El cálculo lambda y los fundamentos de la computación

Eduardo Acuña Yeomans

2016

Índice general

Capítulo 1

Noción informal del cálculo lambda

El cálculo lambda es un sistema formal originalmente creado por Alonzo Church en 1932 con la finalidad de expresar, manipular y estudiar funciones para el desarrollo de los fundamentos de la lógica y las matemáticas [5, p. 248]. A lo largo de la historia, este sistema se ha adaptado para el estudio de los fundamentos de la computación y como sustento teórico para el desarrollo de lenguajes de programación.

Tres características fundamentales del cálculo lambda son

- El lenguaje e interpretación de las expresiones
- La manipulación y transformación de expresiones
- Las nociones de equivalencia entre expresiones

En este capítulo se describen estas tres características del cálculo lambda de manera informal, enfatizando las diferencias sintácticas, conceptuales y operacionales entre este sistema, la matemática clásica y la lógica de primer orden.

El contenido de este capítulo está basado en los primeros dos capítulos del libro *The Lambda Calculus, Its Syntax and Semantics* por H.P. Barendregt [2]; los capítulos 1 y 3 del libro *Lambda Calculus and Combinators, an Introduction* por J.R. Hindley y J.P. Seldin [8]; y la monografía titulada *The Calculi of Lambda-Conversion* por Alonzo Church [6].

En el estudio del cálculo lambda existen dos *lenguajes*

- El lenguaje de las expresiones del sistema
- El *metalenguaje* que describe al cálculo lambda

El lenguaje de las expresiones es un *lenguaje formal* que especifica las secuencias de símbolos que representan expresiones válidas del cálculo lambda, se relaciona con las clases de objetos del sistema que son válidos manipular, comparar y representar. Por otro lado, el metalenguaje es un lenguaje informal que permite describir *cómo* es que estas expresiones son manipuladas y analizadas, así como los mecanismos para representar conceptos y objetos matemáticos como expresiones.

1.1. Expresiones

Existen tres clases de expresiones en el cálculo lambda: los *átomos*, las *abstracciones* y las *aplicaciones*.

1.1.1. Átomos

Las expresiones más simples son los *átomos*, estos son objetos sintácticos usualmente representados con un símbolo como x, y o z. Los átomos son la representación simbólica de las variables [2, p. 577].

Cuando se tiene una expresión que se conforma de varias x se refiere al símbolo como "el átomo x", sin embargo es posible tener diferentes variables representadas con el mismo símbolo en la expresión y para referirse a una de ellas en particular se debe especificar en que parte de la expresión se encuentra la variable x a la que se hace referencia.

Al igual que en la matemática clásica y en la lógica de primer orden, las variables más relevantes son las que se *cuantifican*, por ejemplo en el estudio de funciones, la x

en $f(x) = M$ y en el estudio de enunciados lógicos, la y en $\forall y\ M$ o $\exists y\ M$.

En matemáticas y en computación es usual *nombrar* valores, por ejemplo, declarar que $x = 8$ y expresar a x en diferentes contextos que no necesariamente son expresiones lógicas y funciones. En el cálculo lambda, los nombres son parte del metalenguaje y no del lenguaje de las expresiones.

1.1.2. Abstracciones

Las abstracciones son expresiones con *estructura*, es decir, se conforman de *partes* identificables. Las expresiones que son abstracciones representan la generalización de una expresión y son usualmente asociadas al concepto de *función*.

La definición de función en la matemática clásica es el de una relación entre un conjunto de entradas, llamado *dominio* y un conjunto de salidas, llamado *codominio*. Esta relación tiene además la propiedad de que cada elemento del dominio se asocia exactamente con un elemento del codominio, formalmente, sean A y B dos conjuntos, una función f con dominio A y codominio B es un subconjunto del producto cartesiano $A \times B$, tal que para toda $a \in A$, existe $b \in B$ tal que $(a,\ b) \in f$ y si $(a,\ b') \in f$ con $b' \in B$, entonces $b = b'$.

Las funciones tienen varias maneras de ser representadas. En la definición anterior la representación es la de pares ordenados, en donde la primer componente del par es un elemento en el dominio y la segunda es un elemento en el codominio. Dependiendo del uso que se le dá a las funciones, puede ser conveniente representarlas simbólicamente con expresiones, gráficamente con dibujos, numéricamente con tablas o incluso verbalmente con palabras.

Las abstracciones en el cálculo lambda son representadas simbólicamente con un átomo y con otra expresión, se escriben de la forma $(\lambda x.M)$ donde x es algún átomo llamado variable enlazada o argumento y M es alguna expresión ya sea otra abstracción, una aplicación o un átomo a la cual llamamos cuerpo de la abstracción. Un ejemplo de

abstracción es $(\lambda y.y)$ en donde el argumento y cuerpo de la abstracción son el átomo y.

Además de la semejanza sintáctica con las funciones y las expresiones lógicas cuantificadas, las abstracciones proveen el mecanismo para asociarle un valor a una variable. Sin embargo, tanto las expresiones de funciones como las fórmulas lógicas están basadas en conjuntos y en operaciones sobre conjuntos: $f(x) = M$ asocia a x un valor en el dominio de f en M y $\exists x\ M$ asocia a x valores en el dominio del discurso en el enunciado M. En contraste con el cálculo lambda, por definición no existen conjuntos, números, valores de verdad, ni cuantificadores lógicos en el lenguaje.

Es posible utilizar la definición de función para *describir* operaciones y transformaciones de expresiones en el cálculo lambda, o utilizar lógica de primer orden para aseverar verdades en el sistema, o utilizar aritmética para cuantificar propiedades de las expresiones del cálculo lambda, sin embargo estos objetos matemáticos no están incrustados en el lenguaje de las expresiones y conforman lo que es el *metalenguaje*.

1.1.3. Aplicaciones

Las aplicaciones, al igual que las abstracciones, son expresiones con estructura. Se conforman por otras dos expresiones y se escriben de la forma $(M\,N)$ donde M y N son cualesquiera átomos, abstracciones o aplicaciones. Este tipo de expresiones son de cierta manera complementarias a las abstracciones ya que representan un potencial acto de concretar una expresión.

El concepto relacionado con las aplicaciones en la matemática clásica es el de *aplicación de funciones*, sin embargo, en el cálculo lambda se hace una distinción entre la expresión sintáctica de una aplicación y el acto de "evaluar" una aplicación.

En el estudio de funciones, la aplicación de una función f en x se denota $f(x)$ y es considerada igual al resultado de obtener el elemento del codominio de f a partir del valor de x. Por ejemplo, considerando la función $f(x) = x^2$, la expresión sintáctica

de la aplicación de esta función a 8 es $f(8)$, mientras que el valor de la evaluación de esta aplicación es 64. La notación de una abstracción del cálculo lambda es similar a la notación de aplicación de funciones, pero con el paréntesis abierto antes del nombre de la función, es decir, $(f\ x)$ en lugar de $f(x)$.

En el cálculo lambda, el concepto relacionado con la evaluación de funciones es el de *reducción* de expresiones. En particular cuando se tiene la aplicación de una abstracción $(\lambda x.M)$ en algúna expresión N, se dice que $((\lambda x.M)N)$ se *reduce* a una expresión Z la cual es similar a M solo que con x cambiada por N. En la evaluación de funciones, el procedimiento es parecido, por ejemplo $f(3)$ donde $f = 2x^2 + x - 8$, se evalúa sustituyendo las x de $2x^2 + x - 8$ por 3, obteniendo así la expresión aritmética $2(3)^2 + 3 - 8$, la cual puede ser operada para obtener el resultado 13.

Las abstracciones y aplicaciones del cálculo lambda son en algunos aspectos más restrictivos que las funciones y la aplicación de funciones. La función considera dos conjuntos cualquiera y no importa que propiedades tengan sus elementos o que operaciones se pueden realizar sobre ellos. Por otro lado, las abstracciones y aplicaciones sólo pueden ser descritas a partir de otras expresiones del cálculo lambda.

1.1.4. Extensiones

Cuando se desea representar en el cálculo lambda alguna función, se deben *codificar* como expresiones del lenguaje los elementos del dominio y el codominio de la función, así como las operaciones entre elementos de ambos conjuntos. Por ejemplo, para representar la función $f : \mathbb{N} \to \mathbb{N}$ dada por $f(x) = x^2$ primero se deben codificar los números naturales con expresiones del cálculo lambda, esta codificación debe ser acompañada de la codificación de las operaciones aritméticas elementales como la suma y resta así como de los predicados sobre números naturales como discriminar entre el mayor de dos números o si un número es cero; posteriormente se debe expresar la operación de exponenciación de cualquier número natural como base y el número 2 como exponente.

La codificación es similar a la implementación de algoritmos y estructuras de datos en lenguajes de programación.

El hecho de tener un lenguaje tan reducido y minimalista para las expresiones del cálculo lambda nos obliga a entender con detalle y precisión todos los procesos de manipulación y transformación de expresiones y siendo que todo lo que se representa con el cálculo lambda debe ser codificado como expresiones, los objetos representados pueden ser entendidos de la misma manera.

1.1.5. Ejemplos

Con solo átomos, aplicaciones y abstracciones se pueden formular expresiones complejas. A continuación se presentan seis ejemplos de expresiones del cálculo lambda y se describen diferentes maneras en las cuales estas se pueden componer para formar otras expresiones más complejas.

Ejemplo 1.1.1. Algunas expresiones del cálculo lambda

$$x \tag{a}$$

$$(\lambda x.x) \tag{b}$$

$$(y(\lambda x.x)) \tag{c}$$

$$((\lambda y.(y(\lambda x.x)))(\lambda w.w)) \tag{d}$$

$$(\lambda x.(xx)) \tag{e}$$

$$(\lambda f.(\lambda x.(f(fx)))) \tag{f}$$

Los átomos por si solos son expresiones válidas, en el inciso (a) aparece el átomo x, como tal no tiene mucha utilidad, no podemos decir que toma valores en algún conjunto o que representa algún valor en particular como falso o verdadero, es tan sólo un símbolo. Al ser parte de otra expresión, un átomo puede tener más relevancia, en el

inciso (b) el átomo x es el cuerpo de la abstracción $(\lambda x.x)$ y ahora tiene el potencial de ser cambiado por cualquier otra expresión debido a que también es el argumento.

En el inciso (c) se tiene la aplicación del átomo y en la abstracción del inciso (b). A pesar de ser contraintuitivo, las expresiones de aplicación se componen de dos expresiones cualesquiera, por lo tanto, a pesar de estar asociada conceptualmente con la aplicación de funciones, la expresión $(y(\lambda x.x))$ es válida. La expresión del inciso (d) contiene la expresión anterior en una abstracción en la primer parte de la aplicación y nos permite observar dos ideas importantes: primero, las abstracciones pueden ser aplicadas a abstracciones; segundo, al reducir la aplicación de $(\lambda y.(y(\lambda x.x)))$ a $(\lambda w.w)$, el átomo y es cambiado por la expresión $(\lambda w.w)$, el resultado puede ser reducido nuevamente para obtener la expresión $(\lambda x.x)$.

Ejemplo 1.1.2. Procedimiento de reducir la aplicación de $(\lambda y.(y(\lambda x.x)))$ en $(\lambda w.w)$

1. $((\lambda y.(y(\lambda x.x)))(\lambda w.w))$ — expresión del inciso (d)
2. $((\lambda w.w)(\lambda x.x))$ — al aplicar $(\lambda y.(y(\lambda x.x)))$ a $(\lambda w.w)$
3. $(\lambda x.x)$ — al aplicar $(\lambda w.w)$ a $(\lambda x.x)$

En el inciso (e) se presenta una abstracción cuyo cuerpo es la aplicación de su argumento sobre sí mismo. Lo interesante de esta expresión es que es que encapsula la idea de replicar cualquier expresión a la que sea aplicada. Por ejemplo, si aplicamos $(\lambda x.(x\ x))$ al átomo y y se realiza el procedimiento de reducción como en el ejemplo 1.1.2, se obtiene $(y\ y)$ y en general al realizar la reducción de $((\lambda x.(x\ x))M)$ donde M es cualquier expresión, se obtiene $(M\ M)$. Con esta expresión se puede formular una expresión auto-replicante en el cálculo lambda:

Ejemplo 1.1.3. Procedimiento de reducir la aplicación de $(\lambda x.(x\ x))$ en $(\lambda x.(x\ x))$

1. $((\lambda x.(x\ x))(\lambda x.(x\ x)))$ — expresión del inciso (e) aplicada a sí misma

En $(x\ x)$ de la expresión izquierda se cambia cada x por $(\lambda x.(x\ x))$

2. $((\lambda x.(x\ x))(\lambda x.(x\ x)))$ resultado de la reducción.

A este tipo de expresiones se les llaman "quines" [9, pp. 431–437] término originalmente asociado a una paradoja sobre sistemas lógicos [15]. En la actualidad, el término "quine" hace referencia a un programa cuya *salida* es el programa mismo.

En el inciso (f) se tiene una abstracción cuyo cuerpo es otra abstracción. El concepto interesante que ilustra esta expresión es el de representación de abstracciones de varias variables. Al reducir la aplicación de $(\lambda f.(\lambda x.(f(f\ x))))$ en una expresión cualquiera M, se obtiene $(\lambda x.(M(M\ x)))$. Si posteriormente se reduce la aplicación de este resultado a una expresión cualquiera N se obtiene $(M(M\ N))$. El resultado final es similar al resultado que se obtendría al evaluar en M y N una función $g(f, x) = f(f(x))$, es decir, $g(M, N) = M(M(N))$.

Una manera de codificar funciones de varias variables como abstracciones del cálculo lambda es codificar a *tuplas* o *secuencias* y poder hacer referencia a sus elementos de manera individual, sin embargo, representar secuencias es un procedimiento más complejo que se aborda en la sección 3.4.

1.2. Operaciones

En el cálculo lambda se pueden realizar algunas operaciones para transformar expresiones, estas operaciones son parte del metalenguaje y consisten de una serie de cambios mecánicos a la estructura de las expresiones de acuerdo a un criterio particular.

En la sección 1.1 se describió de manera tangencial el acto de *reducir* una aplicación basándonos en la noción de *cambiar* una expresión por otra, en esta sección se describen operaciones basadas en esta misma noción.

La *sustitución* es la operación que nos permite transformar una expresión cualquie-

ra M cambiando las apariciones de un átomo x por alguna otra expresión M, este procedimiento se denota

$$M\left[x := N\right]$$

En muchos casos la operación de sustitución se puede realizar de fácilmente:

Ejemplo 1.2.1. Sustituciones sencillas

$$x\left[x := y\right] = y \tag{a}$$

$$(x(x(\lambda y.y)))\left[x := z\right] = (z(z(\lambda y.y))) \tag{b}$$

$$(((w\ x)y)z)\left[x := a\right]\left[y := b\right] = (((w\ a)b)z) \tag{c}$$

$$(x\ x)\left[x :=(\lambda w.w)\right] = ((\lambda w.w)(\lambda w.w)) \tag{d}$$

Existen algunos detalles sutiles de la sustitución que se deben tomar en cuenta para evitar obtener expresiones erróneas, en particular cuando se sustituye en expresiones que contienen abstracciones. Para ilustrar estos casos especiales, consideremos la abstracción lambda análoga a una función constante $(\lambda x.y)$, en donde reducir la aplicación de ésta en cualquier otra expresión, resulta siempre en el átomo y. Si se realiza la sustitución $(\lambda x.y)\left[y := z\right]$ se obtiene la expresión $(\lambda x.z)$ la cual también es análoga a una función constante pero con el átomo z. Si no se tiene cuidado, sustituir un átomo por otro en esta abstracción puede resultar en una expresión con diferente *interpretación*.

Ejemplo 1.2.2. Caso patológico de la sustitución ingenua

$$(\lambda x.y)\left[y := x\right]$$

Se puede pensar que el resultado es $(\lambda x.x)$ la cuál es análoga a la función identidad, sin embargo, la sustitución no permite cambiar las expresiones de esta manera.

Para entender la operación de sustitución se tiene que pensar que lo que le da sentido a una variable x es una λx. Consideremos la expresión

$$(\lambda x.(\lambda y.((x\ y)z)))$$

el átomo x que aparece en el cuerpo de la expresión se dice ser una variable *ligada* a la λx, la cual se puede pensar como una especie de "referencia" a la expresión a la que la abstracción es aplicada, esto limita a la operación de sustitución a no *romper* la referencia de una variable ligada. De igual manera, el átomo y es una variable ligada a la λy y debe mantener su referencia bajo la operación de sustitución. Sin embargo, el átomo z es lo que se llama variable *libre*: No está en el *alcance* de alguna λz y puede ser libremente sustituida por alguna otra expresión.

En el ejemplo 1.2.2 se pretende sustituir la variable libre y por una expresión x, lo cual no debería presentar problemas, sin embargo, una sustitución tal cual de y por x *introduciría* una referencia a la λx de la expresión, la cuál no existía previamente. Con esto se identifica que la operación de sustitución $M\,[x := N]$ no debe introducir o eliminar referencias a alguna λ en M.

Para resolver el problema presentado en el ejemplo 1.2.2 se debe considerar otra operación llamada *cambio de variable ligada*. Se parte de la observación que en una expresión del cálculo lambda, las referencias entre λx y las variables x son más importantes que el símbolo con el que se representa el átomo. En las expresiones simbólicas de funciones sucede lo mismo, al expresar $f(x) = x^2$ y $f(y) = y^2$ hacemos referencia a la misma regla de correspondencia y por lo tanto a la misma función (sin considerar el dominio y el codominio de f). En el cálculo lambda, cambiar el símbolo que representa el átomo x en la expresión $(\lambda x.y)$ por otro símbolo no utilizado como z nos permite realizar la sustitución sin problemas.

Ejemplo 1.2.3. Procedimiento de sustitución para el ejemplo 1.2.2

1. $(\lambda x.y)\left[y := x\right]$
2. $(\lambda z.y)\left[y := x\right]$ después de realizar un cambio de variable ligada
3. $(\lambda z.x)$ resultado del procedimiento de sustitución

Cuando se realiza un cambio de variable ligada sobre una abstracción $(\lambda x.M)$ se cambia tanto el átomo x acompañado por la λ, llamada variable *enlazada* como todas las apariciones del átomo en el cuerpo de la abstracción, también llamado *alcance de* λx a menos que en M se encuentre una expresión de la forma $(\lambda x.N)$, ya que las x en N hacen referencia a la λx de $(\lambda x.N)$ no de $(\lambda x.M)$.

En el ejemplo 1.2.3 el cambio de variable ligada únicamente cambió la variable enlazada, en otras expresiones el cambio de variable ligada puede realizarse múltiples veces para transformar varias abstracciones.

Ejemplo 1.2.4. Múltiples cambios de variable ligada

1. $(\lambda f.(\lambda x.(f(f(f\ x)))))$
2. $(\lambda g.(\lambda x.(g(g(g\ x)))))$ Cambiando f por g
3. $(\lambda g.(\lambda y.(g(g(g\ y)))))$ Cambiando x por y

El cambio de variable ligada en una abstracción $(\lambda x.M)$ de x a y resulta en la abstracción

$$(\lambda y.M\left[x := y\right])$$

La definición de la operación de sustitución es recursiva y hace uso de la operación de cambio de variable ligada, considerando a x, y, z como átomos diferentes y M, N y P como expresiones cualquiera:

- $x\,[x := M]$ resulta en M;
- $y\,[x := M]$ resulta en y;
- $(M\,N)\,[x := P]$ resulta en $(M\,[x := P]\,N\,[x := P])$;
- $(\lambda x.M)\,[x := N]$ resulta en $(\lambda x.M)$ debido a que las referencias a x no deben eliminarse;
- $(\lambda y.M)\,[x := N]$ resulta en:
  - $(\lambda y.M)$ cuando x no es una variable libre en M,
  - $(\lambda y.M\,[x := N])$ cuando x es una variable libre en M pero y no es una variable libre en N debido a que esto introduciría una referencia a λy,
  - $(\lambda z.M\,[y := z]\,[x := N])$ cuando x es una variable libre en M y y es una variable libre en N.

La operación de *reducción de aplicaciones* es el mecanismo mediante el cual se puede "concretar" una abstracción haciendo uso de otra expresión como valor de la variable enlazada. De la misma manera en como se efectúa la evaluación de funciones en la matemática clásica, el concretar una función consiste en sustituir todas las apariciones del argumento por el valor en el que la función es aplicada.

La definición de la reducción de la aplicación de $(\lambda x.M)$ en una expresión cualquiera N es

$$M\,[x := N]$$

A continuación se presentan ejemplos de reducciones con los pasos de la transformación

Ejemplo 1.2.5. Procedimiento de reducir la aplicación de $(\lambda x.x)$ en y

1. $((\lambda x.x)y)$
2. $x\left[x := y\right]$
3. y

Ejemplo 1.2.6. Procedimiento de reducir la aplicación de $(\lambda w.w)$ en x dentro de otra expresión

1. $(\lambda x.((\lambda w.w)x))$
2. $(\lambda x.w\left[w := x\right])$
3. $(\lambda x.x)$

Ejemplo 1.2.7. Procedimiento de múltiples reducciones

1. $(((\lambda f.(\lambda x.(f(f(f\ x)))))g)y)$
2. $((\lambda x.(f(f(f\ x)))\left[f := g\right])y)$
3. $((\lambda x.(g(g(g\ x))))y)$
4. $(g(g(g\ x)))\left[x := y\right]$
5. $(g(g(g\ y)))$

El cálculo lambda es un sistema maleable y se permite definir operaciones arbitrarias sobre expresiones para estudiar como el sistema se comporta en diferentes contextos, por ejemplo, se puede considerar una operación similar a la sustitución que permite introducir referencias a una o más λ en una expresión, sin embargo, el presente trabajo está constituido para entender plenamente las ideas centrales del cálculo lambda haciendo solamente uso de las operaciones de *sustitución*, *cambio de variable ligada* y

aplicación de abstracciones.

1.3. Equivalencias

El cálculo lambda se considera formalmente como una *teoría ecuacional*, esto significa que los axiomas del sistema formal son ecuaciones que relacionan expresiones del lenguaje. Esto hace que el concepto de *equivalencia* de expresiones sea de suma importancia.

Es tan relevante la formalización de las nociones de equivalencia que considerar alguna equivalencia entre dos expresiones que se escriben diferente puede cambiar por completo el sistema formal que se estudia. En el desarrollo histórico del cálculo lambda, el estudio de los criterios que permiten establecer que dos expresiones son equivalentes ha dado pie a una gran diversidad de variantes de la teoría original; es por ello que en la literatura se suele hablar de *los cálculos lambda* y no únicamente de un cálculo lambda.

Como se aborda en la sección 1.2, con la operación de sustitución se puede transformar expresiones del cálculo lambda y definir otras operaciones como el cambio de variable ligada y la reducción de aplicaciones. Usualmente, las transformaciones de expresiones se pueden asociar a nociones de equivalencia. En terminología del cálculo lambda, las nociones de equivalencia entre expresiones son asociadas a la propiedad de *convertibilidad*, la cual significa que si dos expresiones M y N son equivalentes en el sistema, es posible transformar M a N y viceversa por medio de un número finito de operaciones.

En esta sección se describen algunos criterios de equivalencia entre expresiones del cálculo lambda y las maneras en las que las equivalencias se relacionan entre sí.

1.3.1. Equivalencia sintáctica

La *equivalencia sintáctica* es una relación binaria entre expresiones que no está asociada a una transformación. Se considera como una equivalencia trivial, ya que asevera la igualdad entre dos expresiones que son escritas exactamente igual, símbolo por símbolo a excepción de abusos de notación. Por ejemplo, la expresión $\sin^2(x)$ es un abuso de notación de $(\sin(x))^2$ y ambas se consideran exactamente iguales. En el cálculo lambda, la equivalencia sintáctica es denotada como

$$M \equiv N$$

cuando M es sintácticamente la misma expresión que N.

Todos los cálculos lambda, al igual que la mayoría de los sistemas formales, comprenden la noción de equivalencia sintáctica. Sin embargo las equivalencias más interesantes son las que involucran transformaciones entre expresiones.

1.3.2. α-convertibilidad

La operación de cambio de variable ligada se relaciona con una equivalencia estructural entre dos expresiones. Cuando se realiza esta operación no se modifica la estructura de la expresión, únicamente se modifica el símbolo usado para representar un átomo.

Considerando la expresión análoga a la función identidad $(\lambda x.x)$ se observa que tiene la misma estructura que la abstracción $(\lambda y.y)$ y que $(\lambda z.z)$, estas tres representan el mismo concepto. De igual manera otras expresiones como $((x\ y)z)$ o $(\lambda w.x)$ son estructuralmente equivalentes a $((a\ b)c)$ y $(\lambda f.h)$ respectivamente. A pesar de que no se escriben sintácticamente igual, la correspondencia que hay entre las posiciones de los átomos en una y otra expresión nos permite considerarlas como equivalentes. Sin embargo, la operación de cambio de variable ligada no considera cambios de nombres

a átomos que sean variables libres.

Esta relación de equivalencia es llamada α-convertibilidad y se denota como

$$M =_\alpha N$$

para dos expresiones del cálculo lambda M y N en donde a partir de un número finito de cambios de variables ligadas en M o parte de M y en N o parte de N se puedan obtener expresiones sintácticamente equivalentes.

Índices de De Bruijn

Una técnica utilizada por algoritmos que verifican si dos expresiones M y N son α-convertibles es la de *índices de De Bruijn*, esta transformación cambia la aparición de átomos por números naturales que representan la "distancia" de los átomos a las λ que hacen referencia.

Ejemplo 1.3.1 (índices de De Bruijn)**.**

$$(\lambda z.((\lambda y.(y(\lambda x.x)))(\lambda x.(z\ x))))$$

Se escribe usando índices de De Bruijn como

$$\lambda(\lambda 1(\lambda 1))(\lambda 2\ 1)$$

En la figura 1.1 se puede observar de manera gráfica la transformación de una notación a otra para este ejemplo, visualizando las expresiones del cálculo lambda como árboles.

Una desventaja de utilizar la notación de De Bruijn es que ciertas expresiones del cálculo lambda no pueden ser escritas, en particular, los átomos no pueden ser variables libres para que esta notación pueda ser utilizada.

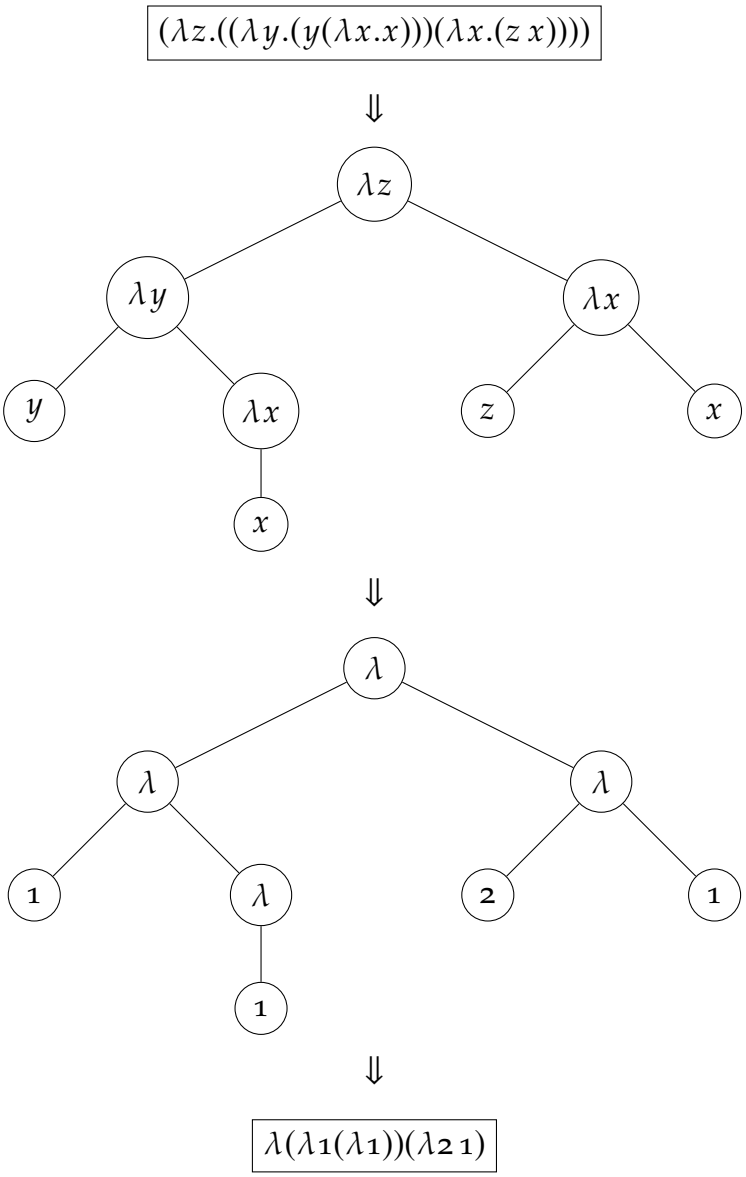


Figura 1.1: Transformación gráfica del ejemplo 1.3.1

1.3.3. β-convertibilidad

Al igual que el cambio de variable ligada, la operación de reducción de aplicaciones es utilizada para describir una equivalencia entre expresiones. La idea básica de esta equivalencia consiste en observar que al aplicar una abstracción $(\lambda x.M)$ a una expresión N, el resultado de su reducción siempre es el mismo. De manera similar a la aplicación de funciones, cuando se define una función $f(x) = x^2$, la aplicación $f(3)$ se suele igualar al resultado de la aplicación: $f(3) = 8$.

Esta relación de equivalencia es llama β-convertibilidad y se denota como

$$M =_\beta N$$

para dos expresiones M y N en donde a partir de un número finito de reducciones a aplicaciones, reducciones inversas y cambios de variable ligada en M o parte de M y N o parte de N se puedan obtener expresiones sintácticamente equivalentes.

El proceso inverso a la reducción de aplicaciones se puede ilustrar considerando la reducción de $(((\lambda g.(\lambda y.(g(g(g\ y)))))f)x)$ a $(f(f(f\ x)))$. No es posible reducir la segunda expresión a la primera, pero ya la segunda expresión es una reducción de la primera, ambas son β-convertibles:

$$(f(f(f\ x))) =_\beta (((\lambda g.(\lambda y.(g(g(g\ y)))))f)x)$$

1.3.4. Relaciones de equivalencia

Todas las nociones de convertibilidad son relaciones de equivalencia, las cuales por definición cumplen con tres propiedades.

Sea $\sim$ una relación de equivalencia

a. Toda expresión M es equivalente a sí misma, es decir, $M \sim M$.

b. Si una expresión M es relacionada con una equivalencia a otra expresión N, entonces N también es relacionada a M, es decir $M \sim N \implies N \sim M$.

c. Si una expresión M se relaciona con una equivalencia a otra expresión N y N se relaciona con la misma equivalencia a P, entonces, M y P se relacionan con esta equivalencia, es decir, $M \sim N,\ N \sim P \implies M \sim P$.

La equivalencia sintáctica corresponde al inciso a de las propiedades de equivalencias mencionadas y es llamada propiedad de *reflexividad*; al igual que la α-conversión y la β-conversión, la equivalencia sintáctica no está asociada a una regla de inferencia. En los incisos b y c se tienen inferencias que parten de expresiones equivalentes y basado en si estas expresiones son equivalentes o no, ciertas propiedades se deben cumplir. En el inciso b la propiedad es llamada *simetría*, mientras que en el inciso c la propiedad es llamada *transitividad*.

La α-conversión y la β-conversión fueron definidas como equivalencias independientes y su definición cumple con las tres propiedades mencionadas a pesar de ser definidas en base a un procedimiento y no en una regla declarativa, sin embargo, es deseable referirse a una sola equivalencia de expresiones que tenga las propiedades de *reflexividad*, *simetría* y *transitividad* y posteriormente considerar otras reglas que la equivalencia deba de cumplir.

Al igual que Haskell Curry en [7, p. 59] se utilizan las letras griegas α y β para referirse a las ecuaciones relacionadas con la α-conversión y β-conversión respectivamente y las letras ρ, σ y τ para referirse a las propiedades de reflexividad, simetría y transitividad respectivamente, se retoma esta convención para elaborar la siguiente definición de una relación de equivalencia $\sim$:

Definición 1.3.1 (Ecuaciones de $\sim$)**.** Las ecuaciones con $\sim$ que se satisfacen para expre-

siones del cálculo lambda son

$$(\lambda x.M) \sim (\lambda y.M\,[x := y]) \tag{α}$$

$$((\lambda x.M)N) \sim M\,[x := N] \tag{β}$$

$$M \sim M \tag{ρ}$$

$$M \sim N \implies N \sim M \tag{σ}$$

$$M \sim N,\ N \sim P \implies M \sim P \tag{τ}$$

Las ecuaciones en la definición 1.3.1 son muy parecidas a las propiedades de la β-conversión, con la excepción de que la β-conversión relaciona expresiones en donde sus partes fueron transformadas y $\sim$ no, por ejemplo

$$(\lambda f.((\lambda x.(f\ x))y)) =_\beta (\lambda f.(f\ y))$$

pero

$$(\lambda f.((\lambda x.(f\ x))y)) \nsim (\lambda f.(f\ y))$$

Para capturar la definición de β-convertibilidad con ecuaciones, es necesario definir a $\sim$ en partes de una expresión. Las siguientes reglas, nombradas por Curry [7, p. 59] como ν, μ y ξ, junto con las reglas de $\sim$ completan la definición declarativa de β-convertibilidad:

Definición 1.3.2. Reglas que debe cumplir $\sim$ para ser $=_\beta$

$$M \sim N \implies (M\,Z) \sim (N\,Z) \tag{ν}$$

$$M \sim N \implies (Z\,M) \sim (Z\,N) \tag{μ}$$

$$M \sim N \implies (\lambda x.M) \sim (\lambda x.N) \tag{ξ}$$

Con estas reglas y a partir de un razonamiento lógico, podemos demostrar la β-equivalencia entre dos expresiones.

Ejemplo 1.3.2. Razonamiento para concluir que $(\lambda f.((\lambda x.(f\ x))y)) =_\beta (\lambda f.(f\ y))$

1. $((\lambda x.(f\ x))y) =_\beta (f\ y)$ por β
2. $(\lambda f.((\lambda x.(f\ x))y)) =_\beta (\lambda f.(f\ y))$ por ξ

Es posible incluir aún más reglas de equivalencia cuando se estudia el cálculo lambda, a pesar de poder trabajar con expresiones en este sistema a partir de equivalencias arbitrarias, usualmente cada regla de equivalencia se asocia con alguna argumentación basada en la noción de función.

Por ejemplo, se pueden considerar dos abstracciones diferentes $(\lambda x.M)$ y $(\lambda y.N)$ que al ser aplicadas a cualquier expresión Z sean β-convertibles a una misma expresión W. Si se relacionan las abstracciones del cálculo lambda con funciones, es natural pensar que M y N sean equivalentes, ya que por definición, dos funciones f y g son equivalentes si para toda x en su dominio $f(x) = g(x)$. Por ejemplo, las funciones $f(n) = \sum_{i=0}^{n} i$ y $g(n) = \frac{n(n+1)}{2}$ a pesar de describir dos procedimientos diferentes para el cálculo de la suma de los primeros n números naturales son "funcionalmente" equivalentes ya que para todo natural $f(n) = g(n)$. Por otro lado, si se relacionan las abstracciones del cálculo lambda con algoritmos, M y N no pudieran ser consideradas equivalentes ya que en el estudio de la complejidad algorítmica, el énfasis en la comparación entre dos procedimientos no es las entradas y salidas, si no el proceso que describen. Por ejemplo, el algoritmo de ordenamiento *merge sort* logra ordenar una secuencia de n números de menor a mayor en $\mathcal{O}(n \log n)$ mientras que el algoritmo *bubble sort* computa el mismo resultado pero en $\mathcal{O}(n^2)$. La equivalencia "funcional" se pudiera incluír en la definición de $\sim$ añadiendo la siguiente regla:

$$(M\,P) \sim (N\,P) \implies M \sim N$$

Con esto se termina la introducción informal al cálculo lambda, las ideas que se han manejado en esta sección son formalizadas y definidas de manera rigurosa en el capítulo 2.

Capítulo 2

Formalización del cálculo lambda

La teoría del cálculo lambda se puede formalizar de diferentes perspectivas, en este trabajo se abordan dos: a partir de la *reducibilidad y convertibilidad* de expresiones y a partir de *sistemas formales*. La primera consiste en definir transformaciones de expresiones mediante procedimientos, mientras que la segunda define axiomas y reglas de inferencia.

Independientemente de la perspectiva de la formalización, los conceptos son los mismos y las definiciones equivalentes. En ambos casos se formaliza la teoría $\boldsymbol{\lambda}$, también llamado cálculo-$\lambda K\beta$.

De acuerdo a Barendregt [2, p. 22], el objeto de estudio principal de la teoría $\boldsymbol{\lambda}$ es el conjunto de términos lambda módulo convertibilidad, estas nociones serán presentadas en las siguientes subsecciones.

2.1. Términos lambda

Esta subsección está basada principalmente en el capítulo 2 de [2].

Los *términos lambda* son las *fórmulas bien formadas* del cálculo lambda, es decir, las expresiones válidas del sistema. El conjunto de todos los términos lambda es un lenguaje formal, denotado como Λ.

El lenguaje Λ se puede definir de diferentes maneras, a continuación se presenta una definición inductiva y posteriormente una construcción a partir de una gramática libre de contexto.

Definición 2.1.1 (Términos lambda)**.** El conjunto Λ tiene elementos que son cadenas conformadas por símbolos en el alfabeto $\Sigma = \{(,\ ),\ .,\ \lambda\} \cup V$, donde V es un conjunto infinito $\{v_0,\ v_{00},\ ...\}$ de variables. Λ es el conjunto más pequeño que satisface:

$$x \in V \implies x \in \Lambda \tag{a}$$

$$M \in \Lambda,\ x \in V \implies (\lambda x.M) \in \Lambda \tag{b}$$

$$M,\ N \in \Lambda \implies (M\,N) \in \Lambda \tag{c}$$

Cada una de estas tres reglas corresponde a las tres clases de términos lambda: la regla (a) define a todos los elementos de V como términos lambda, a estas variables se les llama *átomos*; la regla (b) define a las cadenas de la forma $(\lambda x.M)$ (donde x es un átomo y M es un término lambda) como términos lambda, a estos términos se les llama *abstracciones*; la regla (c) define a las cadenas de la forma $(M\,N)$ (donde M y N son términos lambda) como términos lambda, a estos términos se les llama *aplicaciones*.

Una definición alternativa de Λ es haciendo uso de gramáticas libres de contexto:

Definición 2.1.2 (Términos lambda)**.** El conjunto de términos lambda es el lenguaje generado por la gramática libre de contexto G con categorías sintácticas T (términos lambda), E (aplicaciones), F (abstracciones) y A (átomos); símbolos terminales

$\{(,), ., \lambda, v, {}_0\}$; símbolo inicial T y con las siguientes reglas de producción:

$$T \to E \mid F \mid A \tag{a}$$

$$A \to \mathrm{v}_0 \mid A_0 \tag{b}$$

$$F \to (\lambda\, A\, .\, T\,) \tag{c}$$

$$E \to (\, T\, T\,) \tag{d}$$

Para facilitar la escritura y lectura de los términos lambda, en este trabajo se hacen las siguientes consideraciones sobre la notación:

- Cuando se hace referencia a cualquier término lambda se utilizan las letras mayúsculas M, N, P, etc. Es importante establecer que si en un ejemplo, explicación, teorema o demostración se hace referencia a un término lambda con una letra mayúscula, cualquier otra aparición de esta letra hace referencia a este mismo término dentro de ese contexto.
- Cuando se hace referencia a cualquier átomo se utilizan las letras minúsculas x, y, z, etc. Al igual que en el punto anterior, la aparición de una letra minúscula en un ejemplo, explicación, teorema o demostración hace referencia al mismo átomo.
- Los paréntesis son omitidos de acuerdo a las siguientes equivalencias sintácticas:
  - $((M\,N)P) \equiv M\,N\,P$, en general, se considera la aplicación de términos lambda con asociación a la izquierda. Se tiene que tener cuidado con respetar esta regla, por ejemplo $(M(N(O\,P))) \equiv M(N(O\,P)) \not\equiv M\,N\,O\,P$.
  - $(\lambda x.(MN)) \equiv \lambda x.(MN)$, en general, se puede escribir una abstracción omitiendo los paréntesis externos. Es importante escribir de manera explícita los paréntesis en algunos casos, por ejemplo $((\lambda x.(M\,N))O) \equiv (\lambda x.(M\,N))O \not\equiv \lambda x.(M\,N)O$ ya que el lado derecho de la equivalencia es sintácticamente equivalente a $(\lambda x.((M\,N)O))$.

- $(\lambda x.(\lambda y.(\lambda z.M))) \equiv (\lambda x\ y\ z.M)$, en general, si el cuerpo de una abstracción es también una abstracción, se pueden agrupar las variables ligadas y enlazadas. Éste abuso de notación es consistente con la reducción de funciones de varias variables usada por Schönfinkel [17].

- El símbolo $\equiv$ denota la equivalencia sintáctica entre dos términos lambda.

A continuación se muestran ejemplos de términos lambda asociados a términos sintácticamente equivalentes pero escritos con abuso de notación:

Ejemplo 2.1.1.

$$(((x\ y)z)(y\ x)) \equiv x\ y\ z(y\ x)$$

$$(\lambda x.((u\ x)y)) \equiv \lambda x.u\ x\ y$$

$$(\lambda y.(u(\lambda x.y))) \equiv \lambda y.u(\lambda x.y)$$

$$(((\lambda y.((v\ u)u))z)y) \equiv (\lambda y.v\ u\ u)z\ y$$

$$(((u\ x)(y\ z))(\lambda v.(v\ y))) \equiv u\ x(y\ z)(\lambda v.v\ y)$$

$$((((\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))u)v)w) \equiv (\lambda x\ y\ z.x\ z(y\ z))u\ v\ w$$

Para hacer referencia a una secuencia con una cantidad arbitraria de términos lambda se usa la notación $\vec{x} = x_1, \ldots, x_n$ cuando es secuencia de átomos y $\vec{M} = M_1, \ldots, M_n$ cuando es secuencia de términos lambda en general. Con esta notación se puede abreviar

$$\lambda\vec{x}.M \equiv \lambda x_1\ x_2 \ldots x_n.M$$

$$M\vec{N} \equiv M\ N_1\ N_2 \ldots N_n$$

En algunas demostraciones realizadas por inducción, se usa la expresión “inducción

sobre M" para referirse a la inducción sobre la *longitud* de M.

Definición 2.1.3 (Longitud)**.** La longitud de un término lambda, denotado como $\|M\|$, es la cantidad de apariciones de átomos en el término lambda, se determina a partir de la estructura del término lambda como:

$$\|x\| = 1$$
$$\|M\,N\| = \|M\| + \|N\|$$
$$\|\lambda x.M\| = 1 + \|M\|$$

Por ejemplo, la longitud del término lambda $x(\lambda y.y\ u\ x)$ es 5.

Una cuestión importante al momento de demostrar un teorema o definir un concepto por inducción sobre un término lambda es que usualmente la inducción matemática relaciona proposiciones con números naturales. Sin embargo es posible tener dos términos diferentes M y N tal que $\|M\| = \|N\|$, por ejemplo $\lambda x.x$ y $z\ z$ tienen longitud 2. La inducción sobre la longitud de un término lambda considera la estructura del término, de tal manera que para una proposición P sobre un término lambda M, los casos base de la inducción son aquellos en donde la estructura no es compuesta (en átomos cuya longitud siempre es 1) y la hipótesis de inducción considera que P se cumple para los subtérminos de M cuya longitud siempre es estrictamente menor que $\|M\|$.

El concepto de aparición de un término lambda en otro se formaliza a partir del concepto de subtérmino:

Definición 2.1.4 (Subtérmino)**.** M es un subtérmino de N, denotado $M \subset N$ si $M \in$

Sub(N) es la colección de subtérminos de N definida de manera inductiva como

$$\begin{aligned} \text{Sub}(x) &= \{x\} \\ \text{Sub}(\lambda x.M) &= \text{Sub}(M) \cup \{\lambda x.M\} \\ \text{Sub}(M\,N) &= \text{Sub}(M) \cup \text{Sub}(N) \cup \{M\,N\} \end{aligned}$$

Definición 2.1.5 (Aparición)**.** La aparición de M en N implica que $M \subset N$ o que M es *el* átomo del argumento de una abstracción en N.

Un subtérmino N de M puede aparecer varias veces en M, cuando dos subtérminos N_1 y N_2 de M no tienen apariciones de átomos en común, se dice que son *disjuntas*. Cuando N es subtérmino de M se le llama *activo* si aparece en una aplicación de la forma $N\,Z$, de lo contrario, se le llama *pasivo*.

Cuando $\lambda x.M$ es un subtérmino de P, se dice que la aparición M es el *alcance* de la aparición del átomo x que acompaña a la λ.

Ejemplo 2.1.2. Sea $M \equiv \lambda x.x\ y(\lambda z.y)$:

- el término $x\ y \subset M$;
- el átomo $z \not\subset M$ pero si aparece en M, debido a que z acompaña a λ;
- el término $y(\lambda z.y)$ a pesar de parecer ser un subtérmino de M no lo es, esto se puede corroborar escribiendo los términos sin el abuso de notación: $y(\lambda z.y) \equiv (y(\lambda z.y))$ y $M \equiv \lambda x.xy(\lambda z.y) \equiv (\lambda x.((x\ y)(\lambda z.y)))$, en este caso, la clave está en observar la estructura de la aplicación $x\ y(\lambda z.y)$.
- Las apariciones de x y $\lambda z.y$ en M son disjuntas.
- Los términos x y $(x\ y)$ son subtérminos activos de M, mientras que y y $\lambda z.y$ son subtérminos pasivos.

Las variables de un término lambda se pueden clasificar de diferentes maneras de acuerdo a la posición que tienen en el término y a los subtérminos a los que se asocian.

Definición 2.1.6 (Clasificación de variables)**.** La aparición de un átomo x en un término P es llamada:

- *variable ligada* si es un subtérmino de M en una abstracción $\lambda x.M$ en P;
- *variable enlazada* si y sólo si es la x que acompaña la λ de $\lambda x.M$ en P;
- *variable libre* en otro caso.

Es importante aclarar la diferencia entre un átomo x como subtérmino de un término lambda M y una aparición de x en M: la aparición hace referencia a la posición de x en M. Por ejemplo, en el término lambda $(\lambda x.x)x$ la primera aparición del átomo x es una variable enlazada, la segunda aparición es una variable ligada y la tercera aparición es una variable libre.

Ejemplo 2.1.3. Sea $M \equiv x(\lambda y.x\ y)$:

- El átomo x aparece como variable libre dos veces en M;
- El átomo y aparece como variable ligada en M.

En la definición formal de algunos conceptos es conveniente hacer referencia a las variables libres de un término lambda:

Definición 2.1.7 (Variables libres)**.** El conjunto de variables libres de un término lambda M se denota $\mathrm{FV}(M)$ y se define de manera inductiva como:

$$\mathrm{FV}(x) = \{x\}$$
$$\mathrm{FV}(\lambda x.M) = \mathrm{FV}(M) \setminus \{x\}$$
$$\mathrm{FV}(M\,N) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$$

Cuando $\mathrm{FV}(M) = \emptyset$ se dice que M es un *combinador* o *término cerrado*.

Ejemplo 2.1.4. Consideremos los términos $x(\lambda x.x\ y\ z)$, $\lambda x\ y\ z.y$ y $(\lambda y.x)\lambda x.y$.

- $\mathrm{FV}(x(\lambda x.x\ y\ z)) = \{x,\ y,\ z\}$;
- $\mathrm{FV}(\lambda x\ y\ z.y) = \emptyset$, por lo tanto es un combinador;
- $\mathrm{FV}((\lambda y.x)\lambda x.y) = \{x,\ y\}$.

En ocaciones es importante distinguir los términos lambda cerrados de aquellos que contienen variables libres, para ello se identifica el subconjunto de Λ que contiene a todos los términos cerrados:

Definición 2.1.8 (Términos cerrados)**.** Se denota como Λ^o al conjunto

$$\{M \in \Lambda \mid M \text{ es un término cerrado}\}$$

La notación Λ^o se puede generalizar para identificar diferentes subconjuntos de Λ a partir de las variables libres de los términos lambda:

$$\Lambda^o(\vec{x}) = \{M \in \Lambda \mid \mathrm{FV}(M) \subseteq \{\vec{x}\}\}$$

De tal manera que:

$$\Lambda^o = \Lambda^o(\emptyset)$$

Todos los términos lambda en $\Lambda \setminus \Lambda^o$ tienen al menos una clausura en Λ^o.

Definición 2.1.9 (Clausura)**.** La clausura de un término lambda M con $\mathrm{FV}(M) \neq \emptyset$ es un término lambda

$$(\lambda \vec{x}.M)$$

con $\vec{x} = \mathrm{FV}(M)$

Ejemplo 2.1.5. Consideremos el términio lambda $M \equiv \lambda z.x\ y\ z$

- $\lambda x\ y.\lambda z.x\ y\ z$ es una clausura de M;
- $\lambda y\ x\ z.x\ y\ z$ es una clausura de M;
- $\lambda z\ x\ y.\lambda z.x\ y\ z$ no es una clausura de M.

Al escribir términos lambda con repetición de aplicaciones suele ser conveniente utilizar una notación más compacta. Cuando se aplica n veces un término F por la izquierda a otro término M se denota $F^n\ M$. Cuando se aplica n veces un término M por la derecha a otro término F se denota $F\ M^{\sim n}$. La definición formal de esta notación es:

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

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

Ejemplo 2.1.6. El término lambda $\lambda f\ x.f(f(f(f\ x)))$ se puede escribir de manera compacta como

$$\lambda f\ x.f^4\ x$$

Ejemplo 2.1.7. El término lambda $\lambda f\ x.f\ x\ x\ x\ x$ se puede escribir de manera compacta como

$$\lambda f\ x.f\ x^{\sim 4}$$

Definición 2.1.10 (Sustitución)**.** Para cualquier M, N y x, se define $M[x := N]$ como el resultado de sustituir cada aparición libre de x por N en M de acuerdo a las siguientes

reglas:

$$
\begin{aligned}
x[x := N] &\equiv N; \\
a[x := N] &\equiv a && a \not\equiv x; \\
(P\,Q)[x := N] &\equiv P[x := N]\,Q[x := N]; \\
(\lambda x.P)[x := N] &\equiv \lambda x.P; \\
(\lambda y.P)[x := N] &\equiv \lambda y.P && x \not\equiv y,\ x \notin \mathrm{FV}(P); \\
(\lambda y.P)[x := N] &\equiv \lambda y.P[x := N] && x \not\equiv y,\ x \in \mathrm{FV}(P),\ y \notin \mathrm{FV}(N); \\
(\lambda y.P)[x := N] &\equiv \lambda z.P[y := z][x := N] && x \not\equiv y,\ x \in \mathrm{FV}(P),\ y \in \mathrm{FV}(N),\ z \notin \mathrm{FV}(NP).
\end{aligned}
$$

Ejemplo 2.1.8. Procedimientos de sustituciones para cada uno de los casos de la definición 2.1.10:

- Caso $x[x := N]$

$$y[y := \lambda x.x] \equiv \lambda x.x$$

- Caso $a[x := N]$, donde $a \not\equiv x$

$$z[w := x\,x] \equiv z$$

- Caso $(P\,Q)[x := N]$

$$
\begin{aligned}
(y\,x\,x)[x := y] &\equiv ((y\,x)x)[x := y] \\
&\equiv (y\,x)[x := y]\,x[x := y] \\
&\equiv (y[x := y]\,x[x := y])y \\
&\equiv y\,y\,y
\end{aligned}
$$

- Caso $(\lambda x.P)[x := N]$

$$(\lambda f\, x.f\ f\ x)[f := g] \equiv \lambda f\, x.f\ f\ x$$

- Caso $(\lambda y.P)[x := N]$, donde $x \not\equiv y$, $x \notin \mathrm{FV}(P)$

$$(\lambda f\, x.f\ f\ x)[f := g] \equiv \lambda f\, x.f\ f\ x$$

- Caso $(\lambda y.P)[x := N]$, donde $x \not\equiv y$, $x \in \mathrm{FV}(P)$, $y \notin \mathrm{FV}(N)$

$$\begin{aligned}(\lambda f.x\ \lambda x.f\ f\ x)[x := y] &\equiv \lambda f.(x\ \lambda x.f\ f\ x)[x := y] \\ &\equiv \lambda f.x[x := y]\,(\lambda x.f\ f\ x)[x := y] \\ &\equiv \lambda f.y\ \lambda x.f\ f\ x\end{aligned}$$

- Caso $(\lambda y.P)[x := N]$, donde $x \not\equiv y$, $x \in \mathrm{FV}(P)$, $y \in \mathrm{FV}(N)$ y $z \notin \mathrm{FV}(NP)$

$$\begin{aligned}(\lambda f.x\ \lambda x.f\ f\ x)[x := f] &\equiv \lambda g.(x\ \lambda x.f\ f\ x)[f := g][x := f] \\ &\equiv \lambda g.(x[f := g](\lambda x.f\ f\ x)[f := g])[x := f] \\ &\equiv \lambda g.(x\ \lambda x.(f\ f\ x)[f := g])[x := f] \\ &\equiv \lambda g.(x\ \lambda x.((f\ f)[f := g]x[f := g]))[x := f] \\ &\equiv \lambda g.(x\ \lambda x.((f[f := g]\ f[f := g])x))[x := f] \\ &\equiv \lambda g.(x\ \lambda x.g\ g\ x)[x := f] \\ &\equiv \lambda g.x[x := f](\lambda x.g\ g\ x)[x := f] \\ &\equiv \lambda g.f\ \lambda x.g\ g\ x\end{aligned}$$

En el último caso es importante observar que las apariciones ligadas de x no se sustituyen.

Lema 2.1.1. *Si* $(y\ x) \notin \mathrm{FV}(L)$ $y\ x \not\equiv y$, *entonces*

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$$

En contraste a la operación de sustitución en donde no se permite introducir o quitar referencias a variables enlazadas, el *contexto* es un término con "hoyos":

Definición 2.1.11 (Contexto)**.** Un contexto es un término lambda denotado $C[\ ]$ definido de manera inductiva:

- x es un contexto;
- $[\ ]$ es un contexto;
- Si $C_1[\ ]$ y $C_2[\ ]$ son contextos, entonces $C_1[\ ]\,C_2[\ ]$ y $\lambda x.C_1[\ ]$ también lo son.

Si $C[\ ]$ es un contexto y $M \in \Lambda$, entonces $C[M]$ denota el resultado de reemplazar por M los hoyos de $C[\ ]$. Al realizar esto, las variables libres de M pueden convertirse en variables ligadas de $C[M]$.

Ejemplo 2.1.9. Consideremos el contexto $C[\ ] \equiv \lambda x.x\ \lambda y.[\ ]$ y el término lambda $M \equiv (x\,y)$.

$$\begin{aligned} C[M] &\equiv (\lambda x.x\ \lambda y.[\ ])[(x\ y)] \\ &\equiv (\lambda x.x\ \lambda y.(x\ y)) \end{aligned}$$

El caso análogo con la sustitución es

$$
\begin{aligned}
(\lambda x.x\ \lambda y.w)[w := (x\ y)] &\equiv \lambda z.(x\ \lambda y.w)[x := z][w := (x\ y)] \\
&\equiv \lambda z.(x[x := z](\lambda y.w)[x := z])[w := (x\ y)] \\
&\equiv \lambda z.(z\ \lambda y.w)[w := (xy)] \\
&\equiv \lambda z.z[w := (x\ y)](\lambda y.w)[w := (x\ y)] \\
&\equiv \lambda z.z\ \lambda v.w[w := (x\ y)] \\
&\equiv \lambda z.z\ \lambda v.(x\ y)
\end{aligned}
$$

2.2. Teoría $\lambda K\beta$

El objetivo principal de esta subsección es presentar una formalización del cálculo lambda descrito en el capítulo 1 desde el punto de vista de teorías formales. El nombre técnico de la teoría formal principal de este trabajo es $\boldsymbol{\lambda K\beta}$, se pueden realizar modificaciones y extensiones a esta teoría y los siguientes conceptos permiten estudiar las implicaciones de estos cambios.

Una *teoría formal* $\mathcal{T}$ es una tripleta $(\mathcal{F}, \mathcal{A}, \mathcal{R})$ donde

- $\mathcal{F}$ es el conjunto de todas las *fórmulas* $X = Y$ con X y Y elementos de un lenguaje formal;
- $\mathcal{A}$ es un conjunto de *axiomas* y $\mathcal{A} \subseteq \mathcal{F}$;
- $\mathcal{R}$ es un conjunto de *reglas*.

Una regla es una función $\phi\colon \mathcal{F}^n \to \mathcal{F}$ con $n \geq 1$. Una secuencia de fórmulas $\langle A_1, \ldots, A_n, B\rangle$ tal que

$$\phi(A_1, \ldots, A_n) = B$$

es llamada una *instancia* de ϕ, donde las *premisas* de la instancia son las fórmulas $\langle A_1, ..., A_n \rangle$ y la *conclusión* de la instancia es B. Las instancias se denotan

$$\frac{A_1 \quad ... \quad A_n}{B}$$

Observación. En la literatura se pueden encuentrar diferentes maneras de trabajar con teorías formales, dependiendo del estilo de las teorías y de la manera en como se formaliza su definición, por ejemplo en [1] las reglas se definen como conjuntos de secuencias $\langle A_1, ..., A_{n+1} \rangle$ con n premisas y una conclusión, en donde los axiomas se definen como las reglas en $\mathscr{R}$ con cero premisas. La definición de teoría formal es del estilo Hilbert y está basada en [8, pp. 69–70].

Si consideramos un conjunto $\Gamma \subseteq \mathscr{F}$, una *deducción* deuna fórmula B desde Γ es un árbol dirigido de fórmulas en donde los vértices de un extremo son elementos de $\mathscr{A}$ o Γ, los vértices intermedios son deducidos a partir de los vértices que inciden en ellos a partir de una regla y el vértice de el otro extremo siendo B. Las fórmulas que no sean axiomas cuyo grado de entrada sea cero son llamadas *suposiciones*. Si y solo si existe una deducción para una fórmula B, se dice que B es demostrable en $\mathscr{T}$ suponiendo Γ, denotado

$$\mathscr{T}, \Gamma \vdash B$$

En caso que la deducción no tenga suposiciones, se dice que es una *demostración* y que B es un *teorema*. Cuando $\Gamma = \emptyset$ se escribe

$$\mathscr{T} \vdash B$$

A continuación se presenta la definición de la teoría $\boldsymbol{\lambda K \beta}$, debido a que esta teoría será la que se estudia principalmente en este trabajo, se acorta el nombre $\boldsymbol{\lambda K \beta}$ a $\boldsymbol{\lambda}$.

En el caso particular de la teoría $\boldsymbol{\lambda K \beta}$ se considera que $\mathscr{F}$ tiene elementos $M = N$

donde $M, N \in \Lambda$.

Definición 2.2.1 (Teoría $\boldsymbol{\lambda}$)**.** El conjunto de fórmulas tiene como elementos ecuaciones de la forma:

$$M = N \qquad\qquad \forall M, N \in \Lambda$$

Los axiomas son:

$$\lambda x.M = \lambda y.M[x := y] \qquad \forall y \notin \mathrm{FV}(M) \qquad (\alpha)$$

$$(\lambda x.M)N = M[x := N] \qquad (\beta)$$

$$M = M \qquad (\rho)$$

Y las reglas:

$$\frac{M = N}{Z\,M = Z\,N} \qquad (\mu)$$

$$\frac{M = N}{M\,Z = N\,Z} \qquad (\nu)$$

$$\frac{M = N}{\lambda x.M = \lambda x.N} \qquad (\xi)$$

$$\frac{M = N \quad N = P}{M = P} \qquad (\tau)$$

$$\frac{M = N}{N = M} \qquad (\sigma)$$

Al inicio de esta sección, se menciona que el objeto de estudio de la teoría $\boldsymbol{\lambda}$ es el conjunto de términos lambda módulo convertibilidad. La *convertibilidad* es la noción básica de equivalencia de términos lambda y las ecuaciones de la teoría $\boldsymbol{\lambda}$ formalizan

esta noción.

La relación binaria = en las ecuaciones de la teoría, es una relación de equivalencia y al igual que toda relación de equivalencia es *reflexiva*, *simétrica* y *transitiva*, en = estas propiedades son descritas en las reglas (ρ), (σ) y (τ) respectivamente. La *clase de equivalencia* de un término lambda M con respecto a esta relación de equivalencia es el conjunto de todos los términos lambda N tal que $M = N$, denotado:

$$[M]_\lambda = \{N \in \Lambda \mid M = N\}$$

La frase "módulo convertibilidad" se refiere al conjunto de todas las clases de equivalencia de Λ considerando la relación de equivalencia de la teoría formal con la que se esté trabajando. Que este conjunto sea el objeto de estudio de la teoría $\boldsymbol{\lambda}$ significa que cada elemento de Λ módulo convertibilidad (denotado $\Lambda/ =_\lambda$) es distinto y representa a una clase de términos lambda considerados en $\boldsymbol{\lambda}$ como equivalentes.

Definición 2.2.2 (Demostrabilidad)**.** La demostrabilidad en $\boldsymbol{\lambda}$ de una ecuación $M = N$ es denotada $\boldsymbol{\lambda} \vdash M = N$ e implica que $M = N$ es un teorema en $\boldsymbol{\lambda}$. En caso que la ecuación sea demostrable se dice que M y N son términos *convertibles*.

Ejemplo 2.2.1. Consideramos los términos lambda $M \equiv (\lambda f.x((\lambda y.y\ f)\lambda z.z))w$ y $N \equiv x\ w$. Se muestra que

$$\boldsymbol{\lambda} \vdash M = N$$

formulando el árbol de deducción de la figura 2.1.

Observación. Es posible formular mas de un árbol de deducción para un teorema en una teoría formal.

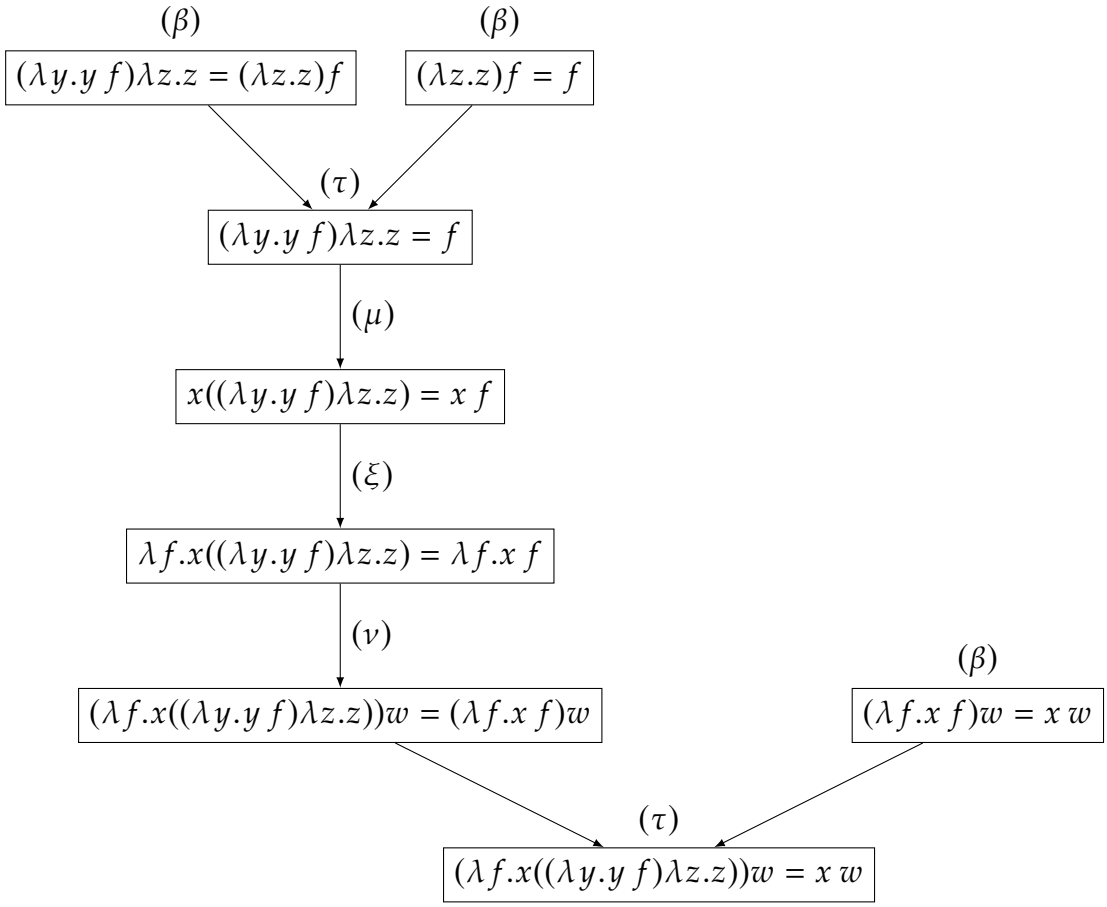


Figura 2.1: Árbol de deducción para la ecuación del ejemplo 2.2.1

Definición 2.2.3 (Combinadores SKI)**.** Tres términos lambda de suma importancia son

$$\boldsymbol{I} \equiv \lambda x.x$$

$$\boldsymbol{K} \equiv \lambda x\, y.x$$

$$\boldsymbol{S} \equiv \lambda x\, y\, z.x\, z(y\, z)$$

Corolario. *Para todo término* $M, N, L \in \Lambda$

$$\boldsymbol{I}\, M =_\lambda M$$

$$\boldsymbol{K}\, M\, N =_\lambda M$$

$$\boldsymbol{S}\, M\, N\, L =_\lambda M\, L(N\, L)$$

Estos tres combinadores generan en la teoría $\boldsymbol{\lambda}$ al conjunto Λ^o con combinaciones de aplicaciones. Debido a que $\boldsymbol{SKK} =_\lambda \boldsymbol{I}$, sólo es necesario combinar con aplicaciones a $\boldsymbol{K}$ y a $\boldsymbol{S}$ para generar cualquier término cerrado.

Definición 2.2.4 (Bases)**.** 1. Sea $\mathfrak{X} \subset \Lambda$. El conjunto de términos *generado* por $\mathfrak{X}$, denotado $\mathfrak{X}^+$, es el conjunto mas pequeño tal que

   a) $\mathfrak{X} \subseteq \mathfrak{X}^+$,

   b) $M, N \in \mathfrak{X}^+ \implies (MN) \in \mathfrak{X}^+$.

2. Sea $\mathcal{P} \subset \Lambda$. $\mathfrak{X} \subset \Lambda$ es una *base* para $\mathcal{P}$ si para toda $M \in \mathcal{P}$ existe $N \in \mathfrak{X}^+$ tal que $N = M$.

3. $\mathfrak{X}$ es llamada una *base* si $\mathfrak{X}$ es una base para Λ^o.

Lema 2.2.1. *Sea* $\lambda x.M$ *una abstracción tal que* Sub(M) *no contiene abstracciones*

1. *Si* $M = x$, *entonces* $\lambda x.M = \boldsymbol{I}$;

2. *Si* $x \notin \mathrm{FV}(M)$, *entonces* $\lambda x.M = (\boldsymbol{K}M)$;

3. *Si* $M = P\,Q$, *entonces* $\lambda x.M = \boldsymbol{S}(\lambda x.P)(\lambda x.Q)$.

Demostración. Ver 2.2.3 y 2.2

1.

$$\begin{aligned}(\boldsymbol{I}\,N) &= N \\ &= ((\lambda x.x)N) \\ &= ((\lambda x.M)N)\end{aligned}$$

2.

$$\begin{aligned}((\boldsymbol{K}\,M)N) &= (\lambda x\,y.x)M\,N \\ &= (\lambda y.M)N \\ &= ((\lambda x.M)N)\end{aligned}$$

3.

$$\begin{aligned}\boldsymbol{S}(\lambda x.P)(\lambda x.Q) &= (\lambda abc.(a\,c)(b\,c))(\lambda x.P)(\lambda x.Q) \\ &= \lambda c.((\lambda x.P)c)((\lambda x.Q)c) \\ &= \lambda c.P[x := c]Q[x := c] \\ &= \lambda c.(P\,Q)[x := c] \\ &= \lambda x.P\,Q \\ &= \lambda x.M\end{aligned}$$

□

Proposición 2.2.2. *$\{\boldsymbol{S}, \boldsymbol{K}, \boldsymbol{I}\}$ es una base, es decir, para todo término $M \in \Lambda^o$, existe un término M' compuesto de aplicaciones de $\boldsymbol{S}$, $\boldsymbol{K}$ e $\boldsymbol{I}$ tal que $M = M'$.*

La demostración de la proposición 2.2.2 consiste en la construcción de un algoritmo para transformar M a M'.

Demostración. Sea $M \in \Lambda^o$, se construye un término $M' \in \{\boldsymbol{S}, \boldsymbol{K}, \boldsymbol{I}\}^+$ tal que $M' = M$ enumerando los subtérminos en M que sean abstracciones de menor a mayor longitud.

Sea $\lambda x.N$ la abstracción con menor longitud en M, según la estructura de N se aplican las siguientes transformaciones:

1. Si $N = a$

   a) Si $a = x$ se transforma $\lambda x.N$ a $\boldsymbol{I}$ en M.

   b) Si $a \neq x$ se transforma $\lambda x.N$ a $(\boldsymbol{K}\, a)$ en M.

2. $N = (P\, Q)$

   a) Si $x \notin \mathrm{FV}(P)$ y $x \notin \mathrm{FV}(Q)$ se transforma $\lambda x.P\, Q$ a $\boldsymbol{S}(\boldsymbol{K}\, P)(\boldsymbol{K}\, Q)$ en M.

   b) Si $x \notin \mathrm{FV}(P)$ y $x \in \mathrm{FV}(Q)$ se transforma $\lambda x.P\, Q$ a $\boldsymbol{S}(\boldsymbol{K}\, P)(\lambda x.Q)$ en M.

   c) Si $x \in \mathrm{FV}(P)$ y $x \notin \mathrm{FV}(Q)$ se transforma $\lambda x.P\, Q$ a $\boldsymbol{S}(\lambda x.P)(\boldsymbol{K}\, Q)$ en M.

   d) Si $x \in \mathrm{FV}(P)$ y $x \in \mathrm{FV}(Q)$ se transforma $\lambda x.P\, Q$ a $\boldsymbol{S}(\lambda x.P)(\lambda x.Q)$ en M.

En los casos 2*a*, 2*b*, 2*c*, 2*d* se forman abstracciones con longitud menor a $\lambda x.N$, por lo tanto serán las que se transformarán después. Ya que la longitud de estas abstracciones es estrictamente menor a $\lambda x.N$ y los casos base 1*a* y 1*b* de la transformación no introducen abstracciones, en una cantidad finita de pasos el término M transformado no tendrá abstracciones de la forma $\lambda x.N$.

Para un término M con sólo una abstracción, a aplicaciones y v variables ligadas (no enlazadas) una cota superior para la máxima cantidad de pasos se calcula considerando que para los términos de la forma $\lambda x.a$ se cumple el caso 1*b* (el cual aumenta la cantidad

de aplicaciones en 1) y que para los términos de la forma $\lambda x.P\,Q$ se cumple el caso 2*d* (el cual aumenta la cantidad de aplicaciones en 2) ya que en estos casos se produce el término con mas aplicaciones, las cuales determinan la cantidad de veces que se repite el algoritmo por cada abstracción en M. Para calcular la cota superior de la cantidad de aplicaciones a' que produce el algoritmo para un término con n abstracciones se plantea la siguiente relación de recurrencia:

$$a'_0 = a$$
$$a'_n = 2 \times a'_{n-1} + v$$

Esta recurrencia describe la función $a' : \mathbb{N} \to \mathbb{N}$:

$$a'(n) = 2^n \times a + (2^n - 1) \times v$$

Para la cota superior de la cantidad de pasos realizados por el algoritmo para un término M con n abstracciones, se plantea la siguiente relación de recurrencia basada en a' y en el hecho de que la cantidad de variables ligadas y no enlazadas no aumenta en los pasos del algoritmo:

$$p_0 = 0$$
$$p_n = p_{n-1} + a'(n-1) + v$$

Esta recurrencia describe la función $p : \mathbb{N} \to \mathbb{N}$:

$$p(n) = (a + v) \times \sum_{i=0}^{n-1} 2^i$$
$$= (a + v) \times (2^n - 1)$$

□

2.2.1. Lemas y corolarios sobre términos lambda en $\lambda K\beta$

Resultados sobre términos lambda en la teoría $\boldsymbol{\lambda K\beta}$.

2.3. Otras teorías

2.3.1. $\lambda I\beta$

En el artículo [6], Alonzo Church presenta una definición del cálculo lambda con un conjunto restringido de términos lambda. A la teoría que considera a este conjunto restringido de términos lambda (denotado Λ_I) y los axiomas y reglas de inferencia de la teoría $\boldsymbol{\lambda}$ cambiando Λ por Λ_I se le conoce como teoría $\boldsymbol{\lambda I\beta}$ (o el cálculo λI).

Definición 2.3.1 (Términos en Λ_I)**.**

$$x \in V \implies x \in \Lambda_I$$
$$M \in \Lambda_I,\ x \in \mathrm{FV}\, M \implies \lambda x.M \in \Lambda_I$$
$$M, N \in \Lambda_I \implies M\, N \in \Lambda_I$$

La diferencia fundamental entre las teorías $\boldsymbol{\lambda K\beta}$ y $\boldsymbol{\lambda I\beta}$ es el término lambda $\boldsymbol{K}$, ya que $\boldsymbol{K} \in \Lambda \setminus \Lambda_I$ pero $\boldsymbol{K} \notin \Lambda_I$. Esto es debido a que el subtérmino $\lambda y.x$ en $\boldsymbol{K}$ de la definición 2.2.3 no puede existir en Λ_I debido a que $y \notin \mathrm{FV}(x)$.

2.3.2. Extensionalidad

El concepto de igualdad de funciones usado en la mayoría de las ramas de la matemática es lo que se conoce como "extensional", esta propiedad de las relaciones de equivalencia hace referencia a las características externas de los objetos que compara, en el caso de las funciones, se incluye la suposición de que para funciones f y g con el mismo dominio

$$\forall x[f(x) = g(x)] \implies f = g$$

Contraria a esta suposición, en la computación, el tema central son los procedimientos y procesos que describen los programas o algoritmos, cuyas igualdades "intensional", es decir, si dos programas computan la misma función matemática, no necesariamente se dice que son el mismo programa ya que uno pudiera ser mas eficiente que otro (la característica de eficiencia es interna a cada algorítmo).

La teoría $\boldsymbol{\lambda}$ también es intensional: existen dos términos lambda F y G tales que para tódo término X

$$\boldsymbol{\lambda} \vdash F\,X = G\,X$$

Pero no $\boldsymbol{\lambda} \vdash F = G$. Por ejemplo, $F \equiv y$ y $G \equiv \lambda x.y\,x$

Cuando se plantea formalizar un cálculo lambda que sea extensional, surge la pregunta, ¿Qué es demostrable en el sistema extensional que no es demostrable en $\boldsymbol{\lambda}$. A continuación se presentan tres diferentes agregados a la teoría $\boldsymbol{\lambda}$ las cuales incluyen la propiedad de extensionalidad y que han sido propuestas en la literatura [8, 2]. Las teorías extendidas son llamadas $\boldsymbol{\lambda\zeta}$, $\boldsymbol{\lambda + ext}$ y $\boldsymbol{\lambda\eta}$ de acuerdo a la regla que se añade a la definición 2.2.1.

Definición 2.3.2 (Reglas de extensionalidad)**.** Cada una de las siguientes reglas nos permite añadir a $\boldsymbol{\lambda}$ la propiedad de extensionalidad.

Reglas de inferencia

$$\frac{M\,x = N\,x}{M = N} \qquad \text{si } x \notin \mathrm{FV}(M\,N) \qquad (\zeta)$$

$$\frac{M\,P = N\,P}{M = N} \qquad \forall P \in \Lambda \qquad (\text{ext})$$

Axiomas

$$\lambda x.M\,x = M \qquad \text{si } x \notin \mathrm{FV}(M) \qquad (\eta)$$

La regla (ζ) dice, de manera informal, que si M y N tienen el mismo efecto sobre un objeto no especificado x, entonces $M = N$. La regla (ext) tiene una infinidad de premisas, una por cada término lambda P, por lo tanto, las deducciones en donde se involucre esta regla serán árboles infinitos.

2.3.3. Equivalencia de teorías

Comparación entre teorías formales, énfasis en η contra ζ contra ext.

Ver [8], [1], [11].

Sea $\mathcal{T}$ una teoría formal, se considera extender $\mathcal{T}$ añadiendo una nueva regla $\mathcal{R}$. Es natural preguntarse primer si $\mathcal{R}$ es derivable en $\mathcal{T}$. Pero ¿qué significa exactamente que $\mathcal{R}$ sea derivable?

Definición 2.3.3 (Regla)**.** Una regla $\mathcal{R}$ de n premisas es un conjunto de secuencias $\mathcal{S}_0, \ldots, \mathcal{S}_{n-1}, \mathcal{S}$ de longitud $n+1$, donde $\mathcal{S}_i, \mathcal{S}$ son elementos de deducción. Un elemento de $\mathcal{R}$ se dice ser una *instancia* de $\mathcal{R}$. Una instancia se denota

$$\frac{\mathcal{S}_0 \quad \mathcal{S}_1 \quad \ldots \quad \mathcal{S}_{n-1}}{\mathcal{S}}$$

$\mathcal{S}$ es la *conclusión* y las $\mathcal{S}_i$ son las *premisas*. Un *axioma* es una regla de cero premisas.

Instancias de axiomas aparecen en árboles de deducción simplemente como vértices superiores o de manera equivalente como elementos de deducción con una línea sobre ellos:

$$\overline{S}$$

Definición 2.3.4 (Reglas derivables y admisibles)**.** Sea $\mathscr{R}$ una regla determinada por una función $\phi\colon \mathscr{F}^n \to \mathscr{F}$. Se dice que $\mathscr{R}$ es *derivable* en $\mathscr{T}$ si y solo si, por cada instancia de $\mathscr{R}$, su conclusión es deducible en $\mathscr{T}$ a partir de sus premisas, es decir

$$\mathscr{T},\ A_1, ..., A_n \vdash B$$

.

Se dice que $\mathscr{R}$ es *admisible* en $\mathscr{T}$ si y solo si, añadir $\mathscr{R}$ a $\mathscr{T}$ como una nueva regla no va a incrementar la cantidad de teoremas de $\mathscr{T}$.

Se dice que $\mathscr{R}$ es *correcta* en $\mathscr{T}$ si y solo si, por cada instancia de $\mathscr{R}$, si todas las premisas son demostrables en $\mathscr{T}$, entonces también es demostrable la conclusión, es decir, si y solo si

$$(\mathscr{T} \vdash A_1), ..., (\mathscr{T} \vdash A_n) \implies (\mathscr{T} \vdash B)$$

Finalmente, una sola fórmula C, como un nuevo axioma propuesto, se dice ser tanto *derivable* como *admisible* en $\mathscr{T}$ si y solo si

$$\mathscr{T} \vdash C$$

2.4. Teoría de reducción

1.14 y 2.9

2.4.1. Contracciones

Transformaciones de términos con un paso.

Definición 2.4.1 (Contracciones Hindley y Seldin)**.** Dado un término lambda X, una *contracción* en X es una tripleta $\langle X, R, Y\rangle$, denotada $X \to_{\mathrm{R}} Y$, donde R es una aparición de un *redex* en X y Y es el resultado de contraer R en X.

Ejemplo 2.4.1.

$$(\lambda x.(\lambda y.y\ x)z)v \to_{(\lambda \mathrm{x}.(\lambda \mathrm{y.y\ x})\mathrm{z})\mathrm{v}} (\lambda y.y\ v)z,$$
$$(\lambda x.(\lambda y.y\ x)z)v \to_{(\lambda \mathrm{y.y\ x})\mathrm{z}} (\lambda x.z\ x)v.$$

2.4.2. Reducciones

Reducciones basadas en contracciones, de Barendregt

Definición 2.4.2 (Relación compatible)**.** Si dice que una relación binaria $\boldsymbol{R}$ sobre Λ es:

1. Una *relación compatible* cuando

$$(M, M') \in \boldsymbol{R} \implies (Z\,M, Z\,M') \in \boldsymbol{R},\ (M\,Z, M'\,Z) \in \boldsymbol{R},\ (\lambda x.M, \lambda x.M') \in \boldsymbol{R}$$

   para toda $M, M', Z \in \Lambda$.

2. Una *relación de congruencia* cuando $\boldsymbol{R}$ es compatible, reflexiva, transitiva y simétrica.

3. Una *relación de reducción* cuando $\boldsymbol{R}$ es compatible, reflexiva y transitiva.

compatible cuando

Nota. Una relación $\boldsymbol{R} \subseteq \Lambda^2$ es compatible cuando

$$(M, M') \in \boldsymbol{R} \implies (C[M], C[M']) \in \boldsymbol{R}$$

para toda $M, M' \in \Lambda$ y todo contexto $C[\quad]$, con un hoyo.

Definición 2.4.3. Una *noción de reducción* en Λ es una relación binaria $\boldsymbol{R}$ en Λ.

Sean $\boldsymbol{R}_1$ y $\boldsymbol{R}_2$ nociones de reducción, la relación $\boldsymbol{R}_1 \cup \boldsymbol{R}_2$ se denota $\boldsymbol{R}_1\boldsymbol{R}_2$.

Definición 2.4.4 (Reducción $\boldsymbol{\beta}$)**.** La regla (β) en la teoría $\boldsymbol{\lambda}$ se puede definir como la reducción:

$$\boldsymbol{\beta} = \{((\lambda x.M)N, M[x := N]) : M, N \in \Lambda\}$$

Definición 2.4.5. Sea $\boldsymbol{R}$ una noción de reducción en Λ, $\boldsymbol{R}$ introduce las relaciones binarias:

- R-reducción en un paso, denotada $\to_{\mathrm{R}}$ y definida de manera inductiva como:

$$(1) \qquad (M, N) \in \boldsymbol{R} \implies M \to_{\mathrm{R}} N$$

$$(2) \qquad M \to_{\mathrm{R}} N \implies Z\,M \to_{\mathrm{R}} Z\,N$$

$$(3) \qquad M \to_{\mathrm{R}} N \implies M\,Z \to_{\mathrm{R}} N\,Z$$

$$(4) \qquad M \to_{\mathrm{R}} N \implies \lambda x.M \to_{\mathrm{R}} \lambda x.N$$

- R-reducción, denotada $\twoheadrightarrow_{\mathrm{R}}$ y definida de manera inductiva como:

$$(1) \qquad M \to_{\mathrm{R}} N \implies M \twoheadrightarrow_{\mathrm{R}} N$$

$$(2) \qquad M \twoheadrightarrow_{\mathrm{R}} M$$

$$(3) \qquad M \twoheadrightarrow_{\mathrm{R}} N,\ N \twoheadrightarrow_{\mathrm{R}} L \implies M \twoheadrightarrow_{\mathrm{R}} L$$

- R-convertibilidad, denotada $=_R$ y definida de manera inductiva como:

$$(1) \qquad M \twoheadrightarrow_R N \implies M =_R N$$

$$(2) \qquad M =_R N \implies N =_R M$$

$$(3) \qquad M =_R N,\ N =_R L \implies M =_R L$$

Lema 2.4.1. *Las relaciones* $\rightarrow_R$, $\twoheadrightarrow_R$ *y* $=_R$ *son compatibles. Por lo tanto* $\twoheadrightarrow_R$ *es una relación de reducción y* $=_R$ *es una relación de congruencia.*

Demostración. □

Usualmente una noción de reducción se introduce de la siguiente manera: "Sea $\boldsymbol{R}$ definida por las siguientes *reglas de contracción* $\boldsymbol{R}$: $M \rightarrow N$ dado que ...".

Esto significa que $\boldsymbol{R} = \{(M, N) : ...\}$, por ejemplo, $\boldsymbol{\beta}$ pudo haber sido introducida por la siguiente regla de contracción

$$\boldsymbol{\beta} : (\lambda x.M)N \rightarrow M[x := N]$$

Definición 2.4.6. 1. Un R-redex es un término M tal que $(M, N) \in R$ para algún término N. En este caso N es llamado un R-contractum de M.

2. Un término M es llamado una forma normal de R, denotado $R - fn$, si M no contiene algún R-redex.

3. Un término N es una $R - fn$ de M (o M tiene la $R - fn$ N) si N es una $R - fn$ y $M =_R N$.

El proceso de pasar de un redex a su contractum es llamado *contracción*. En lugar de escribir "M es una $R - fn$" usualmente se escribe "M está en $R - fn$", pensando en una máquina que ha llegado a su estado final.

Ejemplo 2.4.2. $(\lambda x.x\,x)\lambda y.y$ es un β-redex. Por lo tanto $(\lambda x.x\,x)(\lambda y.y)z$ no está en $\beta - fn$; sin embargo este término tiene la $\beta - fn\ z$.

Plantear las ideas para pasar del concepto de contracción al de reducción y posteriormente al de convertibilidad.

Gráficas de reducción.

Teorema de Church-Rosser y toda la magia necesaria para abordarlo y la magia que valga la pena mencionar en la que CR es importante va aquí.

Probablemente es buena idea separar la sección "Noción informal del cálculo lambda" y "Formalización del cálculo lambda" en dos capítulos diferentes. ¿Valdrá la pena abordar árboles de Böhm?

Capítulo 3

Codificación de objetos

3.1. Álgebra Booleana

El álgebra booleana es una rama del álgebra en donde las expresiones tienen asociado un valor de *falso* o *verdadero*. Estas expresiones son fundamentales en el estudio de circuitos y programas escritos en lenguajes de programación.

Los términos lambda no tienen asignados un valor de verdad y las operaciones que se plantearon en los primeros dos capítulos involucraron el concepto de falso y verdadero únicamente en el metalenguaje y asociando estos valores no a los términos lambda en sí, si no a propiedades de estos, por ejemplo, es falso que $\|\lambda x.x\| = 5$ y es verdadero que $(\boldsymbol{K}\, x) \twoheadrightarrow_\beta (\lambda x.y)$. Sin embargo es posible codificar los valores de verdad como elementos de Λ y construir abstracciones que emulen las propiedades de las operaciones booleanas bajo la β-reducción. De esta manera se pueden escribir términos que, de acuerdo con la codificación establecida, representen expresiones booleanas y términos lambda al mismo tiempo.

En los lenguajes de programación usualmente se mezclan las expresiones booleanas con otras expresiones y objetos a partir de *predicados*, éstos son funciones con algún dominio X y codominio $\{\text{falso},\ \text{verdadero}\}$. Por ejemplo, al escribir un programa en

donde se necesite tomar una desición a partir de si un número n es positivo o negativo se escribiría (en pseudocódigo):

si esPositivo(n) **entonces**

...

de lo contrario

...

fin si

En este ejemplo «esPositivo» es un predicado que es evaluado a falso si `n` no es positivo y a verdadero si lo es.

La codificación de valores de verdad y operaciones booleanas es común incluso en lenguajes de programación populares, por ejemplo en C, el tipo `bool` es codificado como un entero, en donde falso es 0 y verdadero cualquier otro entero, a su vez, los enteros son codificados usualmente como secuencias de 32 bits en complemento a dos. Por lo tanto, si «esPositivo» fuera una función de C: «esPositivo(8)» sería evaluado a 1 y «esPositivo(-8)» sería evaluado a 0.

Al igual que el cálculo lambda, otras teorías que fundamentan las ciencias de la computación también carecen de expresiones y operaciones booleanas. En el caso de la máquina de Turing los cambios de estado en la ejecución de un programa se determinan a partir de su función de transición y predicados simples de igualdad entre símbolos del alfabeto de cinta se realizan en un paso, sin embargo, predicados mas complejos requieren ser codificados con estados, transiciones y anotaciones en su cinta.

3.1.1. Valores de verdad

En el álgebra booleana, los valores de las expresiones son falso y verdadero. El nombre de estos valores no es de relevancia y usualmente falso se representa como 0 y verdadero como 1. El aspecto importante de estos valores es que son distintos y si un valor x no es uno, entonces es el otro.

Podemos ignorar la representación concreta de estos valores y pensar en una situación hipotética: Una persona omnisciente y muda llamada P puede decirme si una oración que le digo es falsa o verdadera dándole una manzana y una pera; si me regresa la manzana significa que la oración es verdadera y si me regresa la pera significa que la oración es falsa. En este planteamiento irreal e hipotético, no fué necesario conocer la estructura de la verdad y la falsedad, solo fué necesario tener a alguien que tomara una desición (en este caso P) y proveer dos objetos que podemos distinguir entre sí (en este caso la manzana y la pera). Las desiciones de esta persona pueden ser los conceptos de falso y verdadero si nunca podemos conocer los valores booleanos.

Detrás del concepto de falso y verdadero, está el concepto de *desición*, la codificación que se desarrolla está basada en este concepto y aparece en [2, p. 133].

Supongamos que P es un término lambda el cual puede ser aplicado a una oración O, al β-reducir $(P\ O)$ se obtiene una decisión D la cual al ser aplicada a dos términos lambda M y N se β-reduce a M si la oración O es verdadera y a M si es falsa:

$$P\ O \twoheadrightarrow_\beta D,$$

$$D\ M\ N \twoheadrightarrow_\beta \begin{cases} M & \text{si } O \text{ es verdadera} \\ N & \text{si } O \text{ es falsa} \end{cases}.$$

Para fines prácticos no es necesario saber cómo es P ni O, lo importante es que cuando O es cierta, D eligirá M y si O es falsa, eligirá N. Por lo tanto, $(P\ O) = D$ es un término lambda de la forma

$$\lambda x\ y.Q$$

Si D es una desición tomada por que O es verdadera, podemos asegurar que $(D\ M\ N) = M$, por lo tanto:

$$D \equiv \lambda x\ y.x$$

Si D es una desición tomada por que O es falsa, podemos asegurar que $(D\ M\ N) = N$, por lo tanto:

$$D \equiv \lambda x\ y.y$$

Teniendo los términos lambda que representan la desición de P ante una oración falsa y ante una oración verdadera, se puede considerar que estos términos representan el concepto de falso y verdadero.

Definición 3.1.1 (Valores de verdad)**.** El concepto de falso y verdadero es codificado en el cálculo lambda como los términos $\boldsymbol{T}$ y $\boldsymbol{F}$ respectivamente.

$$\boldsymbol{T} \equiv \lambda x\ y.x \qquad\qquad \boldsymbol{F} \equiv \lambda x\ y.y$$

Utilizar $\boldsymbol{T}$ y $\boldsymbol{F}$ en términos lambda es similar a imitar a P y determinar cuando O es verdadera o falsa. Esto es debido a que se pueden plantear predicados que sean conceptualmente ilógicos, por ejemplo, si «esPositivo» se define de tal manera que sin importar en que valor sea evaluado siempre resulte en falso, los programas que se escriban no van a funcionar suponiendo que «esPositivo» calcula lo que debe de calcular, sin embargo lo importante de codificar el álgebra booleana es poder manipular los valores de falso y verdadero, no representar un término P que determine verdades absolutas.

3.1.2. Expresiones booleanas

Las expresiones booleanas se conforman de operaciones y valores de verdad. Las operaciones más básicas son la conjunción, la disyunción y la negación, también llama-

das AND, OR, NOT y denotadas $\wedge$, $\vee$ y $\neg$ respectivamente.

La conjunción y la disyunción son operaciones binarias definidas en

$$\{\text{falso},\ \text{verdadero}\}^2 \rightarrow \{\text{falso},\ \text{verdadero}\}$$

y la negación es una operación unaria definida en

$$\{\text{falso},\ \text{verdadero}\} \rightarrow \{\text{falso},\ \text{verdadero}\}.$$

Las tablas de verdad en el cuadro 3.1 establecen los resultados de estas tres operaciones para cada valor en su dominio.

x	y	$x \wedge y$	$x \vee y$
falso	falso	falso	falso
falso	verdadero	falso	verdadero
verdadero	falso	falso	verdadero
verdadero	verdadero	verdadero	verdadero

x	$\neg x$
falso	verdadero
verdadero	falso

Cuadro 3.1: Tablas de verdad para $\wedge$, $\vee$ y $\neg$

En el álgebra booleana, las expresiones se escriben en notación de infijo, utilizan paréntesis para agrupar expresiones y cuando los paréntesis son omitidos la negación tiene mayor presedencia que la conjunción y la conjunción tiene mayor presedencia que la disyunción, por ejemplo:

$$\text{verdadero} \wedge \text{falso} \vee \neg\text{falso}$$

$$\neg(\text{falso} \vee \text{falso})$$

$$\text{verdadero} \wedge (\text{falso} \vee \text{falso})$$

Esta notación es conveniente para escribir expresiones booleanas de manera concisa, pero es únicamente una conveniencia sintáctica del álgebra booleana. La codificación que se desarrolla de las operaciones seguirá las convenciones sintácticas del cálculo

lambda, por ejemplo, suponiendo que $\wedge$, $\vee$, $\neg$ son términos lambda, las expresiones mencionadas escribirían con notación de prefijo:

$$\vee(\wedge\, T\, F)F$$

$$\neg(\vee\, F\, F)$$

$$\wedge\, T(\vee\, F\, F)$$

Al igual que los valores de verdad, las operaciones básicas son codificadas como abstracciones del cálculo lambda. Hay varias metodologías para derivar términos lambda para las operaciones booleanas a partir de T y F, en esta sección se abordarán dos:

- Combinando valores de verdad
- Programando las operaciones

La primer metodología parte de la observación de que la codificación de falso y verdadero son abstracciones, por lo tanto, es posible β-reducirlas al aplicarlas a otros términos; se explora la clase de términos lambda en $\{T,\ F\}^+$.

La segunda metodología presenta la construcción del operador condicional, a partir del cual se derivan las operaciones booleanas como si fueran programas de computadora.

Combinaciones de valores de verdad

Una manera de obtener términos lambda a partir de F y T es β-reducir combinaciones de aplicaciones entre estos valores. En el cuadro 3.2 se muestran los términos obtenidos al reducir combinaciones de dos valores de verdad.

En las reducciones de 3.2 se pueden observar cuatro términos, a partir de estos se puede descubrir la operación de negación:

$\boldsymbol{F}\,\boldsymbol{F}$	$(\lambda x\, y.y)\boldsymbol{F} \twoheadrightarrow_\beta \lambda y.y \equiv \boldsymbol{I}$
$\boldsymbol{F}\,\boldsymbol{T}$	$(\lambda x\, y.y)\boldsymbol{T} \twoheadrightarrow_\beta \lambda y.y \equiv \boldsymbol{I}$
$\boldsymbol{T}\,\boldsymbol{F}$	$(\lambda x\, y.x)\boldsymbol{F} \twoheadrightarrow_\beta \lambda y.\boldsymbol{F} \equiv \boldsymbol{K}\,\boldsymbol{F}$
$\boldsymbol{T}\,\boldsymbol{T}$	$(\lambda x\, y.x)\boldsymbol{T} \twoheadrightarrow_\beta \lambda y.\boldsymbol{T} \equiv \boldsymbol{K}\,\boldsymbol{T}$

Cuadro 3.2: Posibles combinaciones de valores de verdad por pares.

- $(\boldsymbol{F}\,\boldsymbol{F})$ se reduce a la abstracción identidad, esto significa que para cualquier término $M \in \Lambda$

$$\boldsymbol{\lambda} \vdash (\boldsymbol{F}\,\boldsymbol{F}\,M) = M$$

- Al igual que la primer reducción $(\boldsymbol{F}\,\boldsymbol{T})$ se reduce a $\boldsymbol{I}$, por lo tanto se concluye que para cualesquiera términos $M \in \Lambda$, $N \in \{\boldsymbol{F},\ \boldsymbol{T}\}$

$$\boldsymbol{\lambda} \vdash (\boldsymbol{F}\,N\,M) = M$$

- $(\boldsymbol{T}\,\boldsymbol{F})$ se reduce a la abstracción constante de $\boldsymbol{F}$, esto significa que para cualquier término $M \in \Lambda$

$$\boldsymbol{\lambda} \vdash (\boldsymbol{T}\,\boldsymbol{F}\,M) = \boldsymbol{F}$$

- Al igual que la tercer reducción $(\boldsymbol{T}\,\boldsymbol{T})$ se reduce a $(\boldsymbol{K}\,\boldsymbol{T})$, por lo tanto se concluye que para cualesquiera términos $M \in \Lambda$, $N \in \{\boldsymbol{F},\ \boldsymbol{T}\}$

$$\boldsymbol{\lambda} \vdash (\boldsymbol{T}\,N\,M) = N$$

Debido a las reducciones mostradas en el cuadro 3.2 se puede analizar que a partir de un témino $\boldsymbol{F}$, se puede obtener $\boldsymbol{T}$ al reducir $(\boldsymbol{F}\,N\,\boldsymbol{T})$ y que a partir de un término $\boldsymbol{T}$, se puede obtener $\boldsymbol{F}$ al reducir $(\boldsymbol{T}\,\boldsymbol{F}\,M)$. Considerando que $N \equiv \boldsymbol{F}$ y $M \equiv \boldsymbol{T}$ las reducciones serían:

$$\boldsymbol{F}\,\boldsymbol{F}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{T}$$

$$\boldsymbol{T}\,\boldsymbol{F}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{F}$$

Si se considera que $P \in \{\boldsymbol{F},\ \boldsymbol{T}\}$

$$P\,\boldsymbol{F}\,\boldsymbol{T} \twoheadrightarrow_\beta \neg\, P$$

Observación (Sobre la β-reducción). En el tratamiento de la codificación del álgebra booleana en el cálculo lambda, cuando se β-reducen términos lambda que tienen como subtérminos valores que suponemos son $\boldsymbol{F}$ o $\boldsymbol{T}$ se extiende la teoría $\boldsymbol{\lambda}$ con la siguiente ecuación:

$$P\,\boldsymbol{T}\,\boldsymbol{F} = P \qquad\qquad \text{si } P \in \{\boldsymbol{F},\ \boldsymbol{T}\}$$

Definición 3.1.2 (Operación de negación)**.** El término lambda $\neg \equiv (\lambda p.p\,\boldsymbol{F}\,\boldsymbol{T})$ se reduce a $\boldsymbol{T}$ cuando es aplicado a $\boldsymbol{F}$ y viceversa

$$\begin{aligned}\neg\,\boldsymbol{F} &\equiv (\lambda p.p\,\boldsymbol{F}\,\boldsymbol{T})\boldsymbol{F}\\ &\to_\beta \boldsymbol{F}\,\boldsymbol{F}\,\boldsymbol{T}\\ &\twoheadrightarrow_\beta \boldsymbol{T}\end{aligned}$$

$$\begin{aligned}\neg\,\boldsymbol{T} &\equiv (\lambda p.p\,\boldsymbol{F}\,\boldsymbol{T})\boldsymbol{T}\\ &\to_\beta \boldsymbol{T}\,\boldsymbol{F}\,\boldsymbol{T}\\ &\twoheadrightarrow_\beta \boldsymbol{F}\end{aligned}$$

Las reducciones del cuadro 3.2 se pueden aplicar a $\boldsymbol{F}$ y $\boldsymbol{T}$ para obtener todas las posibles combinaciones de aplicaciones de valores de verdad de la forma $((P\,M)N)$, en

el cuadro 3.3 se muestran las reducciones de las nuevas aplicaciones.

$\boldsymbol{F}\,\boldsymbol{F}\,\boldsymbol{F}$	$\boldsymbol{I}\,\boldsymbol{F} \twoheadrightarrow_\beta \boldsymbol{F}$
$\boldsymbol{F}\,\boldsymbol{F}\,\boldsymbol{T}$	$\boldsymbol{I}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{T}$
$\boldsymbol{F}\,\boldsymbol{T}\,\boldsymbol{F}$	$\boldsymbol{I}\,\boldsymbol{F} \twoheadrightarrow_\beta \boldsymbol{F}$
$\boldsymbol{F}\,\boldsymbol{T}\,\boldsymbol{T}$	$\boldsymbol{I}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{T}$
$\boldsymbol{T}\,\boldsymbol{F}\,\boldsymbol{F}$	$\boldsymbol{K}\,\boldsymbol{F}\,\boldsymbol{F} \twoheadrightarrow_\beta \boldsymbol{F}$
$\boldsymbol{T}\,\boldsymbol{F}\,\boldsymbol{T}$	$\boldsymbol{K}\,\boldsymbol{F}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{F}$
$\boldsymbol{T}\,\boldsymbol{T}\,\boldsymbol{F}$	$\boldsymbol{K}\,\boldsymbol{T}\,\boldsymbol{F} \twoheadrightarrow_\beta \boldsymbol{T}$
$\boldsymbol{T}\,\boldsymbol{T}\,\boldsymbol{T}$	$\boldsymbol{K}\,\boldsymbol{T}\,\boldsymbol{T} \twoheadrightarrow_\beta \boldsymbol{T}$

Cuadro 3.3: Posibles combinaciones de valores de verdad con asociación a la izquierda.

Al observar el cuadro 3.3, se distinguen algúnos patrones en los resultados de las reducciones, por ejemplo, si P es un valor de verdad cualquiera, $(P\,\boldsymbol{F}\,\boldsymbol{F})$ se reduce a $\boldsymbol{F}$ y $(P\,\boldsymbol{T}\,\boldsymbol{T})$ se reduce a $\boldsymbol{T}$, las combinaciones mas interesantes se presentan en los renglones 2, 3, 6 y 7.

En búsqueda de las operaciones binarias de conjunción y disyunción se desarrollan tablas de verdad con las posibles combinaciones de dos términos $P, Q \in \{\boldsymbol{F},\ \boldsymbol{T}\}$. La cantidad de combinaciones de estos valores es $2 \times \binom{3}{2} = 2 \times \frac{3!}{2!} = 6$ y son $(P\,Q\,\boldsymbol{F})$, $(P\,Q\,\boldsymbol{T})$, $(P\,\boldsymbol{F}\,Q)$, $(P\,\boldsymbol{T}\,Q)$, $(\boldsymbol{F}\,P\,Q)$, $(\boldsymbol{T}\,P\,Q)$. Las tablas de verdad de estas combinaciones intercambiando las posiciones de P y Q serían las mismas ya que ambos términos toman los valores de falso y verdadero en las tablas de verdad. En el cuadro 3.4 se muestran estas tablas.

P	Q	$P\,Q\,\boldsymbol{F}$	$P\,Q\,\boldsymbol{T}$	$P\,\boldsymbol{F}\,Q$	$P\,\boldsymbol{T}\,Q$	$\boldsymbol{F}\,P\,Q$	$\boldsymbol{T}\,P\,Q$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$
$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$

Cuadro 3.4: Tablas de verdad considerando dos variables P y Q en aplicaciones de tres términos

Las columnas de la combinación $(P\,Q\,\boldsymbol{F})$ y $(P\,\boldsymbol{T}\,Q)$ del cuadro 3.4 corresponden a la operación de conjunción y disyunción respectivamente, como aparecen en el cuadro 3.1. Las otras combinaciones corresponden a operaciones no básicas del álgebra booleana:

$(P\ Q\ \boldsymbol{T})$ es la implicación material; $(P\ \boldsymbol{F}\ Q)$ es la no implicación inversa; $(\boldsymbol{F}\ P\ Q)$ es la proyección de Q; y $(\boldsymbol{T}\ P\ Q)$ es la proyección de P.

Ya que $\boldsymbol{\lambda} \vdash (P\ Q\ \boldsymbol{F}) = (\wedge\ P\ Q)$ y $\boldsymbol{\lambda} \vdash (P\ \boldsymbol{T}\ Q) = (\vee\ P\ Q)$, se construyen los términos $\wedge$ y $\vee$ abstrayendo a P y Q de las igualdades.

Definición 3.1.3 (Operación de conjunción)**.** El término lambda que representa la conjunción es

$$\wedge \equiv \lambda p\ q.p\ q\ \boldsymbol{F}$$

y cumple las siguientes propiedades de β-reducción al ser aplicada a valores de verdad:

$$\begin{aligned} \wedge\ \boldsymbol{F}\ \boldsymbol{F} &\equiv (\lambda p\ q.p\ q\ \boldsymbol{F})\boldsymbol{F}\ \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F}\ \boldsymbol{F}\ \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F} \end{aligned}$$

$$\begin{aligned} \wedge\ \boldsymbol{F}\ \boldsymbol{T} &\equiv (\lambda p\ q.p\ q\ \boldsymbol{F})\boldsymbol{F}\ \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{F}\ \boldsymbol{T}\ \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F} \end{aligned}$$

$$\begin{aligned} \wedge\, \boldsymbol{T}\, \boldsymbol{F} &\equiv (\lambda p\, q.p\, q\, \boldsymbol{F})\boldsymbol{T}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{T}\, \boldsymbol{F}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F} \end{aligned}$$

$$\begin{aligned} \wedge\, \boldsymbol{T}\, \boldsymbol{T} &\equiv (\lambda p\, q.p\, q\, \boldsymbol{F})\boldsymbol{T}\, \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{T}\, \boldsymbol{T}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{T} \end{aligned}$$

Definición 3.1.4 (Operación de disyunción)**.** El término lambda que representa la disyunción es

$$\vee \equiv \lambda p\, q.p\, \boldsymbol{T}\, q$$

y cumple las siguientes propiedades de β-reducción al ser aplicada a valores de verdad:

$$\begin{aligned} \vee\, \boldsymbol{F}\, \boldsymbol{F} &\equiv (\lambda p\, q.p\, \boldsymbol{T}\, q)\boldsymbol{F}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F}\, \boldsymbol{T}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{F} \end{aligned}$$

$$\begin{aligned} \vee\, \boldsymbol{F}\, \boldsymbol{T} &\equiv (\lambda p\, q.p\, \boldsymbol{T}\, q)\boldsymbol{F}\, \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{F}\, \boldsymbol{T}\, \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{T} \end{aligned}$$

$$\begin{aligned} \vee\, \boldsymbol{T}\, \boldsymbol{F} &\equiv (\lambda p\, q.p\, \boldsymbol{T}\, q)\boldsymbol{T}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{T}\, \boldsymbol{T}\, \boldsymbol{F} \\ &\twoheadrightarrow_\beta \boldsymbol{T} \end{aligned}$$

$$\begin{aligned} \vee\, \boldsymbol{T}\, \boldsymbol{T} &\equiv (\lambda p\, q.p\, \boldsymbol{T}\, q)\boldsymbol{T}\, \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{T}\, \boldsymbol{T}\, \boldsymbol{T} \\ &\twoheadrightarrow_\beta \boldsymbol{T} \end{aligned}$$

Esta metodología para encontrar operaciones del álgebra booleana, aplicando los términos codificados de los valores de verdad, es tediosa pero hasta cierto grado efectiva. Como en los casos de las operaciones no básicas mostradas en el cuadro 3.4, operaciones del álgebra booleana pueden ser "descubiertas" y no construídas. Ya que la negación, la conjunción y la disyunción fueron descubiertas con este método, cualquier operación booleana eventualmente será encontrada como combinación de valores de verdad. Sin embargo, descubrir la codificación de una operación booleana complicada utilizando este método es un proceso muy tardado.

Programación de operaciones booleanas

Otra metodología que permite construír las operaciones booleanas como términos lambda es la de partir de un algorítmo que las describa. Usualmente las operaciones booleanas no son definidas como procedimientos, si no como operaciones primitivas del lenguaje utilizado para describirlos.

Consideremos dos términos M y N. Ya que $(\boldsymbol{T}\, M\, N) \twoheadrightarrow_\beta M$ y $(\boldsymbol{F}\, M\, N) \twoheadrightarrow_\beta N$, si $M \twoheadrightarrow_\beta M'$ y $N \twoheadrightarrow_\beta N'$, entonces

$$\boldsymbol{T}\, M\, N \twoheadrightarrow_\beta M'$$

$$\boldsymbol{F}\, M\, N \twoheadrightarrow_\beta N'$$

Es decir, si $P \in \{\boldsymbol{F},\ \boldsymbol{T}\}$:

$$P\, M\, N \twoheadrightarrow_\beta \begin{cases} M' & P \equiv \boldsymbol{T} \\ N' & P \equiv \boldsymbol{F} \end{cases}$$

Esta aplicación de un valor de verdad a dos términos lambda cualquiera permite capturar el concepto de una expresión o sentencia condicional, usualmente llamada en los lenguajes de programación como sentencia `if-then-else`.

Definición 3.1.5 (Expresión condicional)**.** El término lambda que representa a la expresión condicional es

$$\prec \equiv \lambda p\, m\, n.p\, m\, n$$

Y si P es un valor de verdad, entonces

$$\prec P\ M\ N \equiv (\lambda p\ m\ n.p\ m\ n)P\ M\ N$$

$$\twoheadrightarrow_\beta P\ M\ N$$

Un programa de la forma

si P **entonces**

M

de lo contrario

M

fin si

Puede ser traducido a $(\prec\ P\ M\ N)$

Consideremos la siguiente definición en pseudocódigo de la operación de negación:

Algoritmo 3.1 Negación de p

Entrada: $p \in \{\text{falso, verdadero}\}$
Salida: $\neg p$
  si p **entonces**
    regresar falso
  de lo contrario
    regresar verdadero
  fin si

El pseudocódigo se traduce al cálculo lambda como

$$\neg \equiv \lambda p.\ \prec\ p\ \boldsymbol{F}\ \boldsymbol{T}$$

El cuerpo de la abstracción puede ser β-reducido para obtener el término de la definición 3.1.2

$$\lambda p.\prec p\,\boldsymbol{F}\,\boldsymbol{T} \equiv \lambda p.(\lambda p\,m\,n.p\,m\,n)p\,\boldsymbol{F}\,\boldsymbol{T}$$

$$\twoheadrightarrow_\beta \lambda p.p\,\boldsymbol{F}\,\boldsymbol{T}$$

Para la operación de conjunción, se considera el siguiente pseudocódigo:

Algoritmo 3.2 Conjunción de p_1 y p_2

Entrada: $p_1, p_2 \in \{\text{falso},\ \text{verdadero}\}$
Salida: $p_1 \wedge p_2$
  si p_1 **entonces**
    si p_2 **entonces**
      regresar verdadero
    de lo contrario
      regresar false
    fin si
  de lo contrario
    regresar falso
  fin si

Traducido al cálculo lambda como

$$\wedge \equiv \lambda p_1\,p_2.\prec p_1(\prec p_2\,\boldsymbol{T}\,\boldsymbol{F})\boldsymbol{F}$$

Al β-reducir el cuerpo de la abstracción se obtiene el término de la definición 3.1.3

$$\begin{aligned}\lambda p_1\,p_2.\prec p_1(\prec p_2\,\boldsymbol{T}\,\boldsymbol{F})\boldsymbol{F} &\equiv \lambda p_1\,p_2.(\lambda p\,m\,n.p\,m\,n)p_1((\lambda p\,m\,n.p\,m\,n)p_2\,\boldsymbol{T}\,\boldsymbol{F})\boldsymbol{F}\\ &\twoheadrightarrow_\beta \lambda p_1\,p_2.p_1((\lambda p\,m\,n.p\,m\,n)p_2\,\boldsymbol{T}\,\boldsymbol{F})\boldsymbol{F}\\ &\twoheadrightarrow_\beta \lambda p_1\,p_2.p_1(p_2\,\boldsymbol{T}\,\boldsymbol{F})\boldsymbol{F}\\ &=_\lambda \lambda p_1\,p_2.p_1\,p_2\,\boldsymbol{F}\end{aligned}$$

De igual manera, considerando el siguiente pseudocódigo de la operación de disyunción:

Algoritmo 3.3 Disyunción de p_1 y p_2

Entrada: $p_1, p_2 \in \{\text{falso}, \text{verdadero}\}$
Salida: $p_1 \vee p_2$

```
si p1 entonces
  regresar verdadero
de lo contrario
  si p2 entonces
    regresar verdadero
  de lo contrario
    regresar falso
  fin si
fin si
```

Se traduce al cálculo lambda como

$$\vee \equiv \lambda p_1\, p_2.\ \prec\ p_1\, \boldsymbol{T} (\prec\ p_2\, \boldsymbol{T}\, \boldsymbol{F})$$

Y al β-reducir el cuerpo de la abstracción se obtiene el término de la definición 3.1.4

$$\begin{aligned}
\lambda p_1\, p_2.\ \prec\ p_1\, \boldsymbol{T} (\prec\ p_2\, \boldsymbol{T}\, \boldsymbol{F}) &\equiv \lambda p_1\, p_2.(\lambda p\ m\ n.p\ m\ n) p_1\, \boldsymbol{T} ((\lambda p\ m\ n.p\ m\ n) p_2\, \boldsymbol{T}\, \boldsymbol{F}) \\
&\twoheadrightarrow_\beta \lambda p_1\, p_2.p_1\, \boldsymbol{T} ((\lambda p\ m\ n.p\ m\ n) p_2\, \boldsymbol{T}\, \boldsymbol{F}) \\
&\twoheadrightarrow_\beta \lambda p_1\, p_2.p_1\, \boldsymbol{T} (p_2\, \boldsymbol{T}\, \boldsymbol{F}) \\
&=_\lambda \lambda p_1\, p_2.p_1\, \boldsymbol{T}\, p_2
\end{aligned}$$

Utilizando esta técnica, se puede obtener el término lambda para una operación a partir del pseudocódigo basado en valores de verdad y la sentencia `if-then-else`. Teniendo estos resultados resulta natural, generalizar el pseudocódigo para construír un término lambda que a partir de la tabla de verdad de una operación booleana binaria, resulte en la abstracción que codifica la operación.

Definición 3.1.6 (Traducción de operaciones booleanas binarias)**.** Sea $\odot$ una operación booleana binaria con la siguiente tabla de verdad

P	Q	$P \odot Q$
F	***F***	x_1
F	***T***	x_2
T	***F***	x_3
T	***T***	x_4

El procedimiento generalizado es

Algoritmo 3.4 Operación booleana $\odot$ dado x_1, x_2, x_3, x_3

Entrada: $p_1, p_2 \in \{\text{falso, verdadero}\}$
Salida: Valor x_i de la tabla de verdad

```
si p1 entonces
  si p2 entonces
    regresar x4
  de lo contrario
    regresar x3
  fin si
de lo contrario
  si p2 entonces
    regresar x2
  de lo contrario
    regresar x1
  fin si
fin si
```

Y la traducción al cálculo lambda es

$$\lambda x_1\, x_2\, x_3\, x_4.(\lambda p_1\, p_2.(\prec\ p_1(\prec\ p_2\, x_4\, x_3)(\prec\ p_2\, x_2\, x_1)))$$

Ejemplo 3.1.1 (Operaciones *NAND* y *NOR*)**.** Estas operaciones booleanas binarias conforman los conjuntos unitarios {NAND} y {NOR} los cuales son conjuntos funcionalmene completos, es decir, únicamente con la operación NAND se puede emular cualquier operación booleana y únicamente con la operación NOR se puede emular cualquier operación booleana.

La operación *NAND* se denota $P \uparrow Q$ y tiene la siguiente tabla de verdad

P	Q	$P \uparrow Q$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$

Con el proceso de traducción mostrado en la definición 3.1.6, el término lambda $\uparrow$ que codifica la operación NAND sería

$$\begin{aligned}\uparrow &\equiv \lambda p_1\, p_2.\ \prec\ p_1(\prec\ p_2\, \boldsymbol{F}\, \boldsymbol{T})(\prec\ p_2\, \boldsymbol{T}\, \boldsymbol{T}) \\ &\twoheadrightarrow_\beta \lambda p_1\, p_2.p_1(p_2\, \boldsymbol{F}\, \boldsymbol{T})\boldsymbol{T}\end{aligned}$$

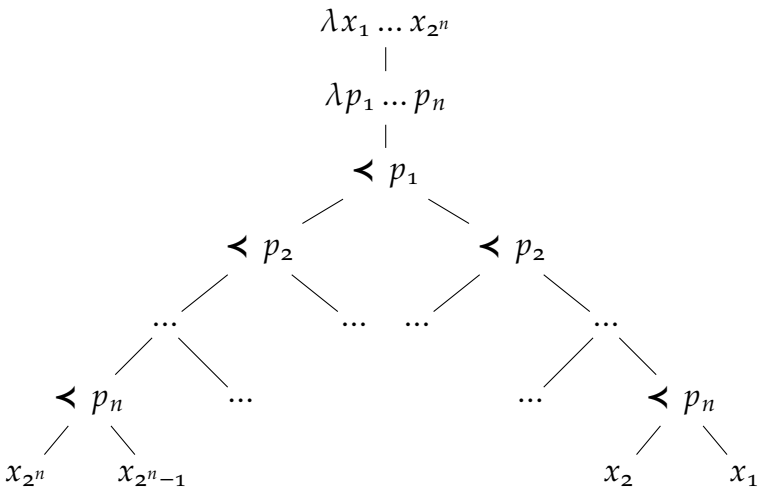
La operación *NOR* se denota $P \downarrow Q$ y tiene la siguiente tabla de verdad

P	Q	$P \downarrow Q$
$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{F}$
$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$

Usando el mismo proceso de traducción que con la operación NAND, se obtiene

$$\begin{aligned}\downarrow &\equiv \lambda p_1\, p_2.\ \prec\ p_1(\prec\ p_2\, \boldsymbol{F}\, \boldsymbol{F})(\prec\ p_2\, \boldsymbol{F}\, \boldsymbol{T}) \\ &\twoheadrightarrow_\beta \lambda p_1\, p_2.p_1\, \boldsymbol{F}(p_2\, \boldsymbol{F}\, \boldsymbol{T})\end{aligned}$$

Cuando se generaliza el método de traducción de 3.1.6 a operaciones booleanas n-árias, se obtiene un término bosquejado de la siguiente manera



3.1.3. Extensiones al álgebra booleana

Conociendo el proceso de codificación del álgebra booleana en el cálculo lambda, resulta simple adaptar la codificación.

Consideremos el caso en donde, además de tener los valores de falso y verdadero, se desea incorporar un valor "desconocido" utilizado para representar un valor que no es ni falso, ni verdadero. La interpretación de estos valores es similar a 3.1.1, pero en lugar de decidir sobre dos términos, se decide sobre tres.

Definición 3.1.7 (Valores de álgebra trivalente)**.** La codificación en términos lambda de los valores de ésta álgebra trivalente son

$$\boldsymbol{T} \equiv \lambda x\, y\, z.x$$

$$\boldsymbol{F} \equiv \lambda x\, y\, z.y$$

$$\boldsymbol{U} \equiv \lambda x\, y\, z.z$$

Al igual que en la codificación bivalente, se puede codificar un término $\prec_3$, similar a $\prec$ de la definición 3.1.5 pero con tres ramificaciones

Definición 3.1.8 (Condicional trivalente)**.**

$$\prec_3 \equiv \lambda p\, m\, n\, o.p\, m\, n\, o$$

De tal manera que, si $P \in \{\boldsymbol{T},\ \boldsymbol{F},\ \boldsymbol{U}\}$

$$(\prec_3\ P\, M\, N\, O) \twoheadrightarrow_\beta \begin{cases} M & P \equiv \boldsymbol{T}; \\ N & P \equiv \boldsymbol{F}; \\ O & P \equiv \boldsymbol{U}. \end{cases}$$

Sea $\odot$ una operación trivalente binaria con la siguiente tabla de valores

P	Q	$P \odot Q$
$\boldsymbol{T}$	$\boldsymbol{T}$	x_1
$\boldsymbol{T}$	$\boldsymbol{F}$	x_2
$\boldsymbol{T}$	$\boldsymbol{U}$	x_3
$\boldsymbol{F}$	$\boldsymbol{T}$	x_4
$\boldsymbol{F}$	$\boldsymbol{F}$	x_5
$\boldsymbol{F}$	$\boldsymbol{U}$	x_6
$\boldsymbol{U}$	$\boldsymbol{T}$	x_7
$\boldsymbol{U}$	$\boldsymbol{F}$	x_8
$\boldsymbol{U}$	$\boldsymbol{U}$	x_9

El procedimiento en pseudocódigo que la describe es:

Traducido al cálculo lambda como

$$\lambda x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7\, x_8\, x_9.(\lambda p_1\, p_2.(\prec_3\ p_1\, R_1\, R_2\, R_3))$$

Donde

Algoritmo 3.5 Operación booleana $\odot$ dado $x_1, x_2, \ldots, x_9$

Entrada: $p_1, p_2 \in \{\text{verdadero, falso, desconocido}\}$
Salida: Valor x_i de la tabla de verdad

```
si p1 = verdadero entonces
  si p2 = verdadero entonces
    regresar x1
  de lo contrario y si p2 = falso entonces
    regresar x2
  de lo contrario y si p2 = desconocido entonces
    regresar x3
  fin si
de lo contrario y si p1 = falso entonces
  si p2 = verdadero entonces
    regresar x4
  de lo contrario y si p2 = falso entonces
    regresar x5
  de lo contrario y si p2 = desconocido entonces
    regresar x6
  fin si
de lo contrario y si p1 = desconocido entonces
  si p2 = verdadero entonces
    regresar x7
  de lo contrario y si p2 = falso entonces
    regresar x8
  de lo contrario y si p2 = desconocido entonces
    regresar x9
  fin si
fin si
```

$$R_1 \equiv (\prec_3\ p_2\, x_1\, x_2\, x_3)$$

$$R_2 \equiv (\prec_3\ p_2\, x_4\, x_5\, x_6)$$

$$R_3 \equiv (\prec_3\ p_2\, x_7\, x_8\, x_9)$$

3.1.4. Lógica en el cálculo lambda

En esta sección se abordó la codificación de valores de verdad y operaciones como la conjunción, disyunción y negación. Usualmente en las ciencias de la computación y las matemáticas, estos objetos se abordan desde la perspectiva algebraica y desde la perspectiva lógica.

En el cálculo lambda se distinguen las dos perspectivas por el lenguaje en el que se manejan los objetos, en el caso del álgebra booleana, los valores de verdad y las operaciones booleanas se codifican como términos lambda y pueden ser combinados con otros términos lambda que no forman parte del álgebra booleana. Por otra parte, la lógica proposicional estudia proposiciones, las cuales son enunciados matemáticos a los cuales se les puede atribuír el valor de falso o verdadero y que, a partir de conectivos lógicos como $\neg$, $\wedge$ y $\vee$, se infieren verdades sobre los enunciados.

El cálculo lambda se puede extender de tal manera que nos permita inferir verdades sobre sus términos, en esta subsección se abordan los cálculos lambda aumentados con nociones *ilativas*, denotados $i\boldsymbol{\lambda}$. El tratamiento de esta extensión está basado en el apéndice B "Lógica combinatoria ilativa" de [2, pp. 573–576].

Definición 3.1.9 (Términos $i\boldsymbol{\lambda}$)**.** Los términos de la teoría $i\boldsymbol{\lambda}$, denotados $i\Lambda$ se definen sobre el alfabeto de Λ extendido con un conjunto C de *constantes lógicas*.

$$M \in \Lambda \implies M \in i\Lambda$$

$$c \in C \implies c \in i\Lambda$$

$$M, N \in i\Lambda \implies (M\,N) \in i\Lambda$$

Definición 3.1.10 (Fórmulas en $i\boldsymbol{\lambda}$)**.** Las fórmulas de la teoría $i\boldsymbol{\lambda}$ se definen de la siguiente manera

- Si $M, N \in i\Lambda$, entonces $M = N$ es una fórmula.
- Si $M \in i\Lambda$, entonces M es una fórmula.

En el estudio de teorías ilativas, la interpretación de $i\boldsymbol{\lambda} \vdash M$ y $i\boldsymbol{\lambda} \vdash M = N$ es "La fórmula M es verdadera" y "La fórmula $M = N$ es verdadera" respectivamente.

Teoría $i\boldsymbol{\lambda}_{KR}$

Constantes $C = \{\boldsymbol{N}\}$ con la interpretación $\neg\, M$ es $(\boldsymbol{N}\, M)$

Paradoja de Kleene-Rosser

$$X \equiv \lambda x.\boldsymbol{N}(x\,x)$$

$$X\,X = (\lambda x.\boldsymbol{N}(x\,x))X = \boldsymbol{N}(X\,X)$$

Se puede derivar que una fórmula $X\,X = \neg(X\,X)$, es decir, en $i\boldsymbol{\lambda}_{KR}$ se pueden derivar contradicciones.

Teoría $i\boldsymbol{\lambda}_{\mathrm{o}}$

Constantes $C = \{\boldsymbol{F},\ \boldsymbol{Q},\ \boldsymbol{E},\ \boldsymbol{\Xi},\ \boldsymbol{\Pi},\ \boldsymbol{P}\}$ con las siguientes interpretaciones

- $M \in N$ es $N\,M$
- $M \implies N$ es $\boldsymbol{P}\,M\,N$
- $M \subseteq N$ es $\boldsymbol{\Xi}\,M\,N$
- N^M es $\boldsymbol{F}\,M\,N$
- $\forall x\ M$ es $\boldsymbol{\Xi}(\lambda x.M)$

Pendientes

- Definibilidad de $\boldsymbol{E}$, $\boldsymbol{F}$, $\boldsymbol{P}$ y $\boldsymbol{\Pi}$ a partir de $\boldsymbol{Q}$, $\boldsymbol{\Xi}$.
- Inconsistencia de $i\boldsymbol{\lambda}_0$ por la paradoja de Curry "Una teoría lambda ilativa es inconsistente si cada M puede ser derivada" (la paradoja de Kleene-Rosser es una consecuencia de la paradoja de Curry).
- Mencionar la técnica utilizada por Church para "escapar" de la paradoja de Curry y presentar el "tradeoff" de tener un sistema suficientemente poderoso: Equivalencia $\boldsymbol{\lambda}$ y máquina de Turing *vs.* separar la noción ilativa de la abstracción.

3.2. Aritmética

Así como se pueden representar los valores de verdad de falso y verdadero en el cálculo lambda, también podemos encontrar representaciones para los números naturales. En esta sección se aborda una representación llamada numerales de Church, también se presentan términos lambda para operar números naturales con esta representación.

Para cada $n \in \mathbb{N}$ el numeral de Church de n es un término lambda denotado como $\bar{n}$ definido como:

n	$\bar{n}$
0	$\lambda x\ y.y$
1	$\lambda x\ y.x\ y$
2	$\lambda x\ y.x(x\ y)$
3	$\lambda x\ y.x(x(x\ y))$
...	...

Cuadro 3.5: Numerales de Church

$$\bar{n} \equiv \lambda x\ y.x^n\ y$$

En la siguiente tabla se puede apreciar mejor la estructura de los numerales de Church

Como se observa en la tabla, un numeral de Church es una abstracción descurrificada de dos argumentos la cual al ser evaluada es reducida a la n-ésima composición del primer argumento evaluada en el segundo argumento.

Una manera de entender esta representación es pensar en los números naturales como un conteo de uno en uno; el 0 es no contar; el 1 es contar uno mas que el 0; el 2 es uno mas que el 1, así que el 2 es uno mas que el uno mas que el 0; el 3 es uno mas que el 2, así que el 3 es uno mas que el uno mas que el uno mas que el 0 y así sucesivamente. La idea de "el uno mas" es la del sucesor, si consideramos a x como una función sucesor y a y como el 0, podemos expresar $x(x(x(y)))$ como sucesor(sucesor(sucesor(0))), así que

$$\begin{aligned} \text{sucesor(sucesor(sucesor(0)))} &= \text{sucesor(sucesor(1))} \\ &= \text{sucesor(2)} \\ &= 3 \end{aligned}$$

Es interesante pensar en diferentes maneras de expresar las operaciones mas elementales de la aritmétima como términos lambda que operen sobre esta representación.

A continuación se presenta una exploración de los términos lambda correspondientes a algunas operaciones elementales de la aritmética: suma, multiplicación, exponenciación y resta. La suma es una repetición de la operación sucesor, la multiplicación una repetición de suma, la exponenciación una repetición de multiplicaciones y la resta una repetición de la operación predecesor. Esto nos lleva a identificar las operaciones de sucesor y predecesor como los algoritmos base para el resto de las operaciones primitivas.

El término *sucesor* debe ser uno tal que al ser aplicado a un numeral $\bar{n}$ se pueda β-reducir al numeral $n \dot{+} 1$. Considerando la definición de $\bar{n} \equiv \lambda x\ y.x^n\ y$, lo que buscamos es una manera de agregarle una x a la composición en el cuerpo de $\bar{n}$ para obtener $\lambda x\ y.x^{n+1}\ y$. Se construye este término considerando primero que será aplicado a un numeral

$$\text{sucesor} \equiv \lambda \bar{n}.?$$

Además el resultado de β-reducir esta aplicación deberá ser una función de dos argumentos (como lo son todos los numerales de Church):

$$\text{sucesor} \equiv \lambda \bar{n}.\lambda x\ y.?$$

Tomando en cuenta que $\bar{n}\ x\ y \equiv x^n\ y$ y que $x\ x^n y \equiv x^{n+1}\ y$:

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

A continuación se β-reduce la aplicación de sucesor al numeral $\bar{4} \equiv \lambda x\ y.x(x(x(x\ y)))$:

$$
\begin{aligned}
&\text{sucesor } \bar{4} \\
\equiv&(\lambda n\ x\ y.x(n\ x\ y))(\lambda x\ y.x(x(x(x\ y)))) \\
=_\alpha&(\lambda n\ x\ y.x(n\ x\ y))(\lambda f\ z.f(f(f(f\ z)))) \\
\to_\beta&(\lambda x\ y.x(((\lambda f\ z.f(f(f(f\ z))))x)y)) \\
\to_\beta&(\lambda x\ y.x((\lambda z.x(x(x(x\ z))))y)) \\
\to_\beta&(\lambda x\ y.x(x(x(x(x\ y))))) \\
\equiv&\bar{5}
\end{aligned}
$$

La otra operación elemental en la aritmética es el *predecesor*, el término que represente esta operación debe ser uno que cumpla con la siguiente definición:

$$\text{predecesor } \bar{0} \twoheadrightarrow_\beta \bar{0}$$

$$\text{predecesor } \bar{n} \twoheadrightarrow_\beta n \dot{-} 1$$

El término lambda del predecesor con la representación de numerales de Church es mucho mas compleja que la del sucesor. Se pudiera pensar que la misma idea utilizada en la derivación del sucesor aplicaría para la derivación del predecesor: si tenemos n aplicaciones de x a y, al aplicar el término que buscamos a un numeral de Church se debe β-reducir a otro numeral con una aplicación de x menos, se utiliza y para añadir una x mas en el cuerpo del numeral. Sin embargo, la estructura de los numerales no nos permite quitar una x usando y facilmente ya que el numeral puede ser aplicado a dos términos, el que representa las x y el que representa a la y; la variable que determina el valor del numeral es x y la sustitución de x por otro término en esta representación se hace con *cada* aparición de x en el cuerpo del numeral, por otro lado, al sustituír la

y por otro término solo podemos hacer mas complejo el término o sustituírla por otra variable.

Para derivar el término del predecesor vamos a presentar un término con una estructura similar a los numerales de Church:

$$\lambda x\ y\ z.z\ x^n\ y$$

La diferencia entre este término y un numeral de Church es que podemos modificar su estructura por enfrente, por atrás y en las composiciones intermedias. Si este término representara $n \dot{+} 1$ pudieramos obtener el cuerpo de $\bar{n}$ de la siguiente manera:

$$\begin{aligned}
&(\lambda x\ y\ z.z\ x^n\ y)x\ y(\lambda a.a)\\
\rightarrow_\beta&(((\lambda y\ z.z\ x^n\ y)y)(\lambda a.a))\\
\twoheadrightarrow_\beta&((\lambda z.z\ x^n\ y)(\lambda a.a))\\
\rightarrow_\beta&(\lambda a.a)x^n\ y\\
\rightarrow_\beta&x^n\ y
\end{aligned}$$

Es decir, mantenemos las x y la y y le aplicamos a $x^n\,y$ el término lambda que representa a la función identidad. Esta manera conveniente de representar a los numerales resulta ser incompleta, ya que no se podrá obtener $x^{n-1}\,y$ a partir del resultado (ya que la z no aparece en el término resultante). Sin embargo, si logramos tener un término que nos genera este término modificado pudiéramos realizar esta transformación dentro de la función predecesor sería mas fácil encontrar $n \dot{-} 1$.

La estructura del sucesor sería:

$$\lambda n\ x\ y.?\,(\lambda a.a)$$

Donde ? debe ser tal que al β-reducirse resulte en un término con la forma $\lambda z.z\ x^{n-1}\ y$. Es conveniente desmenuzar el problema de encontrar este término desconocido: primero sabemos que el numeral de Church $\bar{n}$ puede ser aplicado a dos términos y el primer término al que sea aplicado se sustituirá en todas las apariciones de x, como queremos que la β-reducción nos genere una función cuyo argumento sea la primer variable en la composición del numeral, tenemos que encontrar una manera de propagar un término de la forma $\lambda w.w\ x^m\ y$ de tal manera que al aplicarle otro término nos resulte $\lambda w.w\ x^{m+1}\ y$. De esta manera al aplicar este otro término una y otra vez, resulte $\lambda w.w\ x^{n-1}\ y$ con el cual podemos obtener el cuerpo del predecesor sustituyendo w por $\lambda a.a$.

Este otro término que buscamos será el que sustituirá a la x en $\bar{n}$ para que:

$$?(\lambda w.w\ x^m\ y) \twoheadrightarrow_\beta \lambda w.w\ x^{m+1}\ y$$

Lo que sucede en cada aplicación de este estilo es que se compone una x en cada aplicación y se deja explícita una w que podrá ser sustituída como valor. El término que nos permite hacer esto tiene la siguiente forma:

$$\lambda g\ w.w(g\ x)$$

Al ser aplicado a un término $\lambda w.w\ x^m\ y$ la variable g será sustituída por este término y el resultado será $\lambda w.w((\lambda r.r\ x^m\ y)x)$ (nótese el cambio de nombre de la variable ligada w en el argumento), lo cual se reduce a $\lambda w.w\ x^{m+1}\ y$ el cual mantiene su estructura original.

El percance con esta aproximación a la solución es que el primer valor al que se le aplica el término $\lambda g\ w.w(g\ x)$ debe ser $\lambda w.w\ y$.

Para visualizar una manera de resolver el problema, es conveniente expresar cómo se verían las aplicaciones de $\lambda g\ w.w(g\ x)$ para un numeral de Church en particular. Si

consideramos la aplicación de $\bar{4}\,(\lambda g\, w.w(g\, x))$:

$$\begin{aligned}
&\bar{4}(\lambda g\, w.w(g\, x))\\
\equiv&((\lambda x\, y.x^4\, y)(\lambda g.(\lambda w.(w(g\, x)))))\\
\equiv&((\lambda x\, y.x(x(x(x\, y))))(\lambda g.(\lambda w.(w(g\, x)))))\\
\rightarrow_\beta&(\lambda y.(\lambda g.(\lambda w.(w(g\, x))))((\lambda g.(\lambda w.(w(g\, x))))((\lambda g.(\lambda w.(w(g\, x))))((\lambda g.(\lambda w.(w(g\, x))))y))))
\end{aligned}$$

Esto nos lleva al segundo paso para encontrar la función predecesor, en el desarrollo anterior notamos que la primer aplicación de $\lambda g\, w.w(g\, x)$ es en la variable y la cual está ligada por la λ del término. Sabemos que para obtener $\lambda w.w\, x^3\, y$ debemos β-reducir el término:

$$((\lambda g.(\lambda w.(w(g\, x))))((\lambda g.(\lambda w.(w(g\, x))))((\lambda g.(\lambda w.(w(g\, x))))(\lambda w.w\, y))))$$

Con esto podemos encontrar el valor que tiene que tomar y en el numeral ya que:

$$((\lambda g.(\lambda w.(w(g\, x))))y) \twoheadrightarrow_\beta \lambda w.w\, y$$

El término que buscamos es el que debe sustituír a la variable y en la reducción:

$$\begin{aligned}
&((\lambda g.(\lambda w.(w(g\, x))))?)\\
\twoheadrightarrow_\beta&(\lambda w.(w(?\, x)))
\end{aligned}$$

El término ? debe ser una función que al ser aplicada a x se reduzca a y. El término $\lambda u.y$ cumple con esta propiedad y será el que utilizaremos.

Considerando los términos determinados en el procedimiento anterior, podemos decir cómo será la función predecesor. Primero se aplica $(\lambda g.(\lambda w.(w(g\ x))))$ a $\bar{n}$, este término resultante se aplica a $\lambda u.y$, β-reducir esta aplicación nos resulta $\lambda w.w\ x^{n-1}\ y$ la cual puede ser aplicada a la función identidad $\lambda a.a$ para obtener $x^{n-1}\ y$. Lo cual nos lleva al término completo de predecesor:

$$(\lambda n.(\lambda x\ y.(((n(\lambda g.(\lambda w.(w(g\ x)))))(\lambda u.y))(\lambda a.a))))$$

Teniendo los términos lambda de sucesor y predecesor se puede abordar la derivación de operaciones mas complejas como la de adición, multiplicación, exponenciación y sustracción de numerales de Church siguiendo el mismo enfoque. En este trabajo no se abordan otras operaciones como la división debido al aumento de complejidad por no ser una operación interna, es decir, la división de dos naturales puede ser un racional y no se definió una representación de términos lambda para el conjunto de los racionales.

Un término lambda para la adición de dos numerales $\bar{m}$ y $\bar{n}$ es

$$\lambda m\ n.(\lambda x\ y.n \text{ sucesor } m)$$

y se obtuvo a partir de la observación de que realizar la suma $m + n$ es equivalente a computar el n-ésimo sucesor de m.

Utilizando la estructura de $\bar{n}$ podemos aplicar $\bar{n}$ sucesor $\bar{m}$ para obtener la n-ésima composición de la función sucesor aplicada al numeral $\bar{m}$:

$$
\begin{aligned}
&\bar{n}\ \text{sucesor}\ \bar{m}\\
\equiv&(((\lambda x\ y.x^n\ y)\text{sucesor})(\lambda x\ y.x^m\ y))\\
\rightarrow_\beta&((\lambda y.\text{sucesor}^n\ y)(\lambda x\ y.x^m\ y))\\
\rightarrow_\beta&\text{sucesor}^n\ \lambda x\ y.x^m\ y\\
\twoheadrightarrow_\beta&\lambda x\ y.x^{m+n}\ y\\
\equiv&m\ \bar{+}\ n
\end{aligned}
$$

Un término lambda para la multiplicación de dos numerales de Church es

$$\lambda m\ n\ x\ y.n(m\ x)y$$

el cual aborda la idea de componer $m\ n$ consigo mismo n veces (lo cual equivaldría a sumar n veces m.

En el caso de la adición y la multiplicación, el orden en el que aplicamos el término a los numerales no es de importancia ya que son operaciones conmutativas, $m+n = n+m$ y $m \times n = n \times m$. Sin embargo en la sustracción y la exponenciación no se tiene esta propiedad, por lo que es importante el orden en el que se aplican los numerales a los términos, para ello consideraremos el orden como $m - n$ y m^n.

Basándonos en el término de la adición podemos obtener un término de la sustracción el cual es

$$\lambda m\ n\ x\ y.n\ \text{predecesor}\ m$$

.

Ya que en la adición se dejó explícito el acto de aumentar m veces en 1 a n, cambiamos el término de sucesor por el de predecesor y ahora se decrementa m veces en 1 a n.

Un término lambda para la exponenciación es

$$\lambda m\ n.n\ m$$

es curioso tener una representación tan sencilla para una operación tan compleja como esta. A diferencia de los anteriores términos, al aplicarle a ésta exponenciación dos numerales, el numeral resultante tendrá las variables compuestas las variables que no se componen en las entradas, es decir, si $\bar{m} \equiv \lambda f\ g.f^m\ g$ y $\bar{n} \equiv \lambda x\ y.x^n\ y$, el resultado será $\bar{m^n} \equiv \lambda g\ y.g^{m^n}\ y$.

Para corroborar que estas representaciones calculan de manera correcta la operación correspondiente para los numerales de Church se pueden realizar varias pruebas con diferentes numerales de entrada. En este trabajo no se desarrollarán ejemplos para estos términos.

Los mecanismos que hemos utilizado para derivar las operaciones se basan en construír términos que vayan transformando entradas con una estructura determinada de tal manera que nos acerquemos poco a poco al cálculo de la operación deseada; esta labor llega a ser bastante tediosa y carece de interés algorítmico. A continuación se presenta una manera mas interesante y elegante de abordar el problema de representar operaciones aritméticas.

Se introduce el término lambda que me permite generar hiperoperaciones aritméticas:

$$\lambda f\ u\ m\ n.n(\lambda w.f\ m\ w)u$$

Abstracción de la noción de repetición sobre la estructura de un numeral, considerar propiedades de conmutatividad y asociatividad en operaciones. Abordar el problema del cómputo de operaciones inversas. Determinar un término que nos genere elementos de la secuencia de hiperoperaciones.

3.3. Procesos recursivos

Combinador Y, ordenes de evaluación, funciones recursivas.

$$\boldsymbol{Y} \equiv \lambda f.(\lambda x.f(x\,x))(\lambda x.f(x\,x))$$

Derivación de $\boldsymbol{Y}$

$$F \equiv \lambda n.\prec(\mathbf{o}^?\,n)\bar{1}(\times\,n(F(-\,n\,\bar{1}))) \tag{3.1}$$

$$F' \equiv \lambda f\,n.\prec(\mathbf{o}^?\,n)\bar{1}(\times\,n(f(-\,n\,\bar{1}))) \tag{3.2}$$

$$F'' \equiv \lambda f'\,n.\prec(\mathbf{o}^?\,n)\bar{1}(\times\,n(f'\,f'(-\,n\,\bar{1}))) \tag{3.3}$$

$$C \equiv \lambda f''.f''\,f'' \tag{3.4}$$

$$C\,F'' \twoheadrightarrow_\beta \text{ función factorial a partir de } F'' \tag{3.5}$$

$$F''' \equiv (\lambda f'''.f'(f'''\,f''')) \tag{3.6}$$

$$F'''\,F''' \twoheadrightarrow_\beta f'(F'''\,F''') \tag{3.7}$$

$$C^* \equiv \lambda f'.(\lambda f'''.f'(f'''\,f'''))(\lambda f'''.f'(f'''\,f''')) \tag{3.8}$$

$$C^*\,F' \twoheadrightarrow_\beta \text{ función factorial a partir de } F' \tag{3.9}$$

$$Y \equiv C^*$$

Presentar una breve introducción sobre los combinadores y hablar del combinador $\boldsymbol{Y}$ y cómo nos permite expresar funciones recursivas en el cálculo lambda.

Como ejemplos prácticos de esta subsección sería adecuado desarrollar el término para el cálculo de factoriales o algúna otra función de una sola variable que transforme un numeral de Church en otro. También pudiera expandir la recursividad a términos multivariables currificados como la función Ackermann (abstracción a la generación de hiperoperaciones, ver *The Book of Numbers* de Conway).

3.4. Pares ordenados

Construcción axiomática de pares ordenados, listas, n-tuplas, árboles y otras estructuras complejas.

Presentar la representación de pares ordenados para la construcción de estructuras mas complejas.

$$\text{Car}(\text{Cons}(x, y)) = x$$

$$\text{Cdr}(\text{Cons}(x, y)) = y$$

Esta sección es apropiada para comenzar a relacionar la teoría de autómatas, lenguajes regulares y libres de contexto con sistemas medianamente complejos que se pueden incrustar en el cálculo lambda sin modificar el sistema. Un problema pudiece ser el no determinismo, pero pudiera solventar esto con el desarrollo de operaciones funcionales sobre listas (map, filter, fold, etc).

Cambios en la metodología

Expandir el concepto de valores de verdad al de pares ordenados

Constructor

Algoritmo 3.6 Construír un par

Entrada: a, b
Salida: $(a,\ b)$
Entrada: $x \in \{\text{falso},\ \text{verdadero}\}$

```
CreaPar (primero, segundo) =

    Elige (x) =
```

```
        Si x, entonces:
            primero
        De lo contrario:
            segundo
    Elige
```

$$\begin{aligned} \otimes &\equiv \lambda a\ d.\lambda x.B\ x\ a\ d \\ &\rightarrow_\beta \lambda a\ d.\lambda x.x\ a\ d \\ &\equiv \lambda a\ d\ x.x\ a\ d \end{aligned}$$

Selectores

```
Primero (Elige) =
    Elige(T)
```

$$\otimes_1 \equiv \lambda x.x\ T$$

```
Segundo (Elige) =
    Elige(F)
```

$$\otimes_2 \equiv \lambda x.x\ F$$

Bibliografía

[1] A. S. Troelstra, H. S. *Basic Proof Theory*, 2nd ed. Cambridge Tracts in Theoretical Computer Science 43. Cambridge University Press, 2000.

[2] Barendregt, H. P. *The Lambda Calculus Its Syntax and Semantics*, revised ed ed., vol. 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland, 1984.

[3] Bridges, D., and Palmgren, E. Constructive mathematics. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., winter 2013 ed. 2013.

[4] Cardone, F., and Hindley, J. R. Lambda-calculus and combinators in the 20th century. In *Logic from Russell to Church*, D. M. Gabbay and J. Woods, Eds., vol. 5 of *Handbook of the History of Logic*. North-Holland, 2009, pp. 723 – 817.

[5] Church, A. A set of postulates for the foundation of logic. *Annals of Mathematics 33*, 2 (1932), 346–366.

[6] Church, A. *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jork, 1941.

[7] Haskell Brooks Curry, Robert Feys, W. C. *Combinatory Logic, Volume I*. Studies in Logic and the Foundations of Mathematics 22. North-Holland Publishing Company, 1958.

[8] Hindley, J. R., and Seldin, J. P. *Lambda-Calculus and Combinators*, 2 ed. Cambridge University Press, 2008.

[9] Hofstadter, D. R. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., New York, NY, USA, 1979.

[10] McCarthy, J. Recursive functions of symbolic expressions and their computation by machine, part i. *Commun. ACM 3*, 4 (Apr. 1960), 184–195.

[11] Mendelson, E. *Introduction to Mathematical Logic*, 5 ed. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2010.

[12] Milner, R. Logic for computable functions – description of a machine implementation. Tech. Rep. CS-TR-72-288, Stanford University, Department of Computer Science, May 1972.

[13] Peyton Jones, S., et al. The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming 13*, 1 (Jan 2003).

[14] Pierce, B. C. *Types and Programming Languages*. MIT Press, Cambridge, MA, USA, 2002.

[15] Quine, W. V. *The Ways of Paradox, and Other Essays*. Harvard University Press, 1976.

[16] S. C. Kleene, J. B. R. The inconsistency of certain formal logics. *Annals of Mathematics 36*, 3 (1935), 630–636.

[17] Schönfinkel, M. Über die bausteine der mathematischen logik. *Mathematische Annalen 92*, 3 (1924), 305–316.

[18] Scott, D. S. Lambda calculus then and now. In *ACM Turing Centenary Celebration* (2012), ACM-TURING '12, ACM.

[19] Statman, R. The typed λ-calculus is not elementary recursive. In *Foundations of Computer Science, 1977., 18th Annual Symposium on* (Oct 1977), pp. 90–94.

[20] Sussman, G. J., and Steele Jr, G. L. SCHEME : an interpreter for extended lambda calculus. Tech. Rep. AI 349, Massachussetts Institute of Technology (MIT). Cambridge (MA US), 1975.