc{zero\_da}\ \mathbb{2})$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{2}))((\textsc{sumar}\ \mathbb{3})\mathbb{2}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \mathbb{3})\mathbb{1})\mathbb{5})$:

$$((((\mathsf{Y}\ R)\mathbb{3})\mathbb{1})\mathbb{5}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{3})\mathbb{1})\mathbb{5})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{3}$, en vez de i hay que poner $\mathbb{1}$ y en vez de r hay que poner $\mathbb{5}$:

$$(((\textsc{es\_cero}\ \mathbb{1})\mathbb{5})((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{1}))((\textsc{sumar}\ \mathbb{5})\mathbb{1})))$$

Puesto que el valor del término $(\textsc{es\_cero}\ \mathbb{1})$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{1}))((\textsc{sumar}\ \mathbb{5})\mathbb{1}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \mathbb{3})\mathbb{0})\mathbb{6})$:

$$((((\mathsf{Y}\ R)\mathbb{3})\mathbb{0})\mathbb{6}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{3})\mathbb{0})\mathbb{6})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{3}$, en vez de i hay que poner $\mathbb{0}$ y en vez de r hay que poner $\mathbb{6}$:

$$(((\textsc{es\_cero}\ \mathbb{0})\mathbb{6})((((\mathsf{Y}\ R)\ \mathbb{3})(\textsc{pred}\ \mathbb{0}))((\textsc{sumar}\ \mathbb{6})\mathbb{0})))$$

Puesto que el valor del término $(\textsc{es\_cero}\ \mathbb{0})$ es $\textsc{true}$, el término $\mathbb{6}$ es la forma β-normal.

4.7.3.2 La función *sumatorio*

En este apartado se indicará cómo hay que formular la función *sumatorio* en el λ-cálculo.

El término correspondiente a la función *sumatorio* en el λ-cálculo es el siguiente:

$$(\lambda x.((((\mathsf{Y}\ R)x)x)\mathbb{0}))$$

Para calcular la suma de los números del intervalo $[0..3]$, hay que calcular la forma β-normal del siguiente término:

$$((\lambda x.((((\mathsf{Y}\ R)x)x)\mathbb{0}))\mathbb{3})$$

Es decir, hay que calcular la forma β-normal del siguiente término:

$$((((\mathsf{Y}\ R)\mathbb{3})\mathbb{3})\mathbb{0})$$

Y eso es lo que se ha hecho en el apartado anterior.

4.7.4 Factorial de x, utilizando la función Y

4.7.4.1 La función *factorial_aux*, utilizando la función Y

En la tabla 4.7.2 de la página 135, se muestran las funciones *factorial* y *factorial_aux*, escritas en Haskell. En este apartado se indicará cómo hay que formular la función *factorial_aux* en el λ-cálculo.

Para formular la función *factorial_aux*, hay que formular el término R adecuado para este ejemplo:

$$R \equiv (\lambda v.(\lambda x.(\lambda i.(\lambda r.((((\textsc{igual}\ i)(\textsc{sig}\ x))r)(((v\ x)(\textsc{sig}\ i))((\textsc{mult}\ r)i)))))))$$

Conviene diferenciar con claridad los componentes de ese término:

$$R \equiv (\lambda v.(\lambda x.(\lambda i.(\lambda r.\ (\underbrace{(\underbrace{(\underbrace{(\underbrace{(\textsc{igual}\ i)}(\textsc{sig}\ x)}) r}) \underbrace{(((v\ x)(\textsc{sig}\ i))((\textsc{mult}\ r)i))}})))))$$

El significado de ese término es el siguiente:

- Si el valor de i es igual al valor de $x + 1$, entonces el resultado es r,
- En caso contrario, es decir, si el valor de i no es igual al valor de $x + 1$, entonces el resultado se obtendrá pasándole a la función v los datos x, $i + 1$ y $r * i$.

En la definición de R, todavía no se puede ver qué función va a ocupar el sitio de v, pero al dar la definición de *factorial_aux* se verá que v es el término $(\mathsf{Y}\ R)$. Y ese término es el que permite simular la recursividad en el λ-cálculo.

A continuación, se muestra la formulación de la función *factorial_aux*:

$$(\lambda x.(\lambda i.(\lambda r.((((\mathsf{Y}\ R)x)i)r))))$$

A modo de ejemplo, para calcular el factorial del número 4, hay que calcular la forma β-normal del siguiente término:

$$((((\lambda x.(\lambda i.(\lambda r.((((\mathsf{Y}\ R)x)i)r))))\mathbb{4})\mathbb{1})\mathbb{1})$$

Puesto de otra forma, hay que calcular la forma β-normal del término $((((\mathsf{Y}\ R)\mathbb{4})\mathbb{1})\mathbb{1})$. Y eso es lo que se hará en este apartado.

El primer paso de β-reducción es el siguiente:

$$((((\mathsf{Y}\ R)\mathbb{4})\mathbb{1})\mathbb{1}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{4})\mathbb{1})\mathbb{1})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{4}$, en vez de i hay que poner $\mathbb{1}$ y en vez de r hay que poner $\mathbb{1}$:

$$((((\textsc{igual}\ \mathbb{1})(\textsc{sig}\ \mathbb{4}))\mathbb{1})((((\mathsf{Y}\ R)\ \mathbb{4})(\textsc{sig}\ \mathbb{1}))((\textsc{mult}\ \mathbb{1})\mathbb{1})))$$

Puesto que el valor del término $((\textsc{igual}\ \mathbb{1})(\textsc{sig}\ \mathbb{4}))$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \mathbb{4})(\textsc{hur}\ \mathbb{1}))((\textsc{bider}\ \mathbb{1})\mathbb{1}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \mathbb{4})\mathbb{2})\mathbb{1})$:

$$((((\mathsf{Y}\ R)\ \mathbb{4})\mathbb{2})\mathbb{1}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\mathbb{4})\mathbb{2})\mathbb{1})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\mathbb{4}$, en vez de i hay que poner $\mathbb{2}$ y en vez de r hay que poner $\mathbb{1}$:

$$((((\textsc{igual}\ \mathbb{2})(\textsc{sig}\ \mathbb{4}))\mathbb{1})((((\mathsf{Y}\ R)\ \mathbb{4})(\textsc{sig}\ \mathbb{2}))((\textsc{mult}\ \mathbb{1})\mathbb{2})))$$

Puesto que el valor del término $((\textsc{igual}\ \underline{2})(\textsc{sig}\ \underline{4}))$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \underline{4})(\textsc{sig}\ \underline{2}))((\textsc{mult}\ \underline{1})\underline{2}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \underline{4})\underline{3})\underline{2})$:

$$((((\mathsf{Y}\ R)\underline{4})\underline{3})\underline{2}) \rightarrow_\beta ((((R(\mathsf{Y}\ R))\underline{4})\underline{3})\underline{2})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\underline{4}$, en vez de i hay que poner $\underline{3}$ y en vez de r hay que poner $\underline{2}$:

$$((((\textsc{igual}\ \underline{3})(\textsc{sig}\ \underline{4}))\underline{2})((((\mathsf{Y}\ R)\ \underline{4})(\textsc{sig}\ \underline{3}))((\textsc{mult}\ \underline{2})\underline{3})))$$

Puesto que el valor del término $((\textsc{igual}\ \underline{3})(\textsc{sig}\ \underline{4}))$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \underline{4})(\textsc{sig}\ \underline{3}))((\textsc{mult}\ \underline{2})\underline{3}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \underline{4})\underline{4})\underline{6})$:

$$((((\mathsf{Y}\ R)\underline{4})\underline{4})\underline{6}) \rightarrow_\beta ((((R(\mathsf{Y}\ R))\underline{4})\underline{4})\underline{6})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\underline{4}$, en vez de i hay que poner $\underline{4}$ y en vez de r hay que poner $\underline{6}$:

$$((((\textsc{igual}\ \underline{4})(\textsc{sig}\ \underline{4}))\underline{6})((((\mathsf{Y}\ R)\ \underline{4})(\textsc{sig}\ \underline{4}))((\textsc{mult}\ \underline{6})\underline{4})))$$

Puesto que el valor del término $((\textsc{igual}\ \underline{4})(\underline{5}))$ es **FALSE**, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \underline{4})(\textsc{sig}\ \underline{4}))((\textsc{mult}\ \underline{6})\underline{4}))$. Es decir, el término $((((\mathsf{Y}\ R)\ \underline{4})\underline{5})\underline{24})$:

$$((((\mathsf{Y}\ R)\underline{4})\underline{5})\underline{24}) \rightarrow_\beta ((((R(\mathsf{Y}\ R))\underline{4})\underline{5})\underline{24})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de x hay que poner $\underline{4}$, en vez de i hay que poner $\underline{5}$ y en vez de r hay que poner $\underline{24}$:

$$((((\textsc{igual}\ \underline{5})(\textsc{sig}\ \underline{4}))\underline{24})((((\mathsf{Y}\ R)\ \underline{4})(\textsc{hur}\ \underline{5}))((\textsc{mult}\ \underline{24})\underline{5})))$$

Puesto que el valor del término $((\textsc{igual}\ \underline{5})(\textsc{sig}\ \underline{4}))$ es **TRUE**, el término $\underline{24}$ es la forma β-normal.

4.7.4.2 La función *factorial*

En este apartado se indicará cómo hay que formular la función *factorial* en el λ-cálculo.

El término correspondiente a la función *factorial* en el λ-cálculo es el siguiente:

$$(\lambda x.((((\mathsf{Y}\ R)x)\underline{1})\underline{1}))$$

Para calcular el factorial de $\underline{4}$, hay que calcular la forma β-normal del siguiente término:

$$((\lambda x.((((\mathsf{Y}\ R)x)\underline{1})\underline{1}))\underline{4})$$

Es decir, hay que calcular la forma β-normal del siguiente término:

$$((((\mathsf{Y}\ R)\underline{4})\underline{1})\underline{1})$$

Y eso es lo que se ha hecho en el apartado anterior.

4.8.

Tuplas en el λ-cálculo

4.8.1 Pares ordenados en el λ-cálculo: PAR

En matemáticas, dados dos elementos a y b, el par ordenado formado por esos dos elementos se representa como (a, b). Por tanto, para generar un par ordenado formado por dos elementos a y b, se utilizan paréntesis y coma. El hecho de que se diga que el par es ordenado significa que (a, b) y (b, a) son distintos. Es decir, el orden de los componentes es relevante.

Es habitual definir operaciones sobre los pares ordenados. Las más importantes son la operación $primero$ que devuelve el primer componente y la operación $segundo$ que devuelve el segundo componente. Consecuentemente, $primero(a, b) = a$ y $segundo(a, b) = b$.

En el λ-cálculo, un par ordenado se genera mediante el siguiente término:

$$\textbf{PAR} \equiv (\lambda\gamma.(\lambda\delta.(\lambda\rho.((\rho\ \gamma)\delta))))$$

Si utilizamos las variables h, g y f tendremos lo siguiente:

$$\textbf{PAR} \equiv (\lambda h.(\lambda g.(\lambda f.((f\ h)g))))$$

Por ejemplo, para generar el par ordenado $(5, 2)$, hay que escribir el siguiente término:

$$(((\lambda h.(\lambda g.(\lambda f.((f\ h)g))))\mathbb{5})\mathbb{2})$$

Si utilizamos la abreviatura, nos queda $((\textbf{PAR}\ \mathbb{5})\mathbb{2})$. A continuación se muestra el paso de β-reducción que lleva a la forma β-normal:

$$(((\lambda h.(\lambda g.(\lambda f.((f\ h)g))))\mathbb{5})\mathbb{2})\ \rightarrow_\beta (\lambda f.((f\ \mathbb{5})\mathbb{2}))$$

Por tanto, el par ordenado que en matemáticas se representa como $(5, 2)$, en el λ-cálculo se representa como $(\lambda f.((f\ \mathbb{5})\mathbb{2}))$.

Dado un par ordenado, para obtener el primer componente de ese par hay que poner TRUE en lugar de f. Es decir, hay que aplicar el término $(\lambda f.((f\ \mathbb{5})\mathbb{2}))$ al dato TRUE:

$$((\lambda f.((f\ \mathbb{5})\mathbb{2}))\textsc{true})\ \rightarrow_\beta\ ((\textsc{true}\ \mathbb{5})\mathbb{2})\ \rightarrow_\beta\ \mathbb{5}$$

En cambio, para obtener el segundo componente de ese par hay que poner FALSE en lugar de f. Es decir, hay que aplicar el término $(\lambda f.((f\ \mathbb{5})\mathbb{2}))$ al dato FALSE:

$$((\lambda f.((f\ \mathbb{5})\mathbb{2}))\textsc{false})\ \rightarrow_\beta\ ((\textsc{false}\ \mathbb{5})\mathbb{2})\ \rightarrow_\beta\ \mathbb{2}$$

Esto significa que la operación $primero$ de matemáticas, es implementada mediante la función TRUE en el λ-cálculo, y la operación $segundo$ de matemáticas, es implementada mediante la función FALSE en el λ-cálculo. Este ejemplo nos muestra y aclara que el parámetro f sirve para poder acceder a los componentes del par ordenado.

Por ejemplo, consideramos la función que, dado un par ordenado formado por dos números, devuelve el par ordenado que se obtiene incrementando en uno el valor de los componentes del par ordenado de entrada. Su definición en el λ-cálculo es el siguiente:

$$\textsc{incr\_par} \equiv (\lambda p.((\textsc{par}\ (\textsc{sig}(\textsc{true}\ b)))(\textsc{sig}(\textsc{false}\ b))))$$

A continuación se indica el significado de ese término: dado un par ordenado p, se va a generar otro par ordenado cuyo primer componente es el primer componente de p más 1 (el siguiente) y cuyo segundo componente es el segundo componente de p más 1 (el siguiente).

Si a la función INCR_PAR le damos el par ordenado $(\lambda f.((f\ \mathbb{5})\mathbb{2}))$ —es decir, el par ordenado $((\textsc{par}\ \mathbb{5})\mathbb{2})$— se obtendrá el par ordenado $((\textsc{par}\ \mathbb{6})\mathbb{3})$:

$$(\textsc{incr\_par}((\textsc{par}\ \mathbb{5})\mathbb{2}))\ \rightarrow_\beta \ldots\ \rightarrow_\beta\ ((\textsc{par}\ \mathbb{6})\mathbb{3})$$

Los componentes de un par ordenado pueden ser de distintos tipos. Por ejemplo, podemos tener un par ordenado formado por 8 y $True$. En matemáticas escribiríamos $(8, True)$ y en el λ-cálculo escribiríamos $((\textsc{par}\ \mathbb{8})\textsc{true})$, que viene a ser el término $(\lambda f.((f\ \mathbb{8})\textsc{true}))$. De la misma forma, el par ordenado que en matemáticas se representa como $(True, True)$, en el λ-cálculo se representa como $((\textsc{par}\ \textsc{true})\textsc{true})$ o como $(\lambda f.((f\ \textsc{true})\textsc{true}))$.

4.9.

Listas en el λ-cálculo

4.9.1 Estructura recursiva de las listas

Los pares ordenados siempre constarán de dos componentes. El concepto de lists puede ser entendido como una generalización del concepto de par ordenado: en una lista, al igual que en un par ordenado, el orden de los componentes es significativo; las listas se generan a base de repetir la estructura de par ordenado. Una lista es un par donde el segundo componente es otro par cuyo segundo componente es otro par, etc.

En matemáticas una lista de n componentes puede ser expresada como $[a_1, a_2, a_3, \ldots, a_n]$. Pero realmente, esa lista representa la siguiente estructura recursiva:

$$(a_1, (a_2, (a_3, (\ldots (a_n, ()))))))$$

Por tanto, toda la lista ha de ser entendida como un par ordenado. El primer componente es a_1 y el segundo componente es $(a_2, (a_3, (\ldots (a_n, ()))))$. De la misma forma, la sublista $(a_2, (a_3, (\ldots (a_n, ()))))$ es otro par ordenado cuyo primer componente es a_2 y cuyo segundo componente es $(a_3, (\ldots (a_n, ())))$. La estructura se repite hasta llegar al último par ordenado: $(a_n, ())$. En ese último par ordenado, el primer componente es a_n y el segundo componente es $()$ da. Mediante $()$ se representa la lista vacía:

$$\underbrace{(a_1, \underbrace{(a_2, \underbrace{(a_3, (\ldots \underbrace{(a_n, ())}_{}))}_{})}_{})}_{}$$

La lista vacía es una tupla de cero componentes. Una lista $[a_1]$ de un único componente, es un par ordenado que tiene la siguiente estructura:

$$(\underbrace{a_1}, \underbrace{()})$$

Una lista $[a_1, a_2]$ de dos componentes, es un par ordenado que tiene la siguiente estructura:

$$(\underbrace{a_1}, \underbrace{(a_2, ())})$$

Una lista $[a_1, a_2, a_3]$ de tres componentes, es un par ordenado que tiene la siguiente estructura:

$$(\underbrace{a_1}, \underbrace{(a_2, (a_3, ()))})$$

Cualquier lista que no sea vacía, puede ser entendida como un par ordenado que internamente tiene una estructura recursiva.

Consecuentemente, a la hora de definir listas, vamos areutilizar algunos conceptos relacionados con los pares ordenados. Pero habrá que tener en cuenta que en el caso de las listas, hay que incorporar la noción de tupla vacía o lista vacía.

4.9.2 Lista vacía: VACIA

En el λ-cálculo, la lista vacía será una función. He aquí el término correspondiente a la lista vacía:

$$\textsc{vacia} \equiv (\lambda f.\textsc{true})$$

Recordemos que un par ordenado (a, b) se representa como un término de la forma $((\textsc{par}\ a)b)$, es decir, tendrá la estructura $(\lambda f.((f\ a)b))$. Por tanto, las listas no vacías y la lista vacía tendrán el mismo formato: $(\lambda f.(\dots))$.

4.9.3 Ejemplos de listas

La lista $[5, 9, 8]$ se representa de la siguiente forma en el λ-cálculo:

$$((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$$

La estructura es la siguiente:

$$\underbrace{((\textsc{par}\ 5)\ \underbrace{((\textsc{par}\ 9)\ \underbrace{((\textsc{par}\ 8)\textsc{vacia})}))}}$$

En el λ-cálculo, los componentes de una lista pueden ser de distintos tipos:

$$((\textsc{par}\ 5)((\textsc{par}\ \textsc{true})((\textsc{par}\ 8)\textsc{vacia})))$$

Ese término representa la lista $[5, True, 8]$.

Para representar la lista vacía [ ], tenemos el término VACIA.

4.9.4 Operaciones sobre listas

4.9.4.1 Operación que decide si una lista es vacía: ES_VACIA

La operación que, dada una lista devuelve TRUE si la lista es vacía y devuelve FALSE si la lista no es vacía, se define mediante el siguiente término:

$$\textsc{es\_vacia} \equiv (\lambda z.(z(\lambda h.(\lambda g.\textsc{false}))))$$

En la figura 4.9.1 de la página 149, se muestra el proceso de cálculo de la forma β-normal correspondiente al término (ES_VACIA VACIA). Dicho proceso ha de terminar decidiendo si la lista VACIA es vacía.

Por otra parte, en la figura 4.9.2 de la página 150, se muestra el proceso de cálculo de la forma β-normal correspondiente al término $((\textsc{par}\ \mathbb{5})((\textsc{par}\ \mathbb{9})((\textsc{par}\ \mathbb{8})\textsc{vacia})))$. Por tanto, podemos observar qué ocurre si a la función ES_VACIA le damos como dato una lista no vacía.

4.9.4.2 Primer elemento y resto de la lista: TRUE y FALSE

Puesto que las listas no vacías son pares ordenados, para obtener el elemento del extremo izquierdo se utilizará la función TRUE, mientras que para obtener la sublista que queda tras eliminar el elemento del extremo izquierdo —el resto— se utilizará la función FALSE.

Cuando aplicamos cualquier operación a la lista vacía, el resultado siempre será TRUE. Cuando nos den una lista, lo mejor es averiguar si la lista es vacía. Para ello se ha de utilizar la función ES_VACIA. Y si la lista es vacía, entonces en la mayoría de los casos no tendrá sentido aplicar ninguna operación.

En la figura 4.9.3 de la página 151, se muestra el proceso que se ha de seguir para obtener el primer elemento de la lista $[5, 9, 8]$. Es decir, se muestra el proceso de cálculo de la forma β-normal correspondiente al término $(((\textsc{par}\ \mathbb{5})((\textsc{par}\ \mathbb{9})((\textsc{par}\ \mathbb{8})\textsc{vacia})))\textsc{true})$.

En la figura 4.9.4 de la página 152 y en la figura 4.9.5 de la página 153, se muestra el proceso que se ha de seguir para obtener el resto de la lista $[5, 9, 8]$. Es decir, se muestra el proceso de cálculo de la forma β-normal correspondiente al término

$$(((\textsc{par}\ \mathbb{5})((\textsc{par}\ \mathbb{9})((\textsc{par}\ \mathbb{8})\textsc{vacia})))\textsc{false})$$

4.9.4.3 Suma de los elementos de una lista numérica: SUMA_LISTA

En este apartado se va a definir la función SUMA_LISTA que, dada una lista numérica, devuelve $\mathbb{0}$ si la lista es vacía y devuelve la suma de los componentes de la lista si la lista no es vacía. La

definición de dicha función es recursiva y, por tanto, requiere el uso de la función Y.

En la tabla 4.9.1 de la página 149, se muestran las funciones *suma_lista* y *suma_lista_aux*, escritas en Haskell. La función *suma_lista_aux* tiene tres parámetros: la lista de partida s que nunca cambia; la lista z que va perdiendo elementos por la izquierda; y el parámetro t que sirve para guardar la suma de los elementos que z va perdiendo por la izquierda. Tal como se puede apreciar en la definición de *suma_lista*, al principio s y z son iguales. Por otro lado, indicar que la función $longitud$, dada una lista, devuelve el número de elementos de la lista; la función $elem\_pos$, dadas una lista y una posición de la lista, devuelve el elemento que ocupa esa posición en la lista; y la función $sufijo$, dadas dos listas, decide si la primera lista es una sublista de la segunda lista, pero situada justo en la parte final o derecha.

A continuación, se indica cómo se han de formular las funciones *suma_lista* y *suma_lista_aux* en el λ-cálculo.

4.9.4.3.1 La función *suma_lista_aux* utilizando la función Y

Para definir la función *suma_lista_aux*, hay que formular el término R adecuado:

$$R \equiv (\lambda v.(\lambda s.(\lambda z.(\lambda t.(((\textbf{ES\_VACIA}\ z)t)(((v\ s)(\textbf{FALSE}\ z))((\textbf{SUMAR}\ t)(\textbf{TRUE}\ z))))))))$$

Conviene identificar con claridad los componentes de ese término:

$$R \equiv (\lambda v.(\lambda s.(\lambda z.(\lambda t.\ (\underbrace{(\underbrace{(\underbrace{\textbf{ES\_VACIA}\ z}_{}\ \underbrace{t}_{})}_{}\ \underbrace{(((v\ s)(\textbf{FALSE}\ z))((\textbf{SUMAR}\ t)(\textbf{TRUE}\ z)))}_{})}_{}))))))$$

El significado del término R formulado, es el siguiente:

- Si la lista z es vacía, el resultado es t,
- Si la lista z no es vacía, para obtener el resultado habrá que darle a la función v, como datos de entrada, la lista s, el resto de la lista z y la suma de t y el primer elemento de z.

Conviene recordar que la función TRUE es la que sirve para obtener el primer elemento de una lista y que la función FALSE es la que sirve para obtener el resto de una lista. Por otro lado, en la definición de R, todavía no se puede ver qué función va a ocupar el sitio de v, pero al dar la definición de *suma_lista_aux* se verá que v es el término $(\mathsf{Y}\ R)$. Y ese término es el que permite simular la recursividad en el λ-cálculo.

A continuación, se muestra la formulación de la función *suma_lista_aux*:

$$(\lambda s.(\lambda z.(\lambda t.((((\mathsf{Y}\ R)s)z)t))))$$

$$(\textbf{ES\_VACIA VACIA}) \equiv (\underbrace{(\lambda z.(z(\lambda h.(\lambda g.\textbf{FALSE}))))}_{\textbf{ES\_VACIA}}\ \textbf{VACIA}) \rightarrow_\beta$$

z será sustituida por **VACIA**

$$\rightarrow_\beta (\textbf{VACIA}(\lambda h.(\lambda g.\textbf{FALSE}))) \equiv (\underbrace{(\lambda f.\textbf{TRUE})}_{\textbf{VACIA}}(\lambda h.(\lambda g.\textbf{FALSE}))) \rightarrow_\beta$$

f será sustituida por $(\lambda h.(\lambda g.\textbf{FALSE}))$

$$\rightarrow_\beta \textbf{TRUE}$$

Está en forma β-normal.

Figura 4.9.1. Cálculo de la forma β-normal correspondiente al término (**ES_VACIA VACIA**).

Se considera que se cumplen las condiciones establecidas mediante la especificación. Consecuentemente, no hay casos de error.

```
suma_lista :: [Integer] -> Integer

-- Precondición: ∀k((1 ≤ k ≤ longitud(s)) → (elem_pos(s, k) ∈ ℕ))

suma_lista s = suma_lista_aux s s 0
```

```
suma_lista_aux :: [Integer] -> [Integer] -> Integer -> Integer
-- Precondición:
∀k((1 ≤ k ≤ (longitud s)) → (elem_pos s k) ∈ ℕ))∧
∧(sufijo z s) ∧ Σ_{k=1}^{longitud s} (elem_pos s k) = t + Σ_{k=1}^{longitud z} (elem_pos z k)
suma_lista_aux s z t
  | (es_vacia z)   = t
  | otherwise      = suma_lista_aux s (resto z) (t + (primero z))
```

Tabla 4.9.1. Cálculo de la suma de los elementos de una lista. Funciones escritas en Haskell y utilizando recursividad.

$(\textbf{ES\_VACIA}\ ((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))\ \equiv$

$\equiv\ (\underbrace{(\lambda z.(z(\lambda h.(\lambda g.\textbf{FALSE}))))}_{\textbf{ES\_VACIA}}((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))\ \rightarrow_\beta$

z será sustituida por $((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))$

$\rightarrow_\beta\ (((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))(\lambda h.(\lambda g.\textbf{FALSE})))\ \equiv$

$\equiv\ ((((\underbrace{(\lambda j.(\lambda k.(\lambda f.((f\ j)k))))}_{\textbf{PAR}})\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))(\lambda h.(\lambda g.\textbf{FALSE})))\ \rightarrow_\beta$

j será sustituida por $\mathbb{5}$

$\rightarrow_\beta\ ((\underbrace{(\lambda k.(\lambda f.((f\ \mathbb{5})k)))}\ \underbrace{((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))})(\lambda h.(\lambda g.\textbf{FALSE})))\ \rightarrow_\beta$

k será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$\rightarrow_\beta\ (\underbrace{(\lambda f.((f\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))}\ \underbrace{(\lambda h.(\lambda g.\textbf{FALSE}))})\ \rightarrow_\beta$

f será sustituida por $(\lambda h.(\lambda g.\textbf{FALSE}))$

$\rightarrow_\beta\ ((\underbrace{(\lambda h.(\lambda g.\textbf{FALSE}))}\ \underbrace{\mathbb{5}})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\ \rightarrow_\beta$

h será sustituida por $\mathbb{5}$

$\rightarrow_\beta\ (\underbrace{(\lambda g.\textbf{FALSE})}\ \underbrace{((\textbf{BIK}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))})\ \rightarrow_\beta$

g será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$\rightarrow_\beta\ \textbf{FALSE}$

Está en forma β-normal.

Figura 4.9.2. Cálculo de la forma β-normal correspondiente al término $(\textbf{ES\_VACIA}\ ((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))$.

$$(((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{TRUE}) \equiv$$

$$\equiv ((((\underbrace{(\lambda j.(\lambda k.(\lambda f.((f\ j)k))))}_{\textbf{PAR}}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{TRUE}) \rightarrow_\beta$$

j será sustituida por $\mathbb{5}$

$$\rightarrow_\beta (((\underbrace{(\lambda k.(\lambda f.((f\ \mathbb{5})k)))}\ \underbrace{((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))}))\textbf{TRUE}) \rightarrow_\beta$$

k será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$$\rightarrow_\beta (\underbrace{(\lambda f.((f\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))}\ \textbf{TRUE}) \rightarrow_\beta$$

f será sustituida por $\textbf{TRUE}$

$$\rightarrow_\beta ((\textbf{TRUE}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))) \equiv$$

$$\equiv ((\underbrace{(\lambda v.(\lambda w.v))}_{\textbf{TRUE}}\ \underbrace{\mathbb{5}}\ )((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))) \rightarrow_\beta$$

v será sustituida por $\mathbb{5}$

$$\rightarrow_\beta (\underbrace{(\lambda w.\mathbb{5})}\ \underbrace{((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))}) \rightarrow_\beta$$

w será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$$\rightarrow_\beta \mathbb{5}$$

Está en forma β-normal.

Figura 4.9.3. El primero de la lista: cálculo de la forma β-normal correspondiente al término $(((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{TRUE})$.

$$(((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{FALSE}) \equiv$$

$$\equiv ((((\underbrace{(\lambda j.(\lambda k.(\lambda f.((f\ j)k))))}_{\textbf{PAR}}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{FALSE}) \rightarrow_\beta$$

j será sustituida por $\mathbb{5}$

$$\rightarrow_\beta (((\underbrace{(\lambda k.(\lambda f.((f\ \mathbb{5})k)))}\ \underbrace{((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))})\textbf{FALSE}) \rightarrow_\beta$$

k será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$$\rightarrow_\beta (\underbrace{(\lambda f.((f\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))))}\ \textbf{FALSE}) \rightarrow_\beta$$

f será sustituida por **FALSE**

$$\rightarrow_\beta ((\textbf{FALSE}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))) \equiv$$

$$\equiv ((\underbrace{(\lambda v.(\lambda w.w))}_{\textbf{FALSE}}\ \underbrace{\mathbb{5}}\ )((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))) \rightarrow_\beta$$

v será sustituida por $\mathbb{5}$

$$\rightarrow_\beta (\underbrace{(\lambda w.w)}\ \underbrace{((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))}) \rightarrow_\beta$$

w será sustituida por $((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA}))$

$$\rightarrow_\beta ((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})) \equiv$$

$$\equiv ((\underbrace{(\lambda r.(\lambda s.(\lambda t.((t\ r)s))))}_{\textbf{PAR}}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})) \rightarrow_\beta$$

r será sustituida por $\mathbb{9}$

Continua en la figura 4.9.5 de la página 153....

Figura 4.9.4. Resto de la lista: Primera parte del cálculo de la forma β-normal correspondiente al término $(((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{FALSE})$.

Continuación de la figura 4.9.4 de la página 152....

$$\rightarrow_\beta \underbrace{((\lambda s.(\lambda t.((t\ \mathbb{9})s)))}\underbrace{((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})}\ \rightarrow_\beta$$

s será sustituida por $((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})$

$$\rightarrow_\beta (\lambda t.((t\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\ \equiv$$

$$\equiv\ (\lambda t.((t\ \mathbb{9})((\underbrace{(\lambda c.(\lambda d.(\lambda e.((e\ c)d))))}_{\textbf{PAR}}\ \underbrace{\mathbb{8}}\ )\textbf{VACIA})))\ \rightarrow_\beta$$

c será sustituida por $\mathbb{8}$

$$\rightarrow_\beta (\lambda t.((t\ \mathbb{9})(\underbrace{(\lambda d.(\lambda e.((e\ \mathbb{8})d)))}\ \textbf{VACIA})))\ \rightarrow_\beta$$

d será sustituida por $\textbf{VACIA}$

$$\rightarrow_\beta (\lambda t.((t\ \mathbb{9})\ \underbrace{(\lambda e.((e\ \mathbb{8})\textbf{VACIA}))}))$$

Está en forma β-normal.

Figura 4.9.5. Resto de la lista: Segunda parte del cálculo de la forma β-normal correspondiente al término $(((\textbf{PAR}\ \mathbb{5})((\textbf{PAR}\ \mathbb{9})((\textbf{PAR}\ \mathbb{8})\textbf{VACIA})))\textbf{FALSE})$.

A modo de ejemplo, se va a calcular la suma de los elementos de la lista $[5, 9, 8]$. El λ-término correspondiente a esa lista es el siguiente:

$$((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$$

Para calcular la suma de los elementos de esa lista, hay que calcular la forma β-normal del siguiente término:

$$((((\lambda s.(\lambda z.(\lambda t.((((\mathsf{Y}\ R)s)z)t))))\delta_0)\delta_0)\mathbb{0})$$

donde δ_0 es el término $((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$.

Por tanto, hay que calcular la forma β-normal del término $((((\mathsf{Y}\ R)\delta_0)\delta_0)\mathbb{0})$ y eso es lo que se va realizar a continuación.

El primer paso de β-reducción es el siguiente:

$$((((\mathsf{Y}\ R)\delta_0)\delta_0)\mathbb{0}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\delta_0)\delta_0)\mathbb{0})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de s hay que poner δ_0, en vez de z hay que poner δ_0 y en vez de t hay que poner $\mathbb{0}$:

$$(((\textsc{es\_vacia};\delta_0)\mathbb{0})((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_0))((\textsc{sumar}\ \mathbb{0})(\textsc{true}\ \delta_0))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_0)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_0))((\textsc{sumar}\ \mathbb{0})(\textsc{true}\ \delta_0)))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_1)\mathbb{5})$, donde δ_1 es la lista $((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia}))$:

$$((((\mathsf{Y}\ R)\delta_0)\delta_1)\mathbb{5})) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\delta_0)\delta_1)\mathbb{5})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de s hay que poner δ_0, en vez de z hay que poner δ_1 y en vez de t hay que poner $\mathbb{5}$:

$$(((\textsc{es\_vacia}\ \delta_1)\mathbb{5})((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_1))((\textsc{sumar}\ \mathbb{5})(\textsc{true}\ \delta_1))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_1)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_1))((\textsc{sumar}\ \mathbb{5})(\textsc{true}\ \delta_1)))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_2)\mathbb{14})$, donde δ_2 es la lista $((\textsc{par}\ 8)\textsc{vacia})$:

$$((((\mathsf{Y}\ R)\ \delta_0)\delta_2)\mathbb{14}) \ \rightarrow_\beta \ ((((R(\mathsf{Y}\ R))\ \delta_0)\delta_2)\mathbb{14})$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de s hay que poner δ_0, en vez de z hay que poner δ_2 y en vez de t hay que poner $\mathbb{14}$:

$$(((\textsc{es\_vacia}\ \delta_2)\mathbb{14})((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_2))((\textsc{sumar}\ \mathbb{14})(\textsc{true}\ \delta_2))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_2)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_2))((\textsc{sumar}\ \mathbb{14})(\textsc{true}\ \delta_2)))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_3)\mathbb{22})$, donde δ_3 es la lista $\textsc{vacia}$:

$$((((\mathsf{Y}\ R)\ \delta_0)\delta_3)22)\ \rightarrow_\beta\ ((((R(\mathsf{Y}\ R))\ \delta_0)\delta_3)22)$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de s hay que poner δ_0, en vez de z hay que poner δ_3 y en vez de t hay que poner 22:

$$(((\textsc{es\_vacia}\ \delta_3)22)((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_3))((\textsc{sumar}\ 22)(\textsc{true}\ \delta_3))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_3)$ es TRUE, el término 22 es la forma β-normal.

4.9.4.3.2 La función *suma_lista*

En este apartado se indicará cómo hay que formular la función *suma_lista* en el λ-cálculo.

El término correspondiente a la función *suma_lista* en el λ-cálculo es el siguiente:

$$(\lambda s.((((\mathsf{Y}\ R)s)s)\mathbb{0}))$$

Para calcular la suma de los elementos de la lista $[5, 9, 8]$, hay que calcular la forma β-normal del siguiente término:

$$((\lambda s.((((\mathsf{Y}\ R)s)s)\mathbb{0}))\delta_0)$$

donde δ_0 es el término $((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$

Es decir, hay que calcular la forma β-normal del siguiente término:

$$((((\mathsf{Y}\ R)\delta_0)\delta_0)\mathbb{0})$$

Y eso es lo que se ha hecho en el apartado anterior.

4.10.

La función universal en el λ-cálculo

En este apartado se presenta el concepto de función universal y se indica la manera de representar ese concepto en el λ-cálculo.

4.10.1 El concepto de función universal

Se dice que una función es universal si tras recibir, como datos de entrada, una función h y un dato de entrada d para h, calcula el resultado que devolvería la función h para el dato d. Por tanto, una función universal es capaz de simular el comportamiento de la función h con dato de entrada d.

La versión más simple de función universal es la función $\mathcal{U}$ que tras recibir, como datos de entrada, una función h y un dato de entrada d para h, devuelve el resultado que devolvería la función h para el dato d:

$$\mathcal{U}(h, d) = h(d)$$

$\mathcal{U}$ realiza dicho cálculo para cualquier función h y cualquier dato d.

El concepto de función universal es la versión matemática del ordenador. Cuando a un ordenador se le da un programa π y un dato δ para ese programa, el ordenador nos devuelve el resultado $\pi(\delta)$.

Como casos concretos de π y δ, nos sirven cualquier programa y cualquier dato que pueda ser válido como entrada para dicho programa. He aquí algunos ejemplos concretos:

- El compilador de Haskell es π y un programa escrito por nosotros en Haskell es δ. El ordenador —la función universal— ejecuta el compilador pasándole, a modo de dato de entrada, el programa escrito por nosotros. El ordenador es capaz de simular la tarea del compilador que le hemos pasado.
- Un programa P que hemos escrito nosotros y que sirve para calcular el factorial es π. Cualquier número natural z que elijamos es δ. El ordenador —la función universal—

ejecuta el programa P con dato de entrada z. El ordenador es capaz de simular el cálculo correspondiente a P.

- El programa J correspondiente a un juego es π y cualquier dato d que pueda necesitar ese juego es δ. El ordenador —la función universal— ejecuta el programa J con dato de entrada d. El ordenador es capaz de simular el comportamiento correspondiente a J.

En el mundo físico, la función universal se ha materializado mediante el ordenador. En el plano teórico, cada modelo de computación tiene su manera de formular el concepto de función universal. En el siguiente apartado, se indica cómo formular el concepto de función universal en el λ-cálculo.

4.10.2 λ-término universal: la función universal en el λ-cálculo

4.10.2.1 U, U_2, U_3, U_4, ...

En el λ-cálculo, la función universal se formaliza mediante un λ-término. En concreto, mediante una abstracción. Esa abstracción recibe el nombre de λ-término universal:

$$\mathsf{U} \equiv (\lambda h.(\lambda d.(h\ d)))$$

El término U es una función que tiene dos parámetros de entrada h y d, y devuelve el término $(h\ d)$. De esa manera se obtiene el resultado correspondiente a la aplicación de la función h al dato d.

A modo de ejemplo, podemos considerar el término $((\mathsf{U}\ \textsc{sig})\mathbb{7})$:

$$((\mathsf{U}\ \textsc{sig})\mathbb{7}) \equiv (((\lambda h.(\lambda d.(h\ d)))\ \textsc{sig})\mathbb{7}) \rightarrow_\beta ((\lambda d.(\textsc{sig}\ d))\mathbb{7}) \rightarrow_\beta (\textsc{sig}\ \mathbb{7}) \rightarrow_\beta \mathbb{8}$$

En el λ-cálculo, el propio proceso de cálculo de la forma β-normal desempeñ el papel de función universal y, consecuentemente, en vez de escribir $((\mathsf{U}\ \textsc{sig})\mathbb{7})$, se puede escribir directamente $(\textsc{sig}\ \mathbb{7})$. El resultado obtenido será el mismo en ambos casos:

$$(\textsc{sig}\ \mathbb{7}) \rightarrow_\beta \mathbb{8}$$

Por tanto, no es estrictamente necesario utilizar U, pero si se desea utilizar, se puede utilizar. El proceso de cálculo de la forma β-normal es una manera implícita de formalizar la función universal, mientras que el término U es una manera explícita de formalizar la función universal.

El concepto de función universal puede ser adaptado a casos en los que se trabaje con una función h que requiere más de un dato de entrada:

$$\mathsf{U}_2 \equiv (\lambda h.(\lambda d_1.(\lambda d_2.((h\ d_1)d_2))))$$

El término U_2 formaliza la función universal $\mathcal{U}_2(h, d_1, d_2) = h(d_1, d_2)$.

De la misma forma, se pueden definir U_3, U_4 etc.

4.10.2.2 $\mathsf{U}_{\mathbb{N}}$ y $\mathsf{U}'_{\mathbb{N}}$

Es posible abarcar todos los casos mediante un único λ-término: $\mathsf{U}_{\mathbb{N}}$

El λ-término universal $\mathsf{U}_{\mathbb{N}}$ tendrá dos parámetros: una función h y una lista s formada por todos los argumentos que necesite la función h. Podemos considerar que s será el λ-término correspondiente a una lista de la forma $[d_1, d_2, \ldots, d_k]$. La función $\mathsf{U}_{\mathbb{N}}$ construirá el término $((((h\ d_1)d_2)\ldots)d_k)$ a partir de h y s, para luego calcular la forma β-normal mediante β-reducciones.

La definición de la función universal $\mathsf{U}_{\mathbb{N}}$ es recursiva y, consecuentemente, se necesita una función auxiliar a la que denominaremos $\mathsf{U}'_{\mathbb{N}}$.

En primer lugar definiremos la función auxiliar $\mathsf{U}'_{\mathbb{N}}$. Para ello, hay que formular el término R adecuado:

$$R \equiv (\lambda v.(\lambda h.(\lambda z.(\lambda t.(((\textsc{es\_vacia}\ z)t)(((v\ h)(\textsc{false}\ z))(t(\textsc{true}\ z))))))))$$

Conviene identificar con claridad los componentes de ese término:

$$R \equiv (\lambda v.(\lambda h.(\lambda z.(\lambda t.\underbrace{(\underbrace{(\underbrace{(\textsc{es\_vacia}\ z)}\ \underbrace{t}\ )}\ \underbrace{(((v\ h)(\textsc{false}\ z))(t(\textsc{true}\ z)))}))))))$$

El significado del término R formulado, es el siguiente:

- Si la lista z es vacía, el resultado es t,
- Si la lista z no es vacía, para obtener el resultado habrá que darle a la función v, como datos de entrada, la función h, el resto de la lista z y el término que surge al plantear la aplicación de t al primer elemento de z.

Conviene recordar que la función TRUE es la que sirve para obtener el primer elemento de una lista y que la función FALSE es la que sirve para obtener el resto de una lista. Por otro lado, en la definición de R, todavía no se puede ver qué función va a ocupar el sitio de v, pero al dar la definición de $\mathsf{U}'_{\mathbb{N}}$ se verá que v es el término $(\mathsf{Y}\ R)$. Ese término es el que permite simular la recursividad en el λ-cálculo. Es importante mencionar que el parámetro h no se utiliza para nada pero se ha puesto porque h es el término de partida. De todas formas, se podría quitar.

A continuación, se muestra la formulación de la función $\mathsf{U}'_{\mathbb{N}}$:

$$\mathsf{U}'_{\mathbb{N}} \equiv (\lambda h.(\lambda z.(\lambda t.((((\mathsf{Y}\ R)h)z)t))))$$

A modo de ejemplo, si consideramos la función $(\lambda u.(\lambda v.(\lambda w.(\textsc{sumar}((\textsc{sumar}\ u)v)w))))$ que sirve para calcular la suma de tres números u, v y w y consideramos la lista $[5, 9, 8]$ —es decir, el término $((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$— que contiene los tres números concretos que queremos sumar, la función universal $\mathsf{U}'_{\mathbb{N}}$ generará el siguiente término:

$$((((\lambda u.(\lambda v.(\lambda w.(\textsc{sumar}((\textsc{sumar}\ u)v)w))))\mathbb{5})\mathbb{9})\mathbb{8})$$

Y, después, mediante una serie de β-reducciones se llegará al término $\mathbb{22}$.

El punto de partida es el término

$$(((\mathsf{U}'_{\mathbb{N}}\ \delta_0)\delta_1)\delta_0)$$

donde δ_0 es el término

$$(\lambda u.(\lambda v.(\lambda w.(\textsc{sumar}((\textsc{sumar}\ u)v)w))))$$

y donde δ_1 es el término

$$((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$$

Por tanto, hay que calcular la forma β-normal del siguiente término:

$$((((\lambda s.(\lambda z.(\lambda t.((((\mathsf{Y}\ R)s)z)t))))\delta_0)\delta_1)\delta_0)$$

Es decir, hay que calcular la forma β-normal del siguiente término:

$$((((\mathsf{Y}\ R)\delta_0)\delta_1)\delta_0)$$

Y eso es lo que se va realizar a continuación.

El primer paso de β-reducción es el siguiente:

$$((((\mathsf{Y}\ R)\delta_0)\delta_1)\delta_0)\ \rightarrow_\beta\ ((((R(\mathsf{Y}\ R))\delta_0)\delta_1)\delta_0)$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de h hay que poner δ_0, en vez de z hay que poner δ_1 y en vez de t hay que poner δ_0:

$$(((\textsc{es\_vacia}\ \delta_1)\delta_0)((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_1))(\delta_0(\textsc{true}\ \delta_1))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_1)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_1))(\delta_0(\textsc{true}\ \delta_1)))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_2)(\delta_0\ \mathbb{5}))$, donde δ_2 es la lista $((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia}))$:

$$((((\mathsf{Y}\ R)\delta_0)\delta_2)(\delta_0\ \mathbb{5}))\ \rightarrow_\beta\ ((((R(\mathsf{Y}\ R))\delta_0)\delta_2)(\delta_0\ \mathbb{5}))$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de h hay que poner δ_0, en vez de z hay que poner δ_2 y en vez de t hay que poner $(\delta_0\ \mathbb{5})$:

$$(((\textsc{es\_vacia}\ \delta_2)(\delta_0\ \mathbb{5}))((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_2))((\delta_0\ \mathbb{5})(\textsc{true}\ \delta_2))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_2)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_2))((\delta_0\ \mathbb{5})(\textsc{true}\ \delta_2)))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_3)((\delta_0\ \mathbb{5})\mathbb{9}))$, donde δ_3 es la lista $((\textsc{bik}\ 8)\textsc{vacia})$:

$$((((\mathsf{Y}\ R)\ \delta_0)\delta_3)((\delta_0\ \mathbb{5})\mathbb{9}))\ \rightarrow_\beta\ ((((R(\mathsf{Y}\ R))\ \delta_0)\delta_3)((\delta_0\ \mathbb{5})\mathbb{9}))$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de h hay que poner δ_0, en vez de z hay que poner δ_3 y en vez de t hay que poner $((\delta_0\ \mathbb{5})\mathbb{9})$:

$$(((\textsc{es\_vacia}\ \delta_3)((\delta_0\ \mathbb{5})\mathbb{9}))((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_3))(((\delta_0\ \mathbb{5})\mathbb{9})(\textsc{true}\ \delta_3))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_3)$ es $\textsc{false}$, tenemos que desarrollar el término $((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_3))(((\delta_0\ \mathbb{5})\mathbb{9})(\textsc{true}\ \delta_3))))$. Es decir, el término $((((\mathsf{Y}\ R)\ \delta_0)\delta_4)(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8})))$, donde δ_4 es la lista $\textsc{vacia}$:

$$((((\mathsf{Y}\ R)\ \delta_0)\delta_4)(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8}))\ \rightarrow_\beta\ ((((R(\mathsf{Y}\ R))\ \delta_0)\delta_4)(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8}))$$

Ahora, en la R de la izquierda, en vez de v hay que poner $(\mathsf{Y}\ R)$ y, a continuación, en vez de h hay que poner δ_0, en vez de z hay que poner δ_4 y en vez de t hay que poner $(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8})$:

$$(((\textsc{es\_vacia}\ \delta_4)(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8}))((((\mathsf{Y}\ R)\ \delta_0)(\textsc{false}\ \delta_4))((((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8})(\textsc{true}\ \delta_4))))$$

Puesto que el valor del término $(\textsc{es\_vacia}\ \delta_4)$ es $\textsc{true}$, el término generado es $(((\delta_0\ \mathbb{5})\mathbb{9})\mathbb{8})$. Ese es el término que la función universal produce para a continuación poder llegar al valor final. El asunto es que ese término todavía no está en forma β-normal. Una serie de β-reducciones nos llevarán al término definito que sí está en forma β-normal. Ese término es 22.

A continuación, se indicará cómo hay que formular la función $\mathsf{U}_{\mathbb{N}}$ en el λ-cálculo.

El término correspondiente a la función $\mathsf{U}_{\mathbb{N}}$ en el λ-cálculo es el siguiente:

$$(\lambda h.(\lambda z((((\mathsf{Y}\ R)h)z)h)))$$

Para calcular la suma de los elementos de la lista $[5, 9, 8]$, hay que calcular la forma β-normal del siguiente término:

$$(((\lambda h.(\lambda z((((\mathsf{Y}\ R)h)z)h)))\delta_0)\delta_1)$$

donde δ_0 es el término $(\lambda u.(\lambda v.(\lambda w.(\textsc{sumar}((\textsc{sumar}\ u)v)w))))$ y donde δ_1 es el término $((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$

Es decir, hay que calcular la forma β-normal del siguiente término:

$$((((\mathsf{Y}\ R)\delta_0)\delta_1)\delta_0)$$

Y eso es lo que se ha hecho en el apartado anterior.

4.11.

Resumen

En este apartado se recogen de manera resumida los conceptos de λ-cálculo que han aparecido en los apartados previos.

4.11.1 Términos del λ-cálculo y clasificación de variables

En la figura 4.11.1 de la página 164, se muestran las tres estructuras posibles para los λ-términos y, también, se muestran la clasificación de las variables y dos funciones relacionadas con las variables.

4.11.2 Recursividad en el λ-cálculo: función Y

En la figura 4.11.2 de la página 165, se muestran la definición y el comportamiento de la función Y que se utiliza para simular la recursividad en el λ-cálculo. El término Y requiere, como dato de entrada, una función R que varía según el cálculo recursivo que se pretenda realizar. Es decir, hay que formular una función R distinta por cada cálculo recursivo que se quiera formular. La recursividad surge al calcular la forma β-normal del término $(\mathsf{Y}\ R)$.

4.11.3 Pares ordenados en el λ-cálculo: PAR

En la figura 4.11.3 de la página 165, se muestran la definición de la función PAR y la manera de utilizarla.

4.11.4 Listas en el λ-cálculo

En la figura 4.11.4 de la página 166, se muestran las definiciones de las operaciones relacionadas con listas.

λ-términos:

Variables: $h, g, f, j, a, b, x, y, z, v, \ldots, h_0, h_1, h_2, \ldots, g_0, g_1, g_2, \ldots, h', h'', \ldots$

Abstracciones: $\lambda\delta.Q$

donde δ es una variable y Q es un λ-término.

Aplicaciones: $(E\,R)$

donde E y R son dos λ-términos.

Clasificación de variables: libres, vinculantes y ligadas.

Funciones relacionadas con las variables:

$\mathsf{libres}(E)$: conjunto formado por las variables libres del λ-término E.

$\mathsf{variables}(E)$: conjunto formado por todas las variables del λ-término E.

Figura 4.11.1. Resumen: Términos del λ-cálculo y clasificación de las variables.

Definición de Y:

$$\mathsf{Y} \equiv (\lambda h.\ \underbrace{(\underbrace{(\lambda g.(h(gg)))}\ \underbrace{(\lambda g.(h(gg)))})}$$

Comportamiento recursivo del término $(\mathsf{Y}\ R)$**:**

$$(\mathsf{Y}\ R)\ \rightarrow_\beta\ (R(\mathsf{Y}\ R))\ \rightarrow_\beta\ (R(R(\mathsf{Y}\ R)))\ \rightarrow_\beta\ (R(R(R(\mathsf{Y}\ R))))\ \rightarrow_\beta \ldots$$

El término R será distinto para cada problema recursivo que se quiera resolver.

Figura 4.11.2. Resumen: definición de la función Y; comportamiento recursivo del término $(\mathsf{Y}\ R)$.

Definición de PAR:

$$\textsc{par} \equiv (\lambda h.(\lambda g.(\lambda f.((f\ h)g))))$$

Modo de utilizar PAR:

Para generar el par (a, b), se ha escribir $((\textsc{par}\ a)b)$.

Modo de acceder a los componentes de un par ordenado:

Componente de la derecha: $(((\textsc{par}\ a)b)\textsc{true})$.

Componente de la izquierda: $(((\textsc{par}\ a)b)\textsc{false})$.

Figura 4.11.3. Resumen: definición y modo de uso de la función PAR.

Lista vacía:

$\textsc{vacia} \equiv (\lambda f.\textsc{true})$

La lista no vacía $[5, 9, 8]$ en el λ-cálculo:

$((\textsc{par}\ 5)((\textsc{par}\ 9)((\textsc{par}\ 8)\textsc{vacia})))$

Función que decide si una lista es vacía:

$\textsc{es\_vacia} \equiv (\lambda z.(z(\lambda h.(\lambda g.\textsc{false}))))$

Primer elemento de un lista y resto de una lista:

$\textsc{true}$ y $\textsc{false}$.

Figura 4.11.4. Resumen: algunas operaciones relacionadas con listas.

4.12.

Símbolos griegos

Es habitual utilizar símbolos del alfabeto griego para representar fórmulas de la lógica matemática y también para denominar funciones. Debido a ello, en la tabla 4.12.1 de la página 168, se muestran los símbolos griegos que se pudieran utilizar.

Alfabeto griego moderno

	Mayúscula	minúscula	Nombre (en castellano)
1	A, A	$\alpha, \mathsf{\alpha}$	alfa
2	B, B	$\beta, \mathsf{\beta}$	beta
3	$\Gamma, \mathsf{\Gamma}$	$\gamma, \mathsf{\gamma}$	gamma
4	$\Delta, \mathsf{\Delta}$	$\delta, \mathsf{\delta}$	delta
5	E, E	$\epsilon, \mathsf{\epsilon}, \varepsilon, \mathsf{\varepsilon}$	épsilon
6	Z, Z	$\zeta, \mathsf{\zeta}$	dseta
7	H, H	$\eta, \mathsf{\eta}$	eta
8	$\Theta, \mathsf{\Theta}$	$\theta, \mathsf{\theta}, \vartheta$	ceta
9	I, I	$\iota, \mathsf{\iota}$	iota
10	K, K	$\kappa, \mathsf{\kappa}, \varkappa, \mathsf{\varkappa}$	kappa
11	$\Lambda, \mathsf{\Lambda}$	$\lambda, \mathsf{\lambda}$	lambda
12	M, M	$\mu, \mathsf{\mu}$	mu
13	N, N	$\nu, \mathsf{\nu}$	nu
14	$\Xi, \mathsf{\Xi}$	$\xi, \mathsf{\xi}$	ksi
15	O, O	o, o	ómicron
16	$\Pi, \mathsf{\Pi}$	$\pi, \mathsf{\pi}, \varpi$	pi
17	P, P	$\rho, \mathsf{\rho}, \varrho, \mathsf{\varrho}$	ro
18	$\Sigma, \mathsf{\Sigma}$	$\sigma, \mathsf{\sigma}, \varsigma$	sigma
19	T, T	$\tau, \mathsf{\tau}$	tau
20	Υ, Y	$\upsilon, \mathsf{\upsilon}$	ípsilon
21	$\Phi, \mathsf{\Phi}$	$\phi, \mathsf{\phi}, \varphi, \mathsf{\varphi}$	fi
22	X, X	$\chi, \mathsf{\chi}$	ji
23	$\Psi, \mathsf{\Psi}$	$\psi, \mathsf{\psi}$	psi
24	$\Omega, \mathsf{\Omega}$	$\omega, \mathsf{\omega}$	omega

Letras griegas no pertenecientes al alfabeto griego moderno

	Mayúscula	minúscula	Nombre (en castellano)
25	F	F	digamma
26	Q	$\mathsf{\varrho}$	qoppa
27	$\mathsf{\Koppa}$	$\mathsf{\koppa}$	koppa
28	$\mathsf{\Sampi}$	$\mathsf{\sampi}$	sampi
29	$\mathsf{\Stigma}$	$\mathsf{\stigma}$	estigma

Tabla 4.12.1. Símbolos griegos.