El cálculo lambda y los fundamentos de la computación

Eduardo Acuña Yeomans

2016

Índice general

Capítulo 1

El cálculo lambda

El cálculo lambda es un sistema formal originalmente creado por Alonzo Church en 1932 [5] 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 utilizado para describir expresiones; la interpretación, representación y manejo de funciones; y las nociones de transformación y equivalencia de expresiones.

Este capítulo consiste de dos secciones: primero se introduce informalmente el cálculo lambda, en donde se enfatiza las diferencias conceptuales, operacionales y de notación entre este sistema, la matemática clásica y la lógica de primer orden; en la segunda sección se presenta la formalización del cálculo lambda.

El contenido de este capítulo está basado en los primeros cuatro capítulos del libro *The Lambda Calculus, Its Syntax and Semantics* por H.P. Barendregt [2] y los capítulos 1, 3, 6, 7 y 8 del libro *Lambda Calculus and Combinators, an Introduction* por J.R. Hindley y J.P. Seldin [8].

1.1. Noción informal del cálculo lambda

En el estudio del cálculo lambda existen dos lenguajes, el de las expresiones del sistema y el que se utiliza para describir y analizar estas expresiones, llamado metalenguaje.

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 permite describir *cómo* es que estas expresiones son manipuladas y comparadas, así como los mecanismos para representar conceptos y objetos matemáticos como expresiones.

1.1.1. Expresiones

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

Las expresiones más simples son los átomos, usualmente representados con un símbolo como x, y o z. Estas expresiones también son llamadas variables y al igual que en la matemática clásica y la lógica de primer orden, no tienen mucha importancia por sí solas, pero al estar asociadas a cuantificadores ($\forall x$ y $\exists x$) o a funciones ($f(x)$) pueden representar valores complejos.

Las abstracciones y aplicaciones son expresiones con estructura, es posible identificar y referirse a sus partes. Estos son complementarias: las abstracciones representan la generalización de una expresión y se asocian al concepto de *función*, mientras que las aplicaciones representan el acto de concretar una expresión y se asocian al concepto de *aplicación de funciones*.

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.

Debido a la notación utilizada puede parecer que las abstracciones se relacionan directamente con las funciones $f(x) = M$ o con fórmulas lógicas $\forall x\ M$. Sin embargo, tanto las expresiones de funciones como las fórmulas lógicas están basadas en conjuntos y en operaciones sobre estos conjuntos, 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 cuantificar propiedades de las expresiones utilizando números, sin embargo estos objetos matemáticos no están incrustados en el lenguaje de las expresiones y conforman lo que es el metalenguaje.

Las aplicaciones son expresiones conformadas a partir de otras dos expresiones, se escriben de la forma (MN) donde M y N son cualesquiera átomos, abstracciones o aplicaciones. El concepto relacionado con las aplicaciones en la matemática clásica es el del acto de obtener un elemento del codominio de una función a partir de un elemento en su dominio, por ejemplo, considerando la función $f(x) = x^2$, la aplicación de f en 4 es $f(4)$. La notación del cálculo lambda es similar a la notación tradicional $f(x)$, solo que con el paréntesis abierto antes de la función, es decir, (fx). De manera similar a la aplicación de funciones, existe una operación la cual permite transformar expresiones de la forma (MN) donde N es cualquier expresión y M es una abstracción, a otra expresión Z similar al cuerpo de M pero con el argumento de M intercambiado por N.

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.

Cuando se desea representar alguna función en el cálculo lambda, 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 $f\colon \mathbb{N} \to \mathbb{N}$ 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.

El hecho de tener un lenguaje tan reducido y minimalista para las expresiones del cálculo lambda nos permite 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.

Con solo átomos, aplicaciones y abstracciones se pueden formular expresiones complejas. A continuación se presentan seis ejemplos de expresiones 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 en 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 realizar la operación de aplicar $(\lambda y.(y(\lambda x.x)))$ a $(\lambda w.w)$, el átomo y es intercambiado por la expresión $(\lambda w.w)$ la cual a su vez puede ser aplicada a la expresión $(\lambda x.x)$.

Ejemplo 1.1.2. Procedimiento de aplicar $(\lambda y.(y(\lambda x.x)))$ 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 a partir de la aplicación cualquier expresión a la que sea aplicada. Por ejemplo, si aplicamos $(\lambda x.(xx))$ al átomo y y se realiza el procedimiento de aplicación como en el ejemplo 1.1.2, se obtiene (yy) y en general al realizar la operación de aplicación sobre $((\lambda x.(xx))M)$ donde M es cualquier expresión, se obtiene (MM). Con ésta expresión se puede formular una expresión auto-replicante en el cálculo lambda:

Ejemplo 1.1.3. Procedimiento de aplicar $(\lambda x.(xx))$ a $(\lambda x.(xx))$

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

En (xx) se cambia cada x por $(\lambda x.(xx))$

2. $((\lambda x.(xx))(\lambda x.(xx)))$ resultado del procedimiento.

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 la representación de funciones de varias variables. Al realizar la operación de abstracción de $(\lambda f.(\lambda x.(f(fx))))$ a una expresión cualquiera M se obtiene $(\lambda x.(M(Mx)))$. Si posteriormente se realiza la aplicación de este resultado a una expresión cualquiera N se obtiene $(M(MN))$, esto sería similar al resultado que se obtendría de aplicar una función con argumentos f y x, con cuerpo $f(f(x))$ a dos valores de su dominio M y N.

Otra manera de representar funciones de varias variables como abstracciones del cálculo lambda es codificando *tuplas* o *secuencias* y poder hacer referencia a sus elementos de manera individual en el sistema, sin embargo, representar secuencias es un mecanismo más complejo que se aborda en el siguiente capítulo.

1.1.2. 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 subsección anterior se mencionan dos operaciones que se abordaron tangencialmente: el intercambio en una expresión de un átomo por otra expresión y la operación de aplicación de abstracciones.

La *sustitución* es la operación que nos permite transformar una expresión cualquiera M intercambiando las apariciones de un átomo x por alguna otra expresión M, este procedimiento se denota

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

En muchos casos la operación de sustitución se puede realizar de manera trivial:

Ejemplo 1.1.4. Sustituciones sencillas

$$x\,[x := y] = y \tag{a}$$
$$(x(x(\lambda y.y)))\,[x := z] = ((zz)(\lambda y.y)) \tag{b}$$
$$(((wx)y)z)\,[x := a]\,[y := b] = (((wa)b)z) \tag{c}$$
$$(xx)\,[x := (\lambda w.w)] = ((\lambda w.w)(\lambda w.w)) \tag{d}$$

Existen algunos detalles 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 la función constante $(\lambda x.y)$, la cual al ser aplicada a cualquier otra expresión, resultará siempre en el átomo y. Si se realiza la operación $(\lambda x.y)\,[y := z]$ se obtiene la expresión $(\lambda x.z)$ la cual también es análoga a la 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 1.1.5. Caso patológico de la sustitución ingenua

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

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 un átomo x es una λx. Consideremos la expresión

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

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 1.1.5 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 $M\,[x := y]$ no debe introducir o eliminar referencias a alguna λ en M.

Para resolver el problema presentado en el ejemplo 1.1.5 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 los átomos x (para cualquier átomo x) es más importante 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 1.1.6. Procedimiento de sustitución para el ejemplo 1.1.5

1. $(\lambda x.y)\,[y:=x]$
2. $(\lambda z.y)\,[y:=x]$ 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. En el ejemplo 1.1.6 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 1.1.7. Múltiples cambios de variable ligada

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

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

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

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 diferentes:

- $x\,[x:=M]$ resulta en M;
- $y\,[x:=M]$ resulta en y;
- $(MN)\,[x:=P]$ 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 *aplicación de abstracciones* 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 aplicación 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 aplicación de una abstracción $(\lambda x.M)$ en una expresión N es

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

A continuación se presentan ejemplos de aplicación de abstracciones con los pasos de la transformación

Ejemplo 1.1.8. Procedimiento de aplicación de $(\lambda x.x)$ a y

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

Ejemplo 1.1.9. Procedimiento de aplicación de $(\lambda w.w)$ a x

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

Ejemplo 1.1.10. Procedimiento de múltiples aplicaciones

1. $(((\lambda f.(\lambda x.(f(f(fx)))))g)y)$
2. $((\lambda x.(f(f(fx)))\,[f := g])y)$
3. $((\lambda x.(g(g(gx))))y)$
4. $(g(g(gx)))\,[x := y]$
5. $(g(g(gy)))$

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*.

1.1.3. 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 subsección anterior, con la operación de sustitución se puede transformar expresiones del cálculo lambda y definir otras operaciones como el cambio de variable ligada y la aplicación de abstracciones. 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.

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 son sintácticamente 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.

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 que representa 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 $((xy)z)$ o $(\lambda w.x)$ son estructuralmente equivalentes a $((ab)c)$ 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.

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 1.1.11 (índices de De Bruijn)**.** La expresión $(\lambda z.((\lambda y.(y(\lambda x.x)))(\lambda x.(zx))))$ 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 transformación pueda ser realizada.

Al igual que el cambio de variable ligada, la operación de aplicación de abstracciones es utilizada para describir una equivalencia entre expresiones. La noción básica de esta equivalencia consiste en observar que al aplicar una abstracción $(\lambda x.M)$ a una expresión N, el resultado de dicha operación 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 aplicaciones de abstracciones, cambios de variable ligada o el proceso inverso de aplicación de abstracciones en M o parte de M y N o parte de N se puedan obtener expresiones sintácticamente equivalentes.

Es importante enfatizar que la β-convertibilidad considera el proceso inverso de la aplicación de abstracciones, por ejemplo

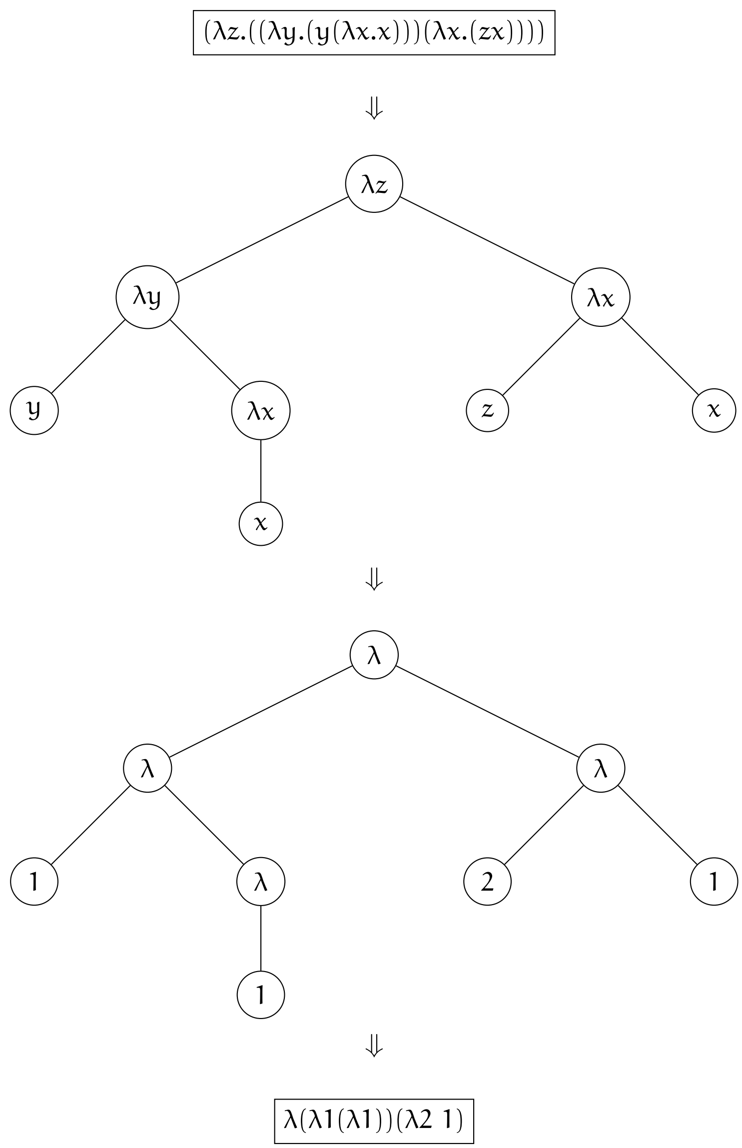

Figura 1.1: Transformación gráfica del ejemplo 1.1.11

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

Todas las relaciones de equivalencia por definición cumplen con tres propiedades:

a. Toda expresión M es equivalente a sí misma.

b. Si una expresión M es relacionada con una equivalencia a otra expresión N, entonces N también es relacionada a 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.

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 propiedades 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 1.1.1 (Ecuaciones de $\sim$)**.** Las ecuaciones con $\sim$ que se satisfacen para expresiones del cálculo lambda son

$$(\lambda x.M) \sim (\lambda y.M\,[x := y]) \qquad (\alpha)$$
$$((\lambda x.M)N) \sim M\,[x := N] \qquad (\beta)$$
$$M \sim M \qquad (\rho)$$
$$M \sim N \implies N \sim M \qquad (\sigma)$$
$$M \sim N,\ N \sim P \implies M \sim P \qquad (\tau)$$

Las ecuaciones en la definición 1.1.1 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

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

pero

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

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

$$M \sim N \implies (MZ) \sim (NZ) \tag{ν}$$
$$M \sim N \implies (ZM) \sim (ZN) \tag{μ}$$
$$M \sim N \implies (\lambda x.M) \sim (\lambda x.N) \tag{ξ}$$

Con estas reglas, las inferencias lógicas nos permiten abordar la equivalencia sobre partes de una expresión.

Ejemplo 1.1.12. Razonamiento para concluir que $(\lambda f.((\lambda x.(fx))y)) \sim (\lambda f.(fy))$

1. $((\lambda x.(fx))y) \sim (fy)$ por β
2. $(\lambda f.((\lambda x.(fx))y)) \sim (\lambda f.(fy))$ 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:

$$(MP) \sim (NP) \implies M \sim N$$

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

1.2. 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 λ, también llamado cálculo-$\lambda K\beta$.

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

1.2.1. 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 1.2.1 (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:

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

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 MN (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 1.2.2 (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:

$$T \to E \mid F \mid A \tag{a}$$
$$A \to v_0 \mid A_0 \tag{b}$$
$$F \to (\lambda A.T) \tag{c}$$
$$E \to (TT) \tag{d}$$

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:

- $((MN)P) \equiv MNP$, 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 $(M(N(OP))) \equiv M(N(OP)) \not\equiv MNOP$.
- $(\lambda x.(MN)) \equiv \lambda x.(MN)$, en general, se puede escribir una abstracción omitiendo los paréntesis externos. Es importante escribir de manera explícita los paréntesis en algunos casos, por ejemplo $((\lambda x.(MN))O) \equiv (\lambda x.(MN))O \not\equiv \lambda x.(MN)O$ ya que el lado derecho de la equivalencia es sintácticamente equivalente a $(\lambda x.((MN)O))$.
- $(\lambda x.(\lambda y.(\lambda z.M))) \equiv (\lambda xyz.M)$, 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 1.2.1.

$$\begin{aligned}
(((xy)z)(yx)) &\equiv xyz(yx) \\
(\lambda x.((ux)y)) &\equiv \lambda x.yxy \\
(\lambda y.(u(\lambda x.y))) &\equiv \lambda u.u(\lambda x.y) \\
(((\lambda y.((vu)u))z)y) &\equiv (\lambda y.vuu)zy \\
(((ux)(yz))(\lambda v.(vy))) &\equiv ux(yz)(\lambda v.vy) \\
((((\lambda x.(\lambda y.(\lambda z.((xz)(yz)))))u)v)w) &\equiv (\lambda xyz.xz(yz))uvw
\end{aligned}$$

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

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

$$M\vec{N} \equiv MN_1N_2...N_n$$

En algunas demostraciones realizadas por inducción, se usa la expresión “inducción sobre M” para referirse a la inducción sobre la *longitud* de M.

Definición 1.2.3 (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:

$$\begin{aligned} \|x\| &= 1 \\ \|MN\| &= \|M\| + \|N\| \\ \|\lambda x.M\| &= 1 + \|M\| \end{aligned}$$

Por ejemplo, la longitud del término lambda $x(\lambda y.yux)$ 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 zz 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 1.2.4 (Subtérmino)**.** M es un subtérmino de N, denotado $M \subset N$ si $M \in \mathrm{Sub}(N)$ es la colección de subtérminos de N definida de manera inductiva como

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

Definición 1.2.5 (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 NZ, 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 1.2.2. Sea $M \equiv \lambda x.xy(\lambda z.y)$:

- el término $xy \subset M$;

- 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 $M \equiv \lambda x.xy(\lambda z.y) \equiv (\lambda x.((xy)(\lambda z.y)))$, en este caso, la clave está en observar la estructura de la aplicación $xy(\lambda z.y)$.
- Las apariciones de x y $\lambda z.y$ en M son disjuntas.
- Los términos x y (xy) 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 1.2.6 (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 1.2.3. Sea $M \equiv x(\lambda y.xy)$:

- 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 1.2.7 (Variables libres)**.** El conjunto de variables libres de un término lambda M se denota $FV(M)$ y se define de manera inductiva como:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

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

Ejemplo 1.2.4. Consideremos los términos $x(\lambda x.xyz)$, $\lambda xyz.y$ y $(\lambda y.x)\lambda x.y$.

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

En ocasiones 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 1.2.8 (Términos cerrados)**.** Se denota como Λ^0 al conjunto

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

La notación Λ^0 se puede generalizar para identificar diferentes subconjuntos de Λ a partir de las variables libres de los términos lambda:

$$\Lambda^0(\vec{x}) = \{M \in \Lambda \mid FV(M) \subseteq \{\vec{x}\}\}$$

De tal manera que:

$$\Lambda^0 = \Lambda^0(\emptyset)$$

Todos los términos lambda en $\Lambda \setminus \Lambda^0$ tienen al menos una clausura en Λ^0.

Definición 1.2.9 (Clausura)**.** La clausura de un término lambda M con $FV(M) \neq \emptyset$ es un término lambda

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

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

Ejemplo 1.2.5. Consideremos el términio lambda $M \equiv \lambda z.xyz$

- $\lambda xy.\lambda z.xyz$ es una clausura de M;
- $\lambda yxz.xyz$ es una clausura de M;
- $\lambda zxy.\lambda z.xyz$ 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^nM. Cuando se aplica n veces un término M por la derecha a otro término F se denota $FM^{\sim n}$. La definición formal de esta notación es:

$$\begin{aligned} F^{n+1}M &\equiv F(F^nM) \\ F^0M &\equiv M \end{aligned}$$

$$\begin{aligned} FM^{\sim n+1} &\equiv (FM^{\sim n})M \\ FM^{\sim 0} &\equiv F \end{aligned}$$

Ejemplo 1.2.6. El término lambda $\lambda fx.f(f(f(fx)))$ se puede escribir de manera compacta como

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

Ejemplo 1.2.7. El término lambda $\lambda fx.fxxxx$ se puede escribir de manera compacta como

$$\lambda fx.fx^{\sim 4}$$

Definición 1.2.10 (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:

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

Ejemplo 1.2.8. Procedimientos de sustituciones para cada uno de los casos de la definición 1.2.10:

- Caso $x[x := N]$

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

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

$$z[w := xx] \equiv z$$

- Caso $(PQ)[x := N]$

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

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

$$(\lambda fx.ffx)[f := g] \equiv \lambda fx.ffx$$

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

$$(\lambda fx.ffx)[f := g] \equiv \lambda fx.ffx$$

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

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

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

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

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

Lema 1.2.1. *Si* $(yx) \notin FV(L)$ *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 1.2.11 (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 1.2.9. Consideremos el contexto $C[\ \ ] \equiv \lambda x.x\lambda y.[\ \ ]$ y el término lambda $M \equiv (xy)$.

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

El caso análogo con la sustitución es

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

1.2.2. Teoría $\boldsymbol{\lambda K\beta}$

El objetivo principal de esta subsección es presentar una formalización del cálculo lambda descrito en 1.1 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 $\langle A_1, ..., A_n, B\rangle$ tal que

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

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

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

Observación. En la literatura se pueden encontrar 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 $\langle A_1, \dots, A_{n+1}\rangle$ con n premisas y una conclusión, en donde los axiomas se definen como las reglas en $\mathcal{R}$ con cero premisas. La definición de teoría formal es del estilo Hilbert y está basada en [8, pp. 69–70].

Si consideramos un conjunto $\Gamma \subseteq \mathcal{F}$, una *deducción* deuna fórmula B desde Γ es un árbol dirigido de fórmulas en donde los vértices de un extremo son elementos de $\mathcal{A}$ 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 $\mathcal{T}$ suponiendo Γ, denotado

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

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

$$\mathcal{T} \vdash B$$

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 $\mathcal{F}$ tiene elementos $M = N$ donde $M, N \in \Lambda$.

Definición 1.2.12 (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:

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

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

$$M = M \tag{ρ}$$

Y las reglas:

$$\frac{M = N}{ZM = ZN} \tag{μ}$$

$$\frac{M = N}{MZ = NZ} \tag{ν}$$

$$\frac{M = N}{\lambda x.M = \lambda x.N} \tag{ξ}$$

$$\frac{M = N \quad N = P}{M = P} \tag{τ}$$

$$\frac{M = N}{N = M} \tag{σ}$$

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:

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

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 1.2.13 (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 1.2.10. Consideramos los términos lambda $M \equiv (\lambda f.x((\lambda y.yf)\lambda z.z))w$ y $N \equiv xw$. Se muestra que

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

formulando el árbol de deducción de la figura 1.2.

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

$$
\begin{tikzcd}[column sep=small]
\overset{(\beta)}{\boxed{(\lambda y.yf)\lambda z.z = (\lambda z.z)f}} \arrow[dr] & & \overset{(\beta)}{\boxed{(\lambda z.z)f = f}} \arrow[dl] & \\
& \overset{(\tau)}{\boxed{(\lambda y.yf)\lambda z.z = f}} \arrow[d, "(\mu)"] & & \\
& \boxed{x((\lambda y.yf)\lambda z.z) = xf} \arrow[d, "(\xi)"] & & \\
& \boxed{\lambda f.x((\lambda y.yf)\lambda z.z) = \lambda f.xf} \arrow[d, "(\nu)"] & & \\
& \boxed{(\lambda f.x((\lambda y.yf)\lambda z.z))w = (\lambda f.xf)w} \arrow[dr] & & \overset{(\beta)}{\boxed{(\lambda f.xf)w = xw}} \arrow[dl] \\
& & \overset{(\tau)}{\boxed{(\lambda f.x((\lambda y.yf)\lambda z.z))w = xw}} &
\end{tikzcd}
$$

Figura 1.2: Árbol de deducción para la ecuación del ejemplo 1.2.10

Definición 1.2.14 (Combinadores SKI)**.** Tres términos lambda de suma importancia son

$$\begin{aligned} \mathbf{I} &\equiv \lambda x.x \\ \mathbf{K} &\equiv \lambda xy.x \\ \mathbf{S} &\equiv \lambda xyz.xz(yz) \end{aligned}$$

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

$$\begin{aligned} \mathbf{I}M &=_{\lambda} M \\ \mathbf{K}MN &=_{\lambda} M \\ \mathbf{S}MNL &=_{\lambda} ML(NL) \end{aligned}$$

Estos tres combinadores generan en la teoría $\boldsymbol{\lambda}$ al conjunto Λ^0 con combinaciones de aplicaciones. Debido a que $\mathbf{SKK} =_{\lambda} \mathbf{I}$, sólo es necesario combinar con aplicaciones a $\mathbf{K}$ y a $\mathbf{S}$ para generar cualquier término cerrado.

Otras teorías

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 1.2.15 (Términos en Λ_I)**.**

$$\begin{aligned} x \in V &\implies x \in \Lambda_I \\ M \in \Lambda_I,\ x \in \mathrm{FV}\, M &\implies \lambda x.M \in \Lambda_I \\ M, N \in \Lambda_I &\implies MN \in \Lambda_I \end{aligned}$$

La diferencia fundamental entre las teorías $\boldsymbol{\lambda K \beta}$ y $\boldsymbol{\lambda I \beta}$ es el término lambda $\mathbf{K}$, ya que $\mathbf{K} \in \Lambda \setminus \Lambda_I$ pero $\mathbf{K} \notin \Lambda_I$. Esto es debido a que el subtérmino $\lambda y.x$ en $\mathbf{K}$ de la definición 1.2.14 no puede existir en Λ_I debido a que $y \notin \mathrm{FV}(x)$.

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

$$\boldsymbol{\lambda} \vdash FX = GX$$

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

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 1.2.12.

Definición 1.2.16 (Reglas de extensionalidad)**.** Cada una de las siguientes reglas nos permite añadir a $\boldsymbol{\lambda}$ la propiedad de extensionalidad.

Reglas de inferencia

$$\frac{Mx = Nx}{M = N} \qquad \text{si } x \notin FV(MN) \qquad (\zeta)$$

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

Axiomas

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

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.

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 1.2.17 (Regla)**.** Una regla $\mathcal{R}$ de n premisas es un conjunto de secuencias $\mathcal{S}_0, ..., \mathcal{S}_{n-1}, \mathcal{S}$ 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

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

$\mathcal{S}$ 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 1.2.18 (Reglas derivables y admisibles)**.** Sea $\mathcal{R}$ una regla determinada por una función $\phi\colon \mathcal{F}^n \to \mathcal{F}$. Se dice que $\mathcal{R}$ es *derivable* en $\mathcal{T}$ si y solo si, por cada instancia de $\mathcal{R}$, su conclusión es deducible en $\mathcal{T}$ a partir de sus premisas, es decir

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

.

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

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

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

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

$$\mathcal{T} \vdash C$$

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

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

1.2.3. Teoría de reducción

Contracciones

Transformaciones de términos con un paso.

Reducciones

Reducciones basadas en contracciones.

Convertibilidad

Ecuaciones desde una perspectiva de reducción

Teorema de Church-Rosser

Confluencia en términos lambda.

1.2.4. Árboles Böhm

Estructura con perspectiva conjuntivista, no se formaliza el modelo pero se describe.

Capítulo 2

Codificación de objetos

2.1. Álgebra Booleana

Uno de los atributos del cálculo lambda es que nos permite codificar información usando términos lambda y definir algoritmos utilizando reducciones que manipulen estos términos. Un buen ejemplo de esto es la representación de valores de verdad, de los cuales podemos derivar las operaciones elementales del álgebra booleana.

Se establecen primero los cimientos de este sistema booleano primitivo determinando la representación de los valores *verdadero* y *falso*. Los algoritmos que se elaboran en esta subsección se basan en los términos que se les asocia a estos dos valores.

2.1.1. Valores de verdad

Es posible formular predicados lógicos con la notación del cálculo lambda. Se le asignan términos lambda a los valores de falso y verdadero. El término asociado a verdadero es $\lambda xy.x$ y el término asociado a falso es $\lambda xy.y$.

A partir de estos términos se pueden obtener nuevos términos lambda para los operadores básicos del álgebra booleana, para el resto de la sección se define $\mathsf{V} \equiv \lambda xy.x$ (verdadero) y $\mathsf{F} \equiv \lambda xy.y$ (falso), es decir, V es una función de dos parámetros que al ser evaluada regresa el primer argumento y F es una función de dos parámetros que al ser evaluada regresa el segundo argumento.

2.1.2. Conectivos lógicos

Explorando con esta representación podemos obtener términos lambda obtenidos a partir de la aplicación de las combinaciones de estos dos valores.

Combinaciones de la aplicación de dos valores de verdad.

- **VV**
  1. $(\lambda xy.x)(\lambda xy.x)$
  2. $\rightarrow_\beta \lambda yxw.x$

  Función de tres parámetros que al ser evaluada regresa el segundo argumento.

- **VF**
  1. $(\lambda xy.x)(\lambda xy.y)$
  2. $\rightarrow_\beta \lambda yxw.w$

  Función de tres parámetros que al ser evaluada regresa el tercer argumento.

- **FV**
  1. $(\lambda xy.y)(\lambda xy.x)$
  2. $\rightarrow_\beta \lambda y.y$

  Función de un parámetro que al ser evaluada regresa el argumento.

- **FF**
  1. $(\lambda xy.y)(\lambda xy.y)$
  2. $\rightarrow_\beta \lambda y.y$

  Función de un parámetro que al ser evaluada regresa el argumento

Combinaciones de la aplicación de tres valores de verdad.

- **VVV**
  1. $(\lambda yxw.x)(\lambda xy.x)$
  2. $\rightarrow_\beta \lambda xw.x$
  3. $\rightarrow_\alpha$ **V**

- **VVF**
  1. $(\lambda yxw.x)(\lambda xy.y)$
  2. $\rightarrow_\beta \lambda xw.x$
  3. $\rightarrow_\alpha$ **V**

- **VFV**
  1. $(\lambda yxw.w)(\lambda xy.x)$
  2. $\rightarrow_\beta \lambda xw.w$

3. $\rightarrow_\alpha \mathbf{F}$

- **VFF**

1. $(\lambda yxw.w)(\lambda xy.y)$
2. $\rightarrow_\beta \lambda xw.w$
3. $\rightarrow_\alpha \mathbf{F}$

- **FVV**

1. $(\lambda y.y)(\lambda xy.x)$
2. $\rightarrow_\beta \lambda xy.x$
3. $\equiv \mathbf{V}$

- **FVF**

1. $(\lambda y.y)(\lambda xy.y)$
2. $\rightarrow_\beta \lambda xy.y$
3. $\equiv \mathbf{F}$

- **FFV**

1. $(\lambda y.y)(\lambda xy.x)$
2. $\rightarrow_\beta \lambda xy.y$
3. $\equiv \mathbf{F}$

- **FFF**

1. $(\lambda y.y)(\lambda xy.y)$
2. $\rightarrow_\beta \lambda xy.x$
3. $\equiv \mathbf{V}$

Un vistazo superficial a estas dos tablas me dice que nos será de más ayuda la segunda, ya que a partir de valores de verdad se obtienen otros valores de verdad. Si suponemos que hay dos predicados P y Q los cuales pueden tomar los valores $\mathbf{V}$ o $\mathbf{F}$, podemos ver estas combinaciones como

- PPQ o QQQ
- PPQ o QQP
- PQP o QPQ
- PQQ o QPP

P	Q	PPP	PPQ	PQP	PQQ
F	**F**	**F**	**F**	**F**	**F**
F	**V**	**F**	**V**	**F**	**V**
V	**F**	**V**	**V**	**F**	**F**
V	**V**	**V**	**V**	**V**	**V**

Cuadro 2.1: Tabla de verdad de algunas combinaciones

Teniendo en cuenta que el conjunto de valores que puede tomar P es el mismo que puede tomar Q, solo consideramos una de las columnas. Se prosigue construyendo una tabla de verdad de estas cuatro posibilidades:

Por los valores de las tablas de verdad, encontramos que:

$$\wedge \equiv \lambda pq.pqp$$

$$\vee \equiv \lambda pq.ppq$$

Con este procedimiento logramos obtener las operaciones booleanas básicas de conjunción (asociado a la compuerta lógica AND) y disyunción (asociado a la compuerta lógica OR). Para poder completar la incrustación del álgebra booleana en el cálculo lambda necesitamos poder representar la operación de negación, la cual derivaremos con otra metodología.

El término lambda que representa la negación debe ser una abstracción que tenga como parámetro un predicado p y que regrese un término de dos parámetros que regrese el primero si p es falso y el segundo si p es verdadero. La forma de éste término es $\lambda pxy.?$ donde ? es un término por definir. Para encontrar este término incógnita hacemos la observación que si p es verdadero el primer argumento que le apliquemos será el resultado, mientras que si p es falso, el segundo argumento que le apliquemos será el resultado. Por lo tanto

$$(\lambda pxy.pyx)\mathbf{V} \rightarrow_\beta \lambda xy.y \equiv \mathbf{F}$$

$$(\lambda pxy.pyx)\mathbf{F} \rightarrow_\beta \lambda xy.x \equiv \mathbf{V}$$

Bajo este mismo procedimiento podemos encontrar otras operaciones útiles relacionadas con estructuras de control como condicionales. Para tener un término lambda de la condicional "Si P entonces M, de lo contrario N", donde P es un valor de verdad con esta codificación y M y N son términos cualesquiera, se construye como $\lambda pmn.?$, donde ? debe de resultar en m si p es verdadero y en n si p es falso. Por lo tanto

$$\text{Condicional} \equiv \lambda pmn.pmn$$

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

Cuadro 2.2: Numerales de Church

Este término debe ser aplicado como

$$(((\text{CondicionalPredicado})\text{Consecuente})\text{Contrario})$$

para ser β-reducido de manera correcta.

Queda pendiente expandir el trabajo de lógica proposicional que se ha desarrollado a la admisión de mas de dos valores de verdad, buscando ser consistente con la metodología de derivación de términos utilizada.

2.2. 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:

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

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 $\text{sucesor}(\text{sucesor}(\text{sucesor}(0)))$, así que

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

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 $\overline{n+1}$. Considerando la definición de $\bar{n} \equiv \lambda xy.x^n y$, lo que buscamos es una manera de agregarle una x a la composición en el cuerpo de $\bar{n}$ para obtener $\lambda xy.x^{n+1}y$. 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):

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

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

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

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

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

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:

$$\begin{aligned} \text{predecesor}\bar{0} &\twoheadrightarrow_\beta \bar{0} \\ \text{predecesor}\bar{n} &\twoheadrightarrow_\beta \overline{n-1} \end{aligned}$$

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:

$$\lambda xyz.zx^n y$$

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 $\overline{n+1}$ pudieramos obtener el cuerpo de $\bar{n}$ de la siguiente manera:

$$\begin{aligned} &(\lambda xyz.zx^n y)xy(\lambda a.a) \\ \to_\beta &(((\lambda yz.zx^n y)y)(\lambda a.a)) \\ \twoheadrightarrow_\beta &((\lambda z.zx^n y)(\lambda a.a)) \\ \to_\beta &(\lambda a.a)x^n y \\ \to_\beta &x^n y \end{aligned}$$

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 $\overline{n-1}$.

La estructura del sucesor sería:

$$\lambda nxy.?(\lambda a.a)$$

Donde ? debe ser tal que al β-reducirse resulte en un término con la forma $\lambda z.zx^{n-1}y$. 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 $\lambda w.wx^m y$ de tal manera que al aplicarle otro término nos resulte $\lambda w.wx^{m+1}y$. De esta manera al aplicar este otro término una y otra vez, resulte $\lambda w.wx^{n-1}y$ 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:

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

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:

$$\lambda gw.w(gx)$$

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

Para visualizar una manera de resolver el problema, es conveniente expresar cómo se verían las aplicaciones de $\lambda gw.w(gx)$ para un numeral de Church en particular. Si consideramos la aplicación de $\bar{4}(\lambda gw.w(gx))$:

$$\begin{aligned}
&\bar{4}(\lambda gw.w(gx))\\
\equiv&((\lambda xy.x^4y)(\lambda g.(\lambda w.(w(gx)))))\\
\equiv&((\lambda xy.x(x(x(xy))))(\lambda g.(\lambda w.(w(gx)))))\\
\to_\beta&(\lambda y.(\lambda g.(\lambda w.(w(gx))))((\lambda g.(\lambda w.(w(gx))))((\lambda g.(\lambda w.(w(gx))))((\lambda g.(\lambda w.(w(gx))))y))))
\end{aligned}$$

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

$$((\lambda g.(\lambda w.(w(gx))))((\lambda g.(\lambda w.(w(gx))))((\lambda g.(\lambda w.(w(gx))))(\lambda w.wy))))$$

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

$$((\lambda g.(\lambda w.(w(gx))))y) \twoheadrightarrow_\beta \lambda w.wy$$

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

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

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 $(\lambda g.(\lambda w.(w(gx))))$ a $\bar{n}$, este término resultante se aplica a $\lambda u.y$, β-reducir esta aplicación nos resulta $\lambda w.wx^{n-1}y$ 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:

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

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

$$\lambda mn.(\lambda xy.n\text{sucesor}m)$$

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

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

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

$$\lambda mnxy.n(mx)y$$

el cual aborda la idea de componer mn 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

$$\lambda mnxy.n\text{predecesor}m$$

.

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

$$\lambda mn.nm$$

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 $\bar{m}\equiv\lambda fg.f^{m}g$ y $\bar{n}\equiv\lambda xy.x^{n}y$, el resultado será $\bar{m^{n}}\equiv\lambda gy.g^{m^{n}}y$.

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:

$$\lambda fumn.n(\lambda w.fmw)u$$

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.

2.3. Procesos recursivos

Combinador Y, ordenes de evaluación, funciones recursivas.

$$\mathbf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

Presentar una breve introducción sobre los combinadores y hablar del combinador $\mathbf{Y}$ 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).

2.4. 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.

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

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

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).

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.