Universidad Nacional Autónoma de México

Facultad de Ciencias

El contenido computacional de la lógica clásica a través de los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$ con tipos

T E S I S

Que para obtener el título de:
Licenciado en Ciencias de la Computación

P r e s e n t a:

Noé Salomón Hernández Sánchez

Director de tesis:
Dr. Favio Ezequiel Miranda Perea

2012

Hoja de Datos del Jurado

1. Datos del alumno
   Hernández
   Sánchez
   Noé Salomón
   52 07 08 42
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Ciencias de la Computación
   303087692
2. Datos del tutor
   Dr
   Favio Ezequiel
   Miranda
   Perea
3. Datos del sinodal 1
   Dr
   Carlos
   Torres
   Alcaraz
4. Datos del sinodal 2
   Dra
   Atocha
   Aliseda
   Llera
5. Datos del sinodal 3
   Dr
   Francisco
   Hernández
   Quiroz
6. Datos del sinodal 4
   Dra
   Gabriela
   Campero
   Arena
7. Datos del trabajo escrito
   El contenido computacional de la lógica clásica a través de los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$ con tipos
   94 p
   2012

Agradecimientos

Este trabajo es fruto del esfuerzo, ayuda, colaboración e inspiración que me han brindado varios individuos e instituciones de quienes es merecida su mención.

Deseo comenzar expresando mi gratitud, que tiende a infinito, a Dios Padre y a nuestro Salvador Jesucristo, cuyo nombre en un principio era la raíz etimológica de la palabra lógica, es decir $\lambda\acute{o}\gamma o\varsigma$.

Mi gratitud es sin límite hacia mi familia, a mis padres y hermanos por su apoyo incondicional y valiosas enseñanzas; su cariño, cuidados y amor han sido una motivación constante a lo largo de todos mis estudios. Los axiomas, teoremas, fórmulas y descubrimientos que ellos han compartido conmigo son el mayor tesoro que guardo.

En un orden exponencial le doy las gracias a mi querida y entrañable Universidad Nacional Autónoma de México y a mi nada trivial Facultad de Ciencias por la magnífica formación académica que he recibido en estos años, por permitirme participar en programas de intercambio internacional y así poder constatar que la educación ofrecida por mi *alma mater* está al nivel de las mejores universidades.

Infinitas gracias a mi mentor y asesor, el Dr. Favio E. Miranda Perea, por mostrarme luz en estas artes oscuras de las cuales él es un gran conocedor.

Gracias sin cota alguna a mis compañeros y amigos: Juan Duron, Guillermina Flores, Yudith Beltrán y a la familia Sainoz Díaz. Su paciencia, interés, aprecio y comprensión fueron fundamentales al escribir este texto.

Un profundo agradecimiento para el Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica, PAPIIT-UNAM IN-108810, por la beca recibida sin la cual no habría sido posible el desarrollo y culminación de este trabajo de investigación.

Índice general

Introducción

Entre los múltiples *podcasts* que se encuentran en el sitio de la BBC existe uno titulado *A brief history of mathematics* [29], en donde el profesor Marcus du Sautoy comenta acerca de personajes importantes a lo largo de la historia de las matemáticas, sus descubrimientos y el impacto que producen sus resultados en otras disciplinas. Señala, por ejemplo, que el análisis de Joseph Fourier es el adecuado para describir ondas sonoras, la distribución de Gauss es la herramienta principal en estadística, la geometría no euclidiana de Riemann juega un papel primordial en la Teoría de la Relatividad de Einstein y Poincaré con su tratado del caos es contemplado al predecir el clima. Lamentablemente, no hace mención explícita de un descubrimiento en lógica matemática y de sus aplicaciones en otras áreas; así que se le podría plantear al profesor du Sautoy tomar el concepto de *deducción natural* introducido por Jaśkowski y Gentzen (véase [26]) con el propósito de modelar correctamente el pensamiento matemático, ya que más tarde se emplearía de forma crucial en la teoría de lenguajes de programación en Ciencias de la Computación. Esto es debido a que Curry y Howard observaron que las fórmulas de la lógica intuicionista corresponden a tipos[1] en el cálculo lambda con tipos simples, que las pruebas lógicas se asocian a programas y que una derivación lógica se relaciona con una derivación de tipos, todo esto conforma el *isomorfismo de Curry-Howard* [4, 5, 15]. Este descubrimiento es impactante, notable y sobresaliente, como bien dice Philip Wadler en [34],

> [...]*la deducción natural de Getzen y el cálculo lambda de Church están a la par en elegancia y trascendencia con la relatividad de Einstein y la física cuántica de Dirac. Además las matemáticas involucradas son mucho más sencillas*[...]

[1]Un tipo es una colección de entidades computacionales que comparten alguna propiedad en común.

Con el paso del tiempo se han formulado versiones del *isomorfismo de Curry-Howard* para distintas lógicas y formalismos matemáticos, entre sus variantes podemos mencionar que las *pruebas al estilo Hilbert* se asocian con la *lógica combinatoria simplemente tipificada*, el *cubo-*λ se relaciona con *los sistemas de tipos puros* y la lógica proposicional de segundo orden *Prop2* se corresponde con el cálculo lambda polimórfico de segundo orden $\lambda 2$ (véase [18, 31]). Sin embargo, hasta antes de la década de los noventas del siglo pasado parecía no existir *isomorfismo de Curry-Howard* para la lógica clásica. Griffin en [10] menciona que,

> [...]*Esta correspondencia ha sido restringida a la lógica constructiva porque se cree ampliamente que, en general, las pruebas clásicas carecen de contenido computacional*[...]

En el mismo artículo el autor resuelve esta cuestión estableciendo una tipificación para una extensión del cálculo lambda, conocido como cálculo $\lambda\mathcal{C}$, con el cual se cumple el *isomorfismo de Curry-Howard* para la lógica clásica. Tiempo después, Parigot (véase [25]) contribuye con la creación del cálculo $\lambda\mu$, cuya versión tipificada también se corresponde con la lógica clásica.

El propósito del presente trabajo es dar a conocer, a través de un estudio detallado de las aportaciones de Griffin y Parigot, que la lógica clásica posee una interpretación computacional dentro de la teoría de lenguajes de programación, por medio del *isomorfismo de Curry-Howard*; además, en este texto en ocasiones se ofrecerán demostraciones y argumentos propios que complementan los resultados de otros autores. Por un lado, Griffin recurre al cálculo $\lambda\mathcal{C}$ de Felleisen para formular una tipificación acorde a la lógica clásica; no obstante, esto propicia que a un término cerrado se le asigne el tipo void o $\bot$, lo cual es inadmisible. Griffin arregla esta situación al simular la evaluación en un ambiente en el que los tipos son válidos y las reducciones se pueden ejecutar. Por otra parte, Parigot extiende el cálculo lambda con términos μ de una manera elegante y conveniente para la definición de operadores de control, lo que le permite a su cálculo corresponderse con la lógica clásica. Finalmente, se apreciará cómo la traducción de Kolmogorov opera sobre el tipo de un término $\lambda\mu$, regresando el tipo de la expresión que resulta de transformar dicho término usando el estilo de programación CPS[2].

El contenido de este trabajo está organizado de la siguiente manera.

[2]Continuation Passing Style.

En el *Capítulo 1* se dan a conocer los conceptos necesarios para el desarrollo posterior. En primer lugar se define la *deducción natural*, que tomando como referente la negación, genera tres sistemas lógicos: *minimal*, *intuicionista* y *clásico*. Luego, se introduce el cálculo lambda puro (λ) de Alonzo Church. Para dar paso a la definición del cálculo lambda con tipos simples ($\lambda^{\to}$), donde los tipos son tipos atómicos o tipos función, además se indican sus propiedades más importantes. Se añaden después a $\lambda^{\to}$ los tipos suma y producto para obtener el cálculo $\lambda^{\to,\bot,+,\times}$. Entonces, se presenta el *isomorfismo de Curry-Howard* para $\lambda^{\to,\bot,+,\times}$ y la lógica intuicionista con conectivos $\to, \bot, \vee, \wedge$.

El *Cápitulo 2* empieza con una discusión de la relevancia que los operadores de control tienen en el desarrollo de programas eficientes, en particular mencionamos a las continuaciones como la herramienta fundamental para definir saltos en la secuencia de evaluación. Proseguimos a estudiar el cálculo $\lambda\mathcal{C}$ de Felleisen, sus nuevos operadores, $\mathcal{C}$ y $\mathcal{A}$, y su sistema de reducciones. Luego, se definen algunos operadores de control mediante términos de $\lambda\mathcal{C}$. Enseguida se obtienen las reglas de tipificación dadas por Griffin para este cálculo y se establece el *isomorfismo de Curry-Howard*. El capítulo termina definiendo a partir de $\mathcal{C}$ las operaciones que incorporan los tipos producto y suma.

El cálculo $\lambda\mu$ de Parigot es introducido en el *Capítulo 3*. Comenzamos proporcionando su definición, es decir, se muestran su gramática, reducciones y reglas de tipificación. Se explica a detalle cómo es que este cálculo se corresponde con la lógica clásica, siendo válido el *isomorfismo de Curry-Howard*. Se indagan las propiedades que cumple y los operadores de control que pueden ser definidos en términos $\lambda\mu$. Por último, se definen dentro del cálculo $\lambda\mu$ las operaciones que introducen los tipos producto y suma, análogamente a como se hizo para el cálculo $\lambda\mathcal{C}$.

Finalmente, en el *Capítulo 4* nos ocupamos de algunas traducciones interesantes. Primero nos enfocamos en la *traducción de Kolmogorov* y después en la transformación CPS de términos del cálculo $\lambda\mu$ a términos $\lambda^{\to}$. Observamos que estas dos traducciones interactúan asombrosamente ya que al transformar términos del cálculo $\lambda\mu$ usando CPS sus tipos cambian siguiendo la *traducción de Kolmogorov*. Para finalizar, mostramos que la transformación CPS, aplicada a términos $\lambda\mu$ restringidos, preserva la igualdad de expresiones.

Para una mejor asimilación de los temas aquí expuestos se recomienda al lector haber cursado las asignaturas de Análisis Lógico y Lenguajes de

Programación y sus Paradigmas, pertenecientes a la carrera de Ciencias de la Computación. En estos cursos tal vez se haya presentado una primera aproximación al *isomorfismo de Curry-Howard* para la lógica intuicionista y el cálculo lambda con tipos simples.

Capítulo 1

Preliminares

La lógica y la computación guardan una relación muy estrecha. En diversas áreas de las Ciencias de la Computación podemos comprobar esta afirmación, por ejemplo: en las bases de datos, donde encontramos el cálculo relacional; en la inteligencia artificial, al tratar de representar el conocimiento usando el lenguaje de la programación lógica; en los circuitos digitales, empleando compuertas lógicas; en la complejidad computacional, con el famoso problema SAT como ejemplo de problema NP-completo y, por supuesto, en los lenguajes de programación. Gran parte del fundamento teórico de lenguajes de programación es resultado de las aportaciones obtenidas de la lógica. Esto es evidente al definir la sintáxis concreta y abstracta, la semántica estática y dinámica, al introducir funciones y recursión, así como al considerar un sistema de tipos[3] en el lenguaje.

En la década de los años 1930 surgieron importantes aportaciones, tanto en lógica matemática como en los fundamentos de lenguajes de programación. Por una parte, en [26] se aclara que Stanislaw Jaśkowski y Gerhard Gentzen publicaron de manera independiente artículos que dieron origen a una nueva forma de deducción en lógica. Ambos pretendían encontrar un sistema que se aproximara de manera más fiel al razonamiento que en realidad se emplea en pruebas matemáticas, dando lugar a la deducción natural. Mientras tanto, Alonzo Church introducía el cálculo lambda como una manera de formalizar el concepto de computabilidad (véase [28]). Su versión es llamada cálculo lambda puro y, aunque originalmente su propósito era uno distinto, su cálculo

[3]Un sistema de tipos es un método sintáctico bien definido que sirve para evitar la presencia de ciertos comportamientos indeseables en los programas, al clasificar los términos involucrados de acuerdo a la clase de valores que éstos calculan.

se convirtió en una de las herramientas esenciales para el estudio formal de lenguajes de programación.

Tiempo después surgiría un resultado que entrelaza el sistema de deducción natural intuicionista, donde las pruebas son constructivas, y el cálculo lambda con tipos simples en el que a los términos del cálculo lambda puro se les añade un mecanismo de asignación de tipos. Esta relación es conocida como isomorfismo de *Curry-Howard* o *fórmulas-como-tipos*.

1.1. Sistemas de Deducción Natural

Los sistemas de deducción natural tienen como principal característica la definición de dos reglas para sus conectivos lógicos: introducción y eliminación, dichas reglas de inferencia son reconocidas y aceptadas como *naturales* o intuitivas en el proceso de pensamiento, es por este motivo que se les conoce como sistemas de deducción natural; otra propiedad importante en estos sistemas es que el uso de reglas de inferencia tiene una mayor ponderación que la definición de axiomas. Acerca de la deducción natural encontramos en la literatura la opinión de diversos autores; por ejemplo, Kalish y Montague mencionan que,

> [...]*(estos sistemas) se dicen que emplean deducción natural y, como indica su nombre, tienen el propósito de reflejar formas intuitivas de razonamiento*[...] (Veáse [26]).

Mientras que Chellas comenta,

> [...]*debido a que las reglas de inferencia tienen un parecido cercano a los patrones de razonamiento encontrados en el lenguaje natural, el sistema de deducción es de una clase llamada deducción natural*[...] (Veáse [26]).

Antes de profundizar más en los sistemas de deducción natural debemos ofrecer algunas definiciones.

Si $\mathsf{VProp} = \{p, q, r, \ldots\}$ es un conjunto infinito numerable de variables proposicionales, entonces el conjunto de *fórmulas* de la lógica proposicional, denotado por FProp, se define como:

- $\mathsf{VProp} \subseteq \mathsf{FProp}$
- $\bot \in \mathsf{FProp}$

- Si $\varphi, \psi \in \mathsf{FProp}$, entonces $(\varphi \to \psi), (\varphi \vee \psi), (\varphi \wedge \psi) \in \mathsf{FProp}$

Dada esta definición notemos lo siguiente:

- Para referirnos a fórmulas lógicas utilizaremos las letras griegas φ, ψ, etc.
- El conectivo de la negación no figura como constructor de fórmulas ya que puede ser definido a través de la implicación, como se verá más adelante.
- La implicación se asocia a la derecha, es decir, $\varphi \to \psi \to \chi$ se entiende como $\varphi \to (\psi \to \chi)$.
- La precedencia de operadores de mayor a menor es $\wedge, \vee, \to$ sin distinción entre $\wedge, \vee$. Por lo que $\varphi \wedge \psi \to \chi$ significa $(\varphi \wedge \psi) \to \chi$.

Ahora bien, introducimos los contextos lógicos, simbolizados por Γ, Δ, etc., que no son más que conjuntos finitos de fórmulas $\{\varphi_1, \ldots, \varphi_n\}$. Al escribir Γ, Δ nos referimos a $\Gamma \cup \Delta$. Similarmente, la notación Γ, φ indica $\Gamma \cup \{\varphi\}$ con $\varphi \notin \Gamma$. Definimos un secuente como una expresión de la forma $\Gamma \vdash \varphi$, con Γ contexto y φ fórmula. En particular, $\vdash \varphi$ se refiere a $\varnothing \vdash \varphi$. El secuente $\Gamma \vdash \varphi$ sirve para definir la derivabilidad de φ tomando como hipótesis las fórmulas en Γ, la definición formal de derivabilidad requiere de reglas de inferencia entre secuentes, las cuales tienen la forma:

$$\frac{\Gamma_1 \vdash \varphi_1 \quad \Gamma_2 \vdash \varphi_2 \quad \ldots \quad \Gamma_n \vdash \varphi_n}{\Gamma \vdash \varphi}\,(Nombre)$$

Esto nos indica que si los secuentes que forman las premisas, esto es $\Gamma_1 \vdash \varphi_1$, $\Gamma_2 \vdash \varphi_2$, $\ldots$, $\Gamma_n \vdash \varphi_n$, son derivables, entonces también es derivable la conclusión $\Gamma \vdash \varphi$. Si no hay premisas, concluimos que $\Gamma \vdash \varphi$ siempre es derivable. Por ende, una derivación del secuente $\Gamma \vdash \varphi$ es un árbol finito de secuentes cumpliendo las condiciones siguientes:

- La etiqueta de la raíz es $\Gamma \vdash \varphi$.
- Todas las hojas son axiomas, es decir, secuentes de la forma $\Gamma, \varphi \vdash \varphi$.
- La etiqueta de cada nodo padre se obtiene de las etiquetas de los nodos hijos usando una de las reglas de inferencia que conforman el sistema de deducción natural.

A menos que se indique lo contrario, para las reglas de inferencia que se dan en este capítulo se considerará un fragmento de los conectivos de la lógica proposicional, únicamente implicación ($\rightarrow$) y falsedad ($\bot$).

Notemos que la negación es uno de los conectivos lógicos más importantes. Su relevancia es tal que la manera en que es definida y tratada marca la pauta para generar distintos sistemas de deducción natural, los cuales pueden ser *minimal*, *intuicionista* o *clásico*. Para los dos últimos sistemas se tienen sus cálculos correspondientes, esto es, el cálculo proposicional intuicionista, denotado por NJ, y el cálculo proposicional clásico, representado por NK. Analicemos a detalle cada uno de estos tres sistemas.

1.1.1. Lógica Minimal

Este sistema de deducción natural fue propuesto en 1936 por I. Johansson, su peculiaridad es definir a la negación como $\neg\varphi =_{def} \varphi \rightarrow \bot$. Recordemos que únicamente empleamos los conectivos $\rightarrow$ y $\bot$ para definir nuestros sistemas lógicos. Para la derivación en lógica minimal utilizamos el símbolo $\vdash_m$, sus reglas de inferencia son las siguientes:

$$\frac{}{\Gamma, \varphi \vdash_m \varphi}\ (Ax) \qquad \frac{\Gamma, \varphi \vdash_m \psi}{\Gamma \vdash_m \varphi \rightarrow \psi}\ (\rightarrow I) \qquad \frac{\Gamma \vdash_m \varphi \rightarrow \psi \quad \Gamma \vdash_m \varphi}{\Gamma \vdash_m \psi}\ (\rightarrow E)$$

Reglas de deducción natural minimal.

Para ejemplificar el uso de estas reglas podemos mostrar que la inferencia $\varphi \rightarrow \psi \vdash_m \neg\psi \rightarrow \neg\varphi$ es derivable. Usando el hecho de que $\neg\varphi$ quiere decir $\varphi \rightarrow \bot$ y empleando la regla $(\rightarrow I)$ dos veces, basta mostrar que $\varphi \rightarrow \psi, \psi \rightarrow \bot, \varphi \vdash_m \bot$. El ocupar la regla $(\rightarrow I)$ para reescribir el secuente a derivar es un ardid que se usará en repetidas ocasiones a lo largo de este trabajo.

La derivación es la siguiente, tomando $\Gamma = \{\varphi \rightarrow \psi, \psi \rightarrow \bot, \varphi\}$,

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_m \varphi \rightarrow \psi}\ (Ax) \quad \dfrac{}{\Gamma \vdash_m \varphi}\ (Ax)}{\Gamma \vdash_m \psi}\ (\rightarrow E) \quad \dfrac{}{\Gamma \vdash_m \psi \rightarrow \bot}\ (Ax)}{\Gamma \vdash_m \bot}\ (\rightarrow E)$$

Como segundo ejemplo se dará la derivación del secuente $\vdash_m \varphi \rightarrow \neg\neg\varphi$, para lo cual es suficiente mostrar que $\varphi, \varphi \rightarrow \bot \vdash_m \bot$. Tomando $\Delta = \{\varphi, \varphi \rightarrow \bot\}$ se tiene,

$$\dfrac{\dfrac{}{\Delta \vdash_m \varphi}\,(Ax) \quad \dfrac{}{\Delta \vdash_m \varphi \to \bot}\,(Ax)}{\Delta \vdash_m \bot}\,(\to E)$$

De esto se observa que algunas propiedades de la negación, como lo es la introducción de la doble negación $\vdash_m \varphi \to \neg\neg\varphi$, son derivables en la lógica minimal aun cuando no hay reglas particulares para ella.

1.1.2. Lógica Intuicionista

Uno de sus fundadores fue L. E. J. Brouwer con su desarrollo de una matemática intuicionista en 1907. La lógica intuicionista se concibió años después como una extensión de la lógica minimal añadiendo la regla de eliminación del $\bot$ o *ex-falso-quodlibet (EFQ)*: $\Gamma \vdash \bot$ implica $\Gamma \vdash \varphi$; en otras palabras, si el sistema es inconsistente, entonces es posible derivar cualquier fórmula φ del sistema.

Denotamos con $\vdash_i$ la relación de derivabilidad en este sistema y con $NJ(\to, \bot)$ el cálculo proposicional intuicionista donde sólo figuran la implicación y la falsedad.

$$\dfrac{}{\Gamma, \varphi \vdash_i \varphi}\,(Ax) \qquad \dfrac{\Gamma \vdash_i \bot}{\Gamma \vdash_i \varphi}\,(EFQ) \qquad \dfrac{\Gamma, \varphi \vdash_i \psi}{\Gamma \vdash_i \varphi \to \psi}\,(\to I)$$

$$\dfrac{\Gamma \vdash_i \varphi \to \psi \quad \Gamma \vdash_i \varphi}{\Gamma \vdash_i \psi}\,(\to E)$$

Reglas del cálculo proposicional intuicionista, $NJ(\to, \bot)$.

Es fácil ver que toda fórmula derivable en la lógica minimal lo es en la lógica intuicionista, ya que las tres reglas de la lógica minimal figuran en $NJ(\to, \bot)$.

Todas las fórmulas generadas a partir de las reglas de la lógica intuicionista son **constructivas**, es decir, cuentan con una derivación directa. Una fórmula en esta lógica es válida o verdadera si podemos construirle una prueba de acuerdo a la interpretación BHK(véase 1.3.2). En este sentido debemos mencionar que la fórmula del tercero excluido, $\varphi \vee \neg\varphi$, no es válida. Esta fórmula surgió al describir ciertas situaciones finitas en las que es cierta, pero estudiosos de la lógica pusieron en duda su validez al contemplar su generalización al infinito. Por ejemplo, en el artículo títulado *Intuitionistic Logic* (véase [23]), Joan Moschovakis afirma que:

> [...]*si* x *y* y *son números naturales* $0, 1, 2, \ldots$ *y el predicado* B(x) *quiere decir que existe un número* $\mathrm{y} > \mathrm{x}$ *tal que* y *y* $\mathrm{y} + 2$ *son ambos números primos, entonces no tenemos método alguno para determinar si* B(x) *es verdadero o falso para una* x *arbitraria, así* $\forall \mathrm{x}\,(\mathrm{B(x)} \vee \neg \mathrm{B(x)})$ *no se puede afirmar en el estado del conocimiento actual. Si* A *representa el enunciado* $\forall \mathrm{x}\,\mathrm{B(x)}$*, entonces* $(\mathrm{A} \vee \neg \mathrm{A})$ *no se puede afirmar porque ni* A *ni* $\neg$A *se han podido probar. De acuerdo a Brouwer la ley del tercero excluido era equivalente a una suposición* a priori *que afirma que todo problema en matemáticas tiene solución*[...]

Por consiguiente, la fórmula $\varphi \vee \neg\varphi$, que no satisface la propiedad disyuntiva[4], no es válida. De este hecho llegamos a que $\neg\neg\varphi \to \varphi$ tampoco es verdadera; curiosamente, ya hemos demostrado que la proposición $\varphi \to \neg\neg\varphi$ es cierta usando lógica minimal.

A continuación se muestra un ejemplo de derivación en lógica intuicionista, buscamos derivar $\vdash_i \varphi \to \neg\varphi \to \psi$. Si $\Gamma = \{\varphi, \varphi \to \bot\}$, basta dar la derivación para $\Gamma \vdash_i \psi$, la cual es,

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_i \varphi}\,(Ax) \quad \dfrac{}{\Gamma \vdash_i \varphi \to \bot}\,(Ax)}{\Gamma \vdash_i \bot}\,(\to E)}{\Gamma \vdash_i \psi}\,(EFQ)$$

1.1.3. Lógica Clásica

En lógica clásica los secuentes que tienen como conclusión a las siguientes fórmulas son derivables bajo un contexto vacío, aseveración que no es cierta para la lógica minimal ni la intuicionista.

Tercero excluido (TE) También conocido como *Tertium non datur*.

$$\varphi \vee \neg\varphi$$

Eliminación de la doble negación $(\neg\neg E)$

$$\neg\neg\varphi \to \varphi$$

[4]La propiedad disyuntiva dice que si $\Gamma \vdash \varphi \vee \psi$, entonces $\Gamma \vdash \varphi$ o $\Gamma \vdash \psi$.

Reducción al absurdo $(\neg RAA)$

$$(\neg\varphi \to \varphi) \to \varphi$$

Ley de Peirce (P)

$$((\varphi \to \psi) \to \varphi) \to \varphi$$

De manera que basta ignorar la regla de *ex-falso-quodlibet* del sistema intuicionista y añadir la regla de eliminación de la doble negación para formar el sistema de deducción natural para la lógica clásica, en donde la derivabilidad se representa con $\vdash_c$. Así, las reglas para el cálculo proposicional clásico para la implicación y la falsedad, representación conocida como $NK(\to, \bot)$, son,

$$\frac{}{\Gamma, \varphi \vdash_c \varphi}\ (Ax) \qquad \frac{\Gamma, \varphi \vdash_c \psi}{\Gamma \vdash_c \varphi \to \psi}\ (\to I) \qquad \frac{\Gamma \vdash_c \varphi \to \psi \quad \Gamma \vdash_c \varphi}{\Gamma \vdash_c \psi}\ (\to E)$$

$$\frac{\Gamma, \varphi \to \bot \vdash_c \bot}{\Gamma \vdash_c \varphi}\ (\neg\neg E)$$

Reglas del cálculo proposicional clásico, $NK(\to, \bot)$.

Como ejemplo de una derivación tenemos la regla de *Reducción al Absurdo* $\vdash_c (\neg\varphi \to \varphi) \to \varphi$. Usando que $\neg\varphi =_{def} \varphi \to \bot$ y tomando $\Gamma = \{\neg\varphi \to \varphi\}$, es suficiente probar que $\Gamma \vdash_c \varphi$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \to \bot \vdash_c \neg\varphi \to \varphi}\,(Ax) \quad \dfrac{}{\Gamma, \varphi \to \bot \vdash_c \neg\varphi}\,(Ax)}{\Gamma, \varphi \to \bot \vdash_c \varphi}\,(\to E) \quad \dfrac{}{\Gamma, \varphi \to \bot \vdash_c \varphi \to \bot}\,(Ax)}{\dfrac{\Gamma, \varphi \to \bot \vdash_c \bot}{\Gamma \vdash_c \varphi}\,(\neg\neg E)}\,(\to E)$$

Para simplificar las derivaciones en ocasiones recurriremos a la propiedad de la monotonía.

Proposición 1.1. *La propiedad de la monotonía es válida en la lógica clásica, es decir,*

$$\frac{\Gamma \vdash_c \varphi}{\Gamma, \psi \vdash_c \varphi} \qquad \text{(Monotonía)}$$

Esta propiedad también es válida en los sistemas de deducción natural minimal e intuicionista.

Demostración. La prueba es por inducción sobre la inferencia $\Gamma \vdash_c \varphi$.

Para probar la propiedad en la lógica minimal e intuicionista se procede por inducción sobre la inferencia lógica respectiva al sistema. □

Es de esperarse que todas aquellas fórmulas derivables en la lógica intuicionista lo sigan siendo en la lógica clásica. Esto queda formalmente estipulado por la proposición siguiente.

Proposición 1.2. *Si* $\Gamma \vdash_i \varphi$ *en* NJ*, entonces* $\Gamma \vdash_c \varphi$ *en* NK*.*

Demostración. Se hace por inducción sobre la derivación de $\Gamma \vdash_i \varphi$ en NJ [5].

La base de la inducción corresponde a la regla (Ax) de $NJ(\to, \bot)$ la cual aparece idéntica en $NK(\to, \bot)$, por lo que la base queda probada. Si la derivación ocupa las reglas $(\to I)$ o $(\to E)$, éstas también son válidas en NK.

El caso interesante es mostrar cuando la derivación termina con la regla *ex-falso-quodlibet*. Para este paso inductivo la hipótesis de inducción nos dice que la premisa de la regla es inferible, es decir, $\Gamma \vdash_c \bot$ es inferible. Debemos demostrar ahora que la conclusión de la regla es válida, i.e. debemos derivar el secuente $\Gamma \vdash_c \varphi$. Como suponemos que $\Gamma \vdash_c \bot$ es derivable, entonces por monotonía obtenemos $\Gamma, \neg\varphi \vdash_c \bot$, usando la regla $(\neg\neg E)$ concluimos que $\Gamma \vdash_c \varphi$ en NK. □

Como observación final notemos lo siguiente.

Proposición 1.3. *Los dos enunciados siguientes son válidos.*

I. *Al tomar la regla* ex-falso-quodlibet (EFQ) *y la* Ley de Peirce (P) *es posible derivar* $(\neg\neg E)$.

II. *De las reglas de* NK *se pueden derivar la regla* ex-falso-quodlibet *y la* Ley de Peirce*.*

Demostración. En [31] la prueba es muy concisa. El argumento siguiente es más detallado.

Recordemos las reglas correspondientes:

$$\frac{\Gamma \vdash_c \bot}{\Gamma \vdash_c \varphi}\,(EFQ) \qquad \frac{\Gamma, \varphi \to \psi \vdash_c \varphi}{\Gamma \vdash_c \varphi}\,(P)$$

[5] Otra manera de aplicar la inducción es sobre la altura del árbol de derivación; sin embargo, es totalmente válido aplicar la inducción sobre la derivación, como se empleó en esta demostración y como se aplicará en demostraciones posteriores.

La prueba de I consiste en suponer que $\Gamma, \varphi \to \bot \vdash_c \bot$ e inferir $\Gamma \vdash_c \varphi$. Podemos escribir nuestra suposición como,

$$\frac{}{\Gamma, \varphi \to \bot \vdash_c \bot}\,(@)$$

Entonces,

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \to \bot \vdash_c \bot}\,(@)}{\Gamma, \varphi \to \bot \vdash_c \varphi}\,(EFQ)}{\Gamma \vdash_c \varphi}\,(P)$$

Para II debemos inferir (EFQ) y (P) a partir de NK. La inferencia de (EFQ) fue hecha en la demostración de la proposición 1.2. La segunda se prueba usando $\Gamma, \varphi \to \psi \vdash_c \varphi$ como hipótesis y concluyendo que $\Gamma \vdash_c \varphi$. Expresamos nuestra hipótesis como,

$$\frac{}{\Gamma, \varphi \to \psi \vdash_c \varphi}\,(\#)$$

Observamos primero que

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \to \psi \vdash_c \varphi}\,(\#)}{\Gamma, \neg\varphi, \varphi \to \psi \vdash_c \varphi}\,\textit{(Monotonía)}}{\Gamma, \neg\varphi \vdash_c (\varphi \to \psi) \to \varphi}\,(\to I)$$

Luego,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \neg\varphi, \varphi, \neg\psi \vdash_c \neg\varphi}\,(Ax) \quad \dfrac{}{\Gamma, \neg\varphi, \varphi, \neg\psi \vdash_c \varphi}\,(Ax)}{\Gamma, \neg\varphi, \varphi, \neg\psi \vdash_c \bot}\,(\to E)}{\Gamma, \neg\varphi, \varphi \vdash_c \psi}\,(\neg\neg E)}{\Gamma, \neg\varphi \vdash_c \varphi \to \psi}\,(\to I)$$

Si a los dos últimos resultados $\Gamma, \neg\varphi \vdash_c (\varphi \to \psi) \to \varphi$ y $\Gamma, \neg\varphi \vdash_c \varphi \to \psi$ les aplicamos la regla $(\to E)$, entonces obtenemos $\Gamma, \neg\varphi \vdash_c \varphi$ $(\star)$. La última parte de la derivación es,

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \neg\varphi \vdash_c \varphi}\,(\star) \quad \dfrac{}{\Gamma, \neg\varphi \vdash_c \neg\varphi}\,(Ax)}{\Gamma, \neg\varphi \vdash_c \bot}\,(\to E)}{\Gamma \vdash_c \varphi}\,(\neg\neg E)$$

□

Se han presentado los sistemas lógicos minimal, intuicionista y clásico, en este texto nos ocuparemos principalmente de los sistemas intuicionista y clásico. Además, se ha apreciado que los secuentes derivables en la lógica minimal lo son también en la lógica intuicionista y que aquellos secuentes que se infieren en la lógica intuicionista son inferibles en la lógica clásica.

1.2. Cálculo lambda

Creado por Alonzo Church a principios de la década de los treinta con la finalidad de fundamentar la matemática y la idea de cómputo efectivo, el cálculo lambda es un sistema de reescritura de términos que consiste de una sola regla de reducción y de un único método para definir funciones, por lo que es considerado el *lenguaje de programación universal más pequeño*. Su universalidad radica en que cualquier función computable puede ser expresada y evaluada usando este cálculo. Su poder de cómputo es equivalente al de las máquinas de Turing; empero, el cálculo lambda hace énfasis en el uso de reglas de reducción o de simplificación y no se ocupa mucho de la máquina que en efecto las implementa, por lo que tiene un enfoque más relacionado al software que al hardware.

Por lo pronto, estudiaremos el cálculo lambda sin tipos, también conocido como cálculo lambda puro.

1.2.1. Cálculo lambda puro

Para definir términos o expresiones E correspondientes al cálculo lambda puro se utiliza la siguiente gramática,

$$E ::= x \mid \lambda x.E \mid EE$$

donde x pertenece a un conjunto de identificadores o variables, $\lambda x.E$ es una expresión llamada abstracción lambda que representa una función, y EE representa la aplicación de expresiones. Notemos que en el cálculo lambda puro toda expresión es una función.

En adelante, para referirnos a expresiones como $\lambda x.\lambda y.\lambda z.xyz$ simplemente escribiremos $\lambda xyz.xyz$, es decir, quitamos todas las presencias de λ conservando sólo la inicial. En situaciones como $(\lambda u.uv)(\lambda w.wx)(\lambda y.yz)$ acordamos que la asociación es hacia la izquierda, por consiguiente, la expresión ante-

rior se asocia $\big((\lambda u.uv)(\lambda w.wx)\big)(\lambda y.yz)$. En general, si se tiene lo siguiente $E_1E_2E_3 \ldots E_n$, la asociación correcta es $(\ldots((E_1E_2)E_3)\ldots)E_n$.

1.2.2. Variables libres y ligadas.

De la expresión $\lambda x.E$ notemos lo siguiente,

- La variable x, que se ubica entre λ y el punto, se conoce como *argumento* de la función.
- La expresión E, la cual aparece después del punto, es el *cuerpo* de la función.

Decimos que una variable está ligada cuando forma parte del cuerpo de una función que la tiene como argumento. Una variable está libre si aparece en una expresión en la cual dicha variable no figura como argumento. Así, en la expresión,

$$\lambda uvw.w(uv)$$

las variables u, v y w están ligadas. Mientras que en la aplicación,

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

se aprecia que el primer término no tiene variables libres ya que la x está ligada; en el segundo se tiene que la y está ligada pero la x no es visible o no está en el alcance de ningún λx, por lo que es una variable libre, esto se debe a que ambas funciones son independientes y las $x's$ que figuran en las dos expresiones son distintas.

La definición recursiva para variables libres de una expresión se presenta en seguida.

Definición 1.1. *El conjunto de variables libres FV de la expresión E se obtiene de la siguiente manera,*

- $FV(x) = x$
- $FV(\lambda x.E_1) = FV(E_1) \setminus \{x\}$
- $FV(E_1E_2) = FV(E_1) \cup FV(E_2)$

Si $FV(E) = \varnothing$, decimos que E es una expresión cerrada, un combinador o un programa.

1.2.3. Sustitución y α-equivalencia.

La operación de sustitución juega un papel vital en el cálculo λ ya que es utilizada para definir su semántica. La acción que efectúa es tomar una expresión E_0 y ponerla en lugar de la variable x en todas aquellas presencias libres de x en E, se denota como $E[x := E_0]$ o $E[E_0/x]$. Sin embargo, el uso descuidado de esta operación puede provocar que una variable que originalmente era libre se ligue, lo cual cambiaría el significado de la expresión. Consideremos la siguiente situación:

$$(\lambda xy.xyz)[z := xy] \tag{1.1}$$

si se realiza la sustitución obtendríamos como resultado,

$$\lambda xy.xy(xy)$$

lo cual liga la segunda presencia de x y de y; no obstante, estas variables estaban libres en la expresión de la que provienen. Así, el resultado obtenido es incorrecto. El error consiste en ligar variables que en un principio eran libres. Para corregir esto podemos, simplemente, cambiar el nombre de las variables ligadas en 1.1, esto produce la sustitución,

$$(\lambda uv.uvz)[z := xy]$$

lo que nos lleva al resultado correcto,

$$\lambda uv.uv(xy),$$

donde x y y siguen siendo libres, como lo eran antes de la sustitución. Este cambio de nombre es válido y da lugar a la definición de α-equivalencia.

Definición 1.2. *Dos expresiones E_0 y E_1 son α-equivalentes si la única diferencia entre ellas se da en los nombres de las variables ligadas, si es así escribimos $E_0 \equiv_\alpha E_1$.*

De esta definición se cumple, por ejemplo, que:

$$\lambda xy.xyy \equiv_\alpha \lambda uv.uvv$$

$$\lambda v.y(\lambda w.wz)v \equiv_\alpha \lambda a.y(\lambda b.bz)a$$

Ahora estamos en posibilidad de dar una definición precisa de la operación de sustitución.

Definición 1.3. *La sustitución en E de todas las presencias libres de la variable x por la expresión E_0, representada por $E[x := E_0]$, se define recursivamente sobre la estructura de E como:*

- $x[x := E_0] = E_0$
- $y[x := E_0] = y \quad si\ x \neq y$
- $(E_1E_2)[x := E_0] = (E_1[x := E_0])(E_2[x := E_0])$
- $(\lambda x.E)[x := E_0] = \lambda x.E$
- $(\lambda w.E)[x := E_0] = \lambda w.E[x := E_0] \quad si\ x \neq w\ y\ w \notin FV(E_0)$

Las condiciones que se requieren en el último punto sirven para evitar capturar variables libres; sin embargo, en el caso en que $w \in FV(E_0)$ usamos la α-equivalencia para crear una expresión en donde la w es renombrada de tal manera que los problemas de captura de variables libres de E_0 por w desaparecen.

Con el concepto de sustitución podemos definir la α-equivalencia del siguiente modo,

$$\lambda w.E \to_\alpha \lambda z.E[w := z] \quad z \notin FV(E)$$

1.2.4. Reducción Beta

Para dar la semántica del cálculo λ puro debemos evaluar las aplicaciones de la forma $(\lambda x.E)t$, es decir, aquellas en las que el lado izquierdo es una abstracción λ, dichas expresiones se conocen como β-*redex* (Reducible Expression), ya que es posible ejecutar una *reducción* $\beta(\to_\beta)$ sobre ellas, ésta consiste en,

$$(\lambda x.E)t \to_\beta E[x := t]$$

Definición 1.4. *La cerradura reflexiva y transitiva de $\to_\beta$ se denota como $\to_\beta^*$.*

Definición 1.5. *Una expresión es* β-normal *si no contiene algún* β-redex.

Definición 1.6. *Sean E y G dos expresiones, decimos que G es una forma β-normal de E si $E \to_\beta^* G$ y G es β-normal.*

El objetivo de la evaluación de E es llegar, a través de una serie de *reducciones* β a una expresión t tal que t es una *forma β-normal de E*. Observemos el siguiente ejemplo:

$$\begin{aligned}(\lambda fxy.fyx)(\lambda xy.yx)ab &\to_\beta (\lambda xy.(\lambda xy.yx)yx)ab\\ &\to_\alpha (\lambda xy.(\lambda xu.ux)yx)ab\\ &\to_\beta (\lambda xy.(\lambda u.uy)x)ab\\ &\to_\beta (\lambda xy.xy)ab\\ &\to_\beta (\lambda y.ay)b\\ &\to_\beta ab\\ &\nrightarrow_\beta\end{aligned}$$

Así, ab es una *forma β-normal de* $(\lambda fxy.fyx)(\lambda xy.yx)ab$.

Sin embargo, no toda expresión en el cálculo lambda puro tiene una *forma β-normal*. Si analizamos lo que pasa con $\omega\omega$, donde $\omega = \lambda x.xx$, entonces $\omega\omega \to_\beta (\lambda x.xx)\omega \to_\beta \omega\omega \to_\beta \cdots$. Por consiguiente, no se tiene una *forma β-normal* para dicha expresión.

1.2.5. Reducción Eta

Un término de la forma $\lambda x.Ex$, donde $x \notin FV(E)$ se conoce como η-redex. La reducción $\eta(\to_\eta)$ se define como,

$$\lambda x.Ex \to_\eta E \qquad \text{si } x \notin FV(E).$$

Definición 1.7. *La cerradura reflexiva y transitiva de $\to_\eta$ se denota como $\to_\eta^*$.*

Definición 1.8. *Un término es* η-normal *si no contiene algún* η-redex.

Definición 1.9. *Sean E y G dos expresiones, decimos que G es una forma η-normal de E si $E \to_\eta^* G$ y G es η-normal.*

Por ejemplo,

$$\lambda xy.fxy \to_\eta \lambda x.fx \to_\eta f$$

vemos que el término f es una forma η-normal de $\lambda xy.fxy$. Por otro lado, el término $\lambda xy.fyx$ por sí solo es η-normal.

El funcionamiento de la reducción η se explica a continuación, si una abstracción lambda desea pasar su argumento a otra función, como en $\lambda x.Ex$,

entonces la abstracción lambda es redundante y podemos disponer de ella, así $\lambda x.Ex \rightarrow_\eta E$. De esta manera, la reducción η no realiza simplificación alguna sino que elimina redundancia, esta es la razón por la que no es considerada una regla de cómputo.

1.2.6. Representación de booleanos y números naturales

En el cálculo lambda existen expresiones que representan elementos de tipo booleano y también términos que son vistos como números naturales, estas expresiones son:

- Booleanos
  - $\mathsf{true} := \lambda x.\lambda y.x$
  - $\mathsf{false} := \lambda x.\lambda y.y$
  - $\mathsf{test} := \lambda b.\lambda t.\lambda e.bte$

  Tomando estas definiciones las reducciones siguientes son ciertas,

  - $\mathsf{test\ true}\ e_1\ e_2 \rightarrow_\beta^* e_1$
  - $\mathsf{test\ false}\ e_1\ e_2 \rightarrow_\beta^* e_2$

  Así que el término test puede considerarse como el operador if...then...else
- Naturales(numerales de Church)
  - $\overline{0} := \lambda s.\lambda z.z$
  - $\overline{1} := \lambda s.\lambda z.sz$
  - $\overline{n} := \lambda s.\lambda z.\underbrace{s(\ldots(s}_{n\ veces}\ z)\ldots)$
  - $\mathsf{suma} := \lambda m.\lambda n.\lambda s.\lambda z.ms(nsz)$ tal que,

$$\forall n, m \in \mathbb{N}(\mathsf{suma}\ \overline{n}\ \overline{m} \rightarrow_\beta^* \overline{n+m})$$

Las expresiones que hemos exhibido para booleanos y números naturales se originan de ciertas definiciones lógicamente especificadas de las cuales no profundizaremos en este texto.

1.2.7. Cálculo lambda con tipos simples

En general, un tipo es una colección de entidades computacionales que comparten alguna propiedad en común. Los tipos en lenguajes de programación juegan un rol importante, sirven, entre otras cosas, para formar expresiones sintácticamente correctas, ofrecen seguridad a los programas que los utilizan y funcionan como un medio para entender mejor el código. Consideremos una versión tipificada del cálculo lambda puro, llamada cálculo lambda con tipos simples ($\lambda^{\rightarrow}$). El conjunto de tipos simples T está definido por:

$$\mathsf{T} ::= \tau \,|\, \mathsf{T} \rightarrow \mathsf{T}$$

donde τ es una colección finita de tipos básicos o atómicos y $\mathsf{T} \rightarrow \mathsf{T}$ es el tipo función. Esta es una definición recursiva en la que se tiene una infinidad de tipos. Siendo $\mathsf{T}_1, \ldots, \mathsf{T}_{n-1}, \mathsf{T}_n$ tipos entonces el tipo función $\mathsf{T}_1 \rightarrow \ldots \rightarrow \mathsf{T}_{n-1} \rightarrow \mathsf{T}_n$ se asocia a la derecha, es decir, la colocación adecuada de los paréntesis es $\mathsf{T}_1 \rightarrow (\ldots \rightarrow (\mathsf{T}_{n-1} \rightarrow \mathsf{T}_n)\ldots)$. Cuando la expresión E es de tipo T escribimos $E : \mathsf{T}$.

Existen dos maneras en las que se pueden añadir tipos al cálculo lambda, *a la Curry* y *a la Church*. La diferencia estriba en que un estilo menciona explícitamente el tipo de la variable y el otro no, de modo que en el estilo *a la Curry* encontramos la expresión $\lambda x.E : \mathsf{T} \rightarrow \mathsf{S}$ en donde el tipo de la x no se conoce; mientras que en el estilo *a la Church* el tipo de la variable x se hace explícito en la abstracción lambda $\lambda x : \mathsf{T}.E : \mathsf{T} \rightarrow \mathsf{S}$. En el cálculo lambda con tipos simples que trata esta sección se adoptará un estilo *a la Curry*.

Como se mencionó, los tipos permiten la construcción de expresiones sintácticamente correctas, esto se logra estableciendo reglas de tipificación las cuales asignan tipos a términos de acuerdo a las relaciones permitidas por el lenguaje. Si el tipo de una expresión es derivable a partir de esas reglas entonces la expresión está bien formada y es aceptada. Emplearemos uso de los contextos de tipos, representados por Γ, Δ, etc., conjunto finito de variables y sus tipos respectivos, i.e. $\Gamma = \{x_1 : \mathsf{T_1}, \ldots, x_n : \mathsf{T_n}\}$. Así, $\Gamma(x) = \mathsf{T}$ significa que $(x : \mathsf{T}) \in \Gamma$. La relación de tipificación $\Gamma \vdash E : \mathsf{T}$ indica que a la expresión E se le asigna el tipo T bajo el contexto de tipos Γ. De manera que las reglas de tipificación tienen la forma:

$$\frac{\Gamma_1 \vdash E_1 : \mathsf{T}_1 \quad \Gamma_2 \vdash E_2 : \mathsf{T}_2 \quad \ldots \quad \Gamma_n \vdash E_n : \mathsf{T}_n}{\Gamma \vdash E : \mathsf{T}} \, (Nombre)$$

Se entiende del siguiente modo: si las tipificaciones que forman las premisas, $\Gamma_1 \vdash E_1 : \mathsf{T}_1, \Gamma_2 \vdash E_2 : \mathsf{T}_2, \ldots, \Gamma_n \vdash E_n : \mathsf{T}_n$, son derivables, entonces también lo es la tipificación en la conclusión, $\Gamma \vdash E : \mathsf{T}$. Si en la regla no aparecen premisas, entonces la conclusión $\Gamma \vdash E : \mathsf{T}$ siempre es derivable.

La derivabilidad de tipos para el cálculo lambda con tipos simples sigue las reglas de tipificación que se muestran a continuación,

$$\frac{}{\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}}\ (RVar) \qquad \frac{\Gamma, x : \mathsf{T} \vdash E : \mathsf{S}}{\Gamma \vdash \lambda x.E : \mathsf{T} \to \mathsf{S}}\ (RFun)$$

$$\frac{\Gamma \vdash E_1 : \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash E_2 : \mathsf{T}}{\Gamma \vdash E_1 E_2 : \mathsf{S}}\ (RApp)$$

Reglas de tipificación para $\lambda^{\to}$.

Realicemos una derivación de tipos para ilustrar el uso de las reglas. Verifiquemos que $\vdash \lambda xy.xyy : (A \to A \to B) \to A \to B$, para este fin usaremos el contexto de tipos $\Gamma = \{x : A \to A \to B, y : A\}$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : A \to A \to B}\ (RVar) \quad \dfrac{}{\Gamma \vdash y : A}\ (RVar)}{\Gamma \vdash xy : A \to B}\ (RApp) \quad \dfrac{}{\Gamma \vdash y : A}\ (RVar)}{\Gamma \vdash xyy : B}\ (RApp)}{x : A \to A \to B \vdash \lambda y.xyy : A \to B}\ (RFun)}{\vdash \lambda xy.xyy : (A \to A \to B) \to A \to B}\ (RFun)$$

Ahora que contamos con tipos y sirviéndonos de la idea de sustitución los siguientes resultados son válidos,

Proposición 1.4. *Si $\Gamma \vdash M : \mathsf{T}$ y $\Gamma(x) = \Gamma'(x)$ para toda $x \in FV(M)$ entonces $\Gamma' \vdash M : \mathsf{T}$.*

Demostración. La demostración se encuentra en [31]. □

Proposición 1.5. *Si $\Gamma, x : \mathsf{T} \vdash E : \mathsf{S}$ y $\Gamma \vdash E' : \mathsf{T}$ entonces $\Gamma \vdash E[x := E'] : \mathsf{S}$.*

Demostración. Varios autores mencionan simplemente que la prueba es por inducción sobre los términos E del cálculo $\lambda^{\to}$ (véase [20] y [31]). Aquí se tiene la prueba completa, presentando el caso base, la hipótesis de inducción y el paso inductivo.

- *Base de inducción.* Para el término $E = x_0$ tenemos $\overbrace{\Gamma', x_0 : \mathsf{S}}^{\Gamma}, x : \mathsf{T} \vdash x_0 : \mathsf{S}$, sabemos además que $\Gamma \vdash E' : \mathsf{T}$. Deseamos conocer el tipo de $x_0[x := E']$ bajo el contexto Γ.

  - Si $x \neq x_0$ entonces la sustitución nos da $x_0[x := E'] = x_0$, luego $\Gamma \vdash x_0[x := E'] : \mathsf{S}$.
  - Si $x = x_0$, tenemos $\mathsf{T} = \mathsf{S}$ ya que los contextos son conjuntos. Así la sustitución es $x_0[x := E'] = E'$ y $\Gamma \vdash x_0[x := E'] : \mathsf{S}$.

  Vemos que en ambos casos se cumple la proposición.

- *Hipótesis de inducción.* La proposición es válida para E_0 y E_1, es decir, si $\Gamma, x : \mathsf{T} \vdash E_0 : \mathsf{S}_0$ y $\Gamma, x : \mathsf{T} \vdash E_1 : \mathsf{S}_1$ con $\Gamma \vdash E' : \mathsf{T}$ entonces $\Gamma \vdash E_0[x := E'] : \mathsf{S}_0$ y $\Gamma \vdash E_1[x := E'] : \mathsf{S}_1$

- *Paso inductivo.*

  - Para el caso $E = \lambda x_0.E_0$ suponemos $\Gamma, x : \mathsf{T} \vdash \lambda x_0.E_0 : \mathsf{S}_0 \to \mathsf{S}_1$, $\Gamma \vdash E' : \mathsf{T}$ y $x_0 \neq x$ ya que si $x_0 = x$ el resultado es trivial usando la proposición 1.4. Por *H.I.* tenemos $\Gamma, x_0 : \mathsf{S}_0 \vdash E_0[x := E'] : \mathsf{S}_1$. Por la regla $(RFun)$ se satisface $\Gamma \vdash \lambda x_0.(E_0[x := E']) : \mathsf{S}_0 \to \mathsf{S}_1$, de aquí que,
    $$\Gamma \vdash (\lambda x_0.E_0)[x := E'] : \mathsf{S}_0 \to \mathsf{S}_1$$
  - Si $E = E_0E_1$ suponemos que $\Gamma, x : \mathsf{T} \vdash E_0E_1 : \mathsf{S}_1$ y $\Gamma \vdash E' : \mathsf{T}$. Al aplicar la hipótesis de inducción a E_0 y E_1 tenemos,
    $$\Gamma \vdash E_0[x := E'] : \mathsf{S}_0 \to \mathsf{S}_1 \qquad \Gamma \vdash E_1[x := E'] : \mathsf{S}_0$$
    De donde al aplicar la regla $(RApp)$ conseguimos,
    $$\Gamma \vdash (E_0[x := E'])(E_1[x := E']) : \mathsf{S}_1$$
    que puede considerarse como,
    $$\Gamma \vdash (E_0E_1)[x := E'] : \mathsf{S}_1$$

□

Para el cálculo lambda con tipos simples existen tres propiedades de gran importancia, que relacionan sintaxis con semántica, esto es, expresiones correctamente tipificadas con pasos de la evaluación.

Proposición 1.6. (Reducción del sujeto) *Si* $E_1 \to_\beta E_2$ *y* $\Gamma \vdash E_1 : \mathsf{T}$ *entonces* $\Gamma \vdash E_2 : \mathsf{T}$.

En la tipificación $E : \mathsf{T}$, E es conocido como sujeto y T como predicado, por eso el nombre de reducción del sujeto para esta propiedad.

Demostración. Inducción sobre $E_1 \to_\beta E_2$ □

Proposición 1.7. (Normalización fuerte) *No se tiene secuencia infinita* $E \to_\beta E_1 \to_\beta E_2 \to_\beta \ldots$ *para el término* E *con* $\vdash E : \mathsf{T}$.

Demostración. La prueba no es trivial y no se mostrará aquí. Pero puede consultarse en [9]. □

Proposición 1.8. (Confluencia o propiedad de Church-Rosser) *Dado que* $\vdash E : \mathsf{T}$, *si* $E \to_\beta^* E_1$ *y* $E \to_\beta^* E_2$ *entonces existe* E_3 *tal que* $E_1 \to_\beta^* E_3$ *y* $E_2 \to_\beta^* E_3$.

Demostración. Las demostración no es trivial y está fuera del alcance de este trabajo. Se puede encontrar en [2]. □

La propiedad de confluencia es importante ya que nos indica que el resultado de reducir una expresión $\lambda^\to$ será al final el mismo, aunque el orden en que se simplifiquen los redexes en la expresión varie y genere expresiones intermedias distintas.

1.2.8. Tipos sumas y productos

El cálculo lambda con tipos simples $\lambda^\to$ es útil como fundamento del estudio de lenguajes de programación; no obstante, estamos interesados en proveer de mayor expresividad y nuevas operaciones a los lenguajes. Es por eso que extendemos los tipos presentes en $\lambda^\to$ con el tipo suma ($\mathsf{T}+\mathsf{S}$), el tipo void, el tipo producto ($\mathsf{T} \times \mathsf{S}$), el tipo unit y el tipo Nat. Un elemento de tipo $\mathsf{T}+\mathsf{S}$ es de tipo T o bien de tipo S, en el que una etiqueta muestra el tipo de donde proviene. Definimos el tipo void, denotado también por $\bot$, como el tipo suma donde la elección es entre cero candidatos representando el tipo vacío. Un valor correspondiente al tipo $\mathsf{T} \times \mathsf{S}$ es un par ordenado donde el término

en la primera entrada es de tipo T y el de la segunda de tipo S. El tipo $\mathbf{1}$ o tipo unit[6] cuenta solo con un elemento, un valor trivial que servirá cuando una expresión regresa un valor irrelevante. Por último, introducimos el tipo Nat para representar a los números naturales, además será el tipo del valor que resulta de la operación aritmética de adición.

El nuevo cálculo generado es $\lambda^{\to,\bot,+,\times}$. Los posibles tipos se extienden de la siguiente manera,

$$\mathsf{T} := \mathsf{T} + \mathsf{T} \mid \mathsf{void} \mid \mathsf{T} \times \mathsf{T} \mid \mathsf{unit} \mid \mathsf{T} \to \mathsf{T} \mid \mathsf{Nat}$$

Se extienden las expresiones reconocidas por el lenguaje para abarcar los siguientes valores y operadores,

- Constantes numéricas $0, 1, \ldots$ de tipo Nat.
- Operador aritmético de adición que toma como argumento dos números naturales x y y, regresando el valor $x + y$. Se denota por $\mathsf{suma}(x, y)$.
- $\mathsf{Abort}_{\mathsf{T}}\,(E)$ ocasiona la terminación del programa con tipo de regreso T.
- $\mathsf{inl}^{\mathsf{T}+\mathsf{S}}(E_1)$ y $\mathsf{inr}^{\mathsf{T}+\mathsf{S}}(E_2)$, donde $E_1 : \mathsf{T}$ y $E_2 : \mathsf{S}$. Ambas *inyecciones* generan términos con tipo $\mathsf{T}+\mathsf{S}$.
- $\mathsf{case}\ E\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\}$ representa el análisis por casos.
- El valor $*$, único elemento presente en el tipo unit.
- $\mathsf{fst}\ E$ y $\mathsf{snd}\ E$, regresan el primer y segundo elemento del par ordenado E, respectivamente.
- $\langle E_1, E_2 \rangle$ término que representa un par ordenado, su tipo es $\mathsf{T} \times \mathsf{S}$.

Presentamos a continuación las reglas de tipificación para $\lambda^{\to,\bot,+,\times}$.

$$\frac{}{\Gamma \vdash n\ :\ \mathsf{Nat}}\ (RNat) \qquad \frac{\Gamma \vdash x\ :\ \mathsf{Nat} \quad \Gamma \vdash y\ :\ \mathsf{Nat}}{\Gamma \vdash \mathsf{suma}(x, y)\ :\ \mathsf{Nat}}\ (RSuma)$$

$$\frac{}{\Gamma, x\ :\ \mathsf{T} \vdash x\ :\ \mathsf{T}}\ (RVar) \qquad \frac{\Gamma, x\ :\ \mathsf{T} \vdash E\ :\ \mathsf{S}}{\Gamma \vdash \lambda x.E\ :\ \mathsf{T} \to \mathsf{S}}\ (RFun)$$

[6] El tipo void presente en lenguajes de programación, como son Java y C, es en realidad el tipo unit.

$$\frac{\Gamma \vdash E_1 \,:\, \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash E_2 \,:\, \mathsf{T}}{\Gamma \vdash E_1 E_2 \,:\, \mathsf{S}}\ (RApp)$$

$$\frac{\Gamma \vdash E_1 \,:\, \mathsf{T} \quad \Gamma \vdash E_2 \,:\, \mathsf{S}}{\Gamma \vdash \langle E_1, E_2 \rangle \,:\, \mathsf{T} \times \mathsf{S}}\ (RProd) \qquad \frac{\Gamma \vdash E \,:\, \mathsf{T} \times \mathsf{S}}{\Gamma \vdash \mathsf{fst}\, E \,:\, \mathsf{T}}\ (RFst)$$

$$\frac{\Gamma \vdash E \,:\, \mathsf{T} \times \mathsf{S}}{\Gamma \vdash \mathsf{snd}\, E \,:\, \mathsf{S}}\ (RSnd)$$

$$\frac{\Gamma \vdash E \,:\, \mathsf{T}}{\Gamma \vdash \mathsf{inl}^{\mathsf{T}+\mathsf{S}}(E) \,:\, \mathsf{T} + \mathsf{S}}\ (RInl) \qquad \frac{\Gamma \vdash E \,:\, \mathsf{S}}{\Gamma \vdash \mathsf{inr}^{\mathsf{T}+\mathsf{S}}(E) \,:\, \mathsf{T} + \mathsf{S}}\ (RInr)$$

$$\frac{\Gamma \vdash E \,:\, \mathsf{T}_1 + \mathsf{T}_2 \quad \Gamma, x : \mathsf{T}_1 \vdash E_1 \,:\, \mathsf{T} \quad \Gamma, y : \mathsf{T}_2 \vdash E_2 \,:\, \mathsf{T}}{\Gamma \vdash \mathsf{case}\ E\ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\} : \mathsf{T}}\ (RCase)$$

$$\frac{\Gamma \vdash E : \mathsf{void}}{\Gamma \vdash \mathsf{Abort}_{\mathsf{T}}(E) \,:\, \mathsf{T}}\ (RAbort)$$

Reglas de tipificación para $\lambda^{\to,\bot,+,\times}$

Detengámonos un instante para discutir la regla $(RAbort)$. Ésta se presenta cuando $\Gamma \vdash E : \mathsf{void}$, es decir, en el caso en el que la expresión E genere un error ningún tipo se le puede asignar, no hay candidato para el tipo de E, así que le asociamos el tipo void. Entonces, debido a que hubo un error en E, la ejecución del programa termina regresando un tipo T cualquiera.

Utilizando las reglas anteriores comprobemos que el tipo asignado a la expresión,

$$\vdash \mathsf{case}\ \mathsf{inr}^{(\mathsf{Nat}\to\mathsf{Nat})+\mathsf{Nat}}(7)\ \{\mathsf{inl}(x).\,13 \,|\, \mathsf{inr}(y).\,\mathsf{suma}\,(9, y)\} : \mathsf{Nat}$$

sea correcto. Comencemos con estas tres derivaciones sencillas,

$$\frac{\dfrac{}{\vdash 7 : \mathsf{Nat}}\ (RNat)}{\vdash \mathsf{inr}^{(\mathsf{Nat}\to\mathsf{Nat})+\mathsf{Nat}}(7) : (\mathsf{Nat} \to \mathsf{Nat}) + \mathsf{Nat}}\ (RInr)$$

$$\frac{}{x : \mathsf{Nat} \to \mathsf{Nat} \vdash 13 : \mathsf{Nat}}\ (RNat)$$

$$\frac{\dfrac{}{y : \mathsf{Nat} \vdash 9 : \mathsf{Nat}}\ (RNat) \qquad \dfrac{}{y : \mathsf{Nat} \vdash y : \mathsf{Nat}}\ (RVar)}{y : \mathsf{Nat} \vdash \mathsf{suma}\,(9, y) : \mathsf{Nat}}\ (RSuma)$$

Por las derivaciones de arriba y por la regla $(RCase)$ concluimos que,

$$\vdash \mathsf{case}\ \mathsf{inr}^{(\mathsf{Nat}\to\mathsf{Nat})+\mathsf{Nat}}(7)\ \{\mathsf{inl}(x).\,13 \mid \mathsf{inr}(y).\,\mathsf{suma}\,(9,y)\} : \mathsf{Nat}$$

Una vez establecido el mecanismo de asignación de tipos podemos dar la semántica para $\lambda^{\to,\bot,+,\times}$. Los valores que las expresiones pueden producir son,

$$val := \mathsf{inl}^{\mathsf{T}+\mathsf{S}}(val) \mid \mathsf{inr}^{\mathsf{T}+\mathsf{S}}(val) \mid \lambda x : \mathsf{T}.E \mid * \mid \langle val, val\rangle \mid 0 \mid 1 \mid \ldots$$

Para describir la ejecución de los programas en $\lambda^{\to,\bot,+,\times}$ requerimos de una máquina abstracta que tome un estado[7] y devuelva el siguiente obedeciendo una relación de transición $(\to)$. Las reglas de evaluación que definen el sistema de transición están expuestas abajo. Se adoptará por una estrategia de evaluación de *llamada-por-valor* [8].

$$\frac{E_0 \to E_0'}{\mathsf{suma}(E_0,E_1) \to \mathsf{suma}(E_0',E_1)} \qquad \frac{E_1 \to E_1'}{\mathsf{suma}(n_0,E_1) \to \mathsf{suma}(n_0,E_1')}$$

$$\frac{}{\mathsf{suma}(n_0,n_1) \to n_0+n_1}$$

$$\frac{E_0 \to E_0'}{E_0E_1 \to E_0'E_1} \qquad \frac{E_1 \to E_1'}{(\lambda x.E)E_1 \to (\lambda x.E)E_1'}$$

$$\frac{}{(\lambda x.E)val \to E[x := val]}$$

$$\frac{E \to E'}{\mathsf{inl}^{\mathsf{T}+\mathsf{S}}(E) \to \mathsf{inl}^{\mathsf{T}+\mathsf{S}}(E')} \qquad \frac{E \to E'}{\mathsf{inr}^{\mathsf{T}+\mathsf{S}}(E) \to \mathsf{inr}^{\mathsf{T}+\mathsf{S}}(E')}$$

$$\frac{E \to E'}{\mathsf{case}\ E\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\} \to \mathsf{case}\ E'\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\}}$$

$$\frac{}{\mathsf{case}\ \mathsf{inl}^{\mathsf{T}+\mathsf{S}}(val)\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\} \to E_1[x := val]}$$

$$\frac{}{\mathsf{case}\ \mathsf{inr}^{\mathsf{T}+\mathsf{S}}(val)\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\} \to E_2[y := val]}$$

$$\frac{E_1 \to E_1'}{\langle E_1,E_2\rangle \to \langle E_1',E_2\rangle} \qquad \frac{E_2 \to E_2'}{\langle val,E_2\rangle \to \langle val,E_2'\rangle}$$

[7] Un estado de la máquina abstracta es un término de $\lambda^{\to,\bot,+,\times}$.

[8] En una estrategia de *llamada-por-valor* es necesario obtener, en primer lugar, los valores de los argumentos de la expresión para después, considerando estos valores, evaluar la expresión misma.

$$\frac{E \to E'}{\mathsf{fst}\, E \to \mathsf{fst}\, E'} \qquad \frac{}{\mathsf{fst}\, \langle val_1, val_2 \rangle \to val_1}$$

$$\frac{E \to E'}{\mathsf{snd}\, E \to \mathsf{snd}\, E'} \qquad \frac{}{\mathsf{snd}\, \langle val_1, val_2 \rangle \to val_2}$$

Reglas de evaluación para $\lambda^{\to,\bot,+,\times}$

Como ejemplo, observemos la evaluación siguiente,

$$\begin{aligned}
&\mathsf{case}\ \mathsf{inr}^{(\mathsf{Nat}\to\mathsf{Nat})+\mathsf{Nat}}(7)\ \{\mathsf{inl}(x).\, 13 \mid \mathsf{inr}(y).\, \mathsf{suma}\,(9, y)\} \\
&\to (\mathsf{suma}\,(9, y))[y := 7] = \mathsf{suma}\,(9, 7) \\
&\to 16
\end{aligned}$$

Las propiedades de *reducción del sujeto*, *normalización fuerte* y *confluencia* siguen siendo válidas en esta extensión.

1.3. Isomorfismo de Curry-Howard

El *isomorfismo de Curry-Howard* da a conocer que el comportamiento que se presenta en un sistema lógico de deducción natural es el mismo que se encuentra en un sistema de tipos de un lenguaje de programación, en particular, las fórmulas lógicas pueden ser consideradas como tipos, por esta razón también se le conoce como *isomorfismo de fórmulas-como-tipos*, o *proposiciones-como-tipos*. En esta sección analizaremos esta relación para la lógica intuicionista, $NJ(\to, \bot, \vee, \wedge)$, y el cálculo lambda $\lambda^{\to,\bot,+,\times}$. Para apreciar de mejor manera la correspondencia entre los elementos del cálculo λ y los elementos de la lógica intuicionista NJ se muestra la siguiente tabla,

λ	**NJ**
Variable como término del lenguaje	Hipótesis
Término	Prueba o construcción
Variable de tipo	Variable proposicional
Tipo	Fórmula
Constructor de tipo	Conectivo
Asignación de tipo	Demostrabilidad
Redex	Demostración redundante
Normalización de β-reducción	Normalización de pruebas

Los tres resultados que estudiaremos son,

1. En el *isomorfismo de Curry-Howard* las derivaciones de fórmulas lógicas corresponden a derivaciones de tipos.

2. Las pruebas de la lógica intuicionista $NJ(\to, \bot, \vee, \wedge)$ se relacionan con los términos $\lambda^{\to,\bot,+,\times}$.

3. La normalización de pruebas corresponde a la reducción de términos.

1.3.1. Isomorfismo de Curry-Howard para $NJ(\to, \bot, \vee, \wedge)$ y $\lambda^{\to,\bot,+,\times}$

El sistema de tipos en el cual nos centraremos es el definido anteriormente para $\lambda^{\to,\bot,+,\times}$ omitiendo todo lo relacionado al tipo Nat y a la operación $\mathsf{suma}(x, y)$. Ocuparemos el cálculo proposicional intuicionista $NJ(\to, \bot)$ aumentado con las reglas de introducción y eliminación del $\vee$ y del $\wedge$. Esta modificación produce el cálculo $NJ(\to, \bot, \vee, \wedge)$ cuyas reglas de inferencia son las siguientes,

$$\frac{\Gamma \vdash_i \varphi \quad \Gamma \vdash_i \psi}{\Gamma \vdash_i \varphi \wedge \psi}\,(\wedge I) \qquad \frac{\Gamma \vdash_i \varphi \wedge \psi}{\Gamma \vdash_i \varphi}\,(\wedge_1 E) \qquad \frac{\Gamma \vdash_i \varphi \wedge \psi}{\Gamma \vdash_i \psi}\,(\wedge_2 E)$$

$$\frac{\Gamma \vdash_i \varphi}{\Gamma \vdash_i \varphi \vee \psi}\,(\vee_1 I) \quad \frac{\Gamma \vdash_i \psi}{\Gamma \vdash_i \varphi \vee \psi}\,(\vee_2 I)$$

$$\frac{\Gamma \vdash_i \varphi \vee \psi \quad \Gamma, \varphi \vdash_i \chi \quad \Gamma, \psi \vdash_i \chi}{\Gamma \vdash_i \chi}\,(\vee E)$$

$$\frac{}{\Gamma, \varphi \vdash_i \varphi}\,(Ax) \qquad \frac{\Gamma, \varphi \vdash_i \psi}{\Gamma \vdash_i \varphi \to \psi}\,(\to I)$$

$$\frac{\Gamma \vdash_i \varphi \to \psi \quad \Gamma \vdash_i \varphi}{\Gamma \vdash_i \psi}\,(\to E)$$

$$\frac{\Gamma \vdash_i \bot}{\Gamma \vdash_i \varphi}\,(EFQ)$$

Reglas del sistema $NJ(\to, \bot, \vee, \wedge)$.

Necesitamos primero saber cómo pasar de fórmulas a tipos y viceversa. Entonces, si φ es una fórmula, el tipo φ^* que se le asocia se define recursivamente como:

$$\begin{aligned}\bot^* &= \mathsf{void}\\ (\varphi \wedge \psi)^* &= \varphi^* \times \psi^*\\ (\varphi \vee \psi)^* &= \varphi^* + \psi^*\\ (\varphi \to \psi)^* &= \varphi^* \to \psi^*\end{aligned}$$

La transformación anterior $(-)^*$ es biyectiva, por lo que tiene inversa $(-)^\circ$ que transforma tipos en fórmulas.

Esta definición se amplia para que abarque contextos lógicos y contextos de tipos. Si $\Gamma = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ es un contexto lógico, definimos su contexto de tipos asociado como $\Gamma^* = \{x_1 : \varphi_1^*,\ x_2 : \varphi_2^*, \ldots,\ x_n : \varphi_n^*\}$. Si se tiene $\Delta = \{y_1 : \mathsf{T}_1,\ y_2 : \mathsf{T}_2, \ldots,\ y_m : \mathsf{T}_m\}$ contexto de tipos, su contexto lógico correspondiente es $\Delta^\circ = \{\mathsf{T}_1^\circ,\ \mathsf{T}_2^\circ, \ldots,\ \mathsf{T}_m^\circ\}$.

Teorema 1.1. (Isomorfismo de Curry-Howard) *Sean φ una fórmula en $NJ(\to, \bot, \vee, \wedge)$, T un tipo de $\lambda^{\to,\bot,+,\times}$, Γ un contexto lógico y Δ un contexto de tipos.*

I. *Si $\Delta \vdash E : \mathsf{T}$ en $\lambda^{\to,\bot,+,\times}$ entonces $\Delta^\circ \vdash_i \mathsf{T}^\circ$ en $NJ(\to, \bot, \vee, \wedge)$.*

II. *Si $\Gamma \vdash_i \varphi$ en $NJ(\to, \bot, \vee, \wedge)$ entonces existe un término E tal que $\Gamma^* \vdash E : \varphi^*$ en $\lambda^{\to,\bot,+,\times}$.*

Demostración. Para I procedemos por inducción sobre la derivación de tipos $\Delta \vdash E : \mathsf{T}$.

- *Base de inducción.*

$$\frac{}{\Delta, x : \mathsf{T} \vdash x : \mathsf{T}}\ (RVar)$$

  Aplicando la transformación $(-)^\circ$ llegamos a que $(\Delta, x : \mathsf{T})^\circ = \Delta^\circ, \mathsf{T}^\circ$ y ocupando (Ax) de las reglas para $NJ(\to, \bot, \vee, \wedge)$ llegamos a $\Delta^\circ, \mathsf{T}^\circ \vdash_i \mathsf{T}^\circ$.

- La prueba para cada uno de los casos del paso inductivo se encuentra en [24]. Seleccionamos como muestra cuando la derivación de tipos

termina empleando la regla ($RCase$). Suponemos que la proposición es válida para $\Delta \vdash E : \mathsf{T_1} + \mathsf{T_2},\ \Delta, x : \mathsf{T_1} \vdash E_1 : \mathsf{T}$ y $\Delta, y : \mathsf{T_2} \vdash E_2 : \mathsf{T}$. Por hipotésis de inducción sus inferencias lógicas respectivas son $\Delta^\circ \vdash_i \mathsf{T_1}^\circ \vee \mathsf{T_2}^\circ,\ \Delta^\circ, \mathsf{T_1}^\circ \vdash_i \mathsf{T}^\circ$ y $\Delta^\circ, \mathsf{T_2}^\circ \vdash_i \mathsf{T}^\circ$. Usando este hecho y la regla $(\vee E)$ concluimos que para $\Delta \vdash \mathsf{case}\ E\ \{\mathsf{inl}(x).E_1 \mid \mathsf{inr}(y).E_2\} : \mathsf{T}$ la inferencia $\Delta^\circ \vdash_i \mathsf{T}^\circ$ es válida en $NJ(\to, \bot, \vee, \wedge)$.

La demostración para II es por inducción sobre la inferencia $\Gamma \vdash_i \varphi$.

- *Base de inducción.* La inferencia es,

$$\frac{}{\Gamma, \varphi \vdash_i \varphi}\,(Ax)$$

  Al transformar Γ, φ obtenemos $\Gamma^*, x_n : \varphi^*$. Usando la regla de tipificación $(RVar)$ llegamos a $\Gamma^*, x_n : \varphi^* \vdash x_n : \varphi^*$, con $E = x_n$.

- Ahora debemos comprobar la validez del teorema para cada uno de los casos del paso inductivo, esto se muestra en [24]. Como ejemplo tomemos cuando la inferencia termina usando la regla de eliminación de la implicación $(\to\ E)$. Por HI para $\Gamma \vdash_i \varphi \to \psi$ y $\Gamma \vdash_i \varphi$ existen términos E_1 y E_2 tales que $\Gamma^* \vdash E_1 : \varphi^* \to \psi^*$ y $\Gamma^* \vdash E_2 : \varphi^*$. De modo que para $\Gamma \vdash_i \psi$ tomemos $E = E_1 E_2$ cumpliendo $\Gamma^* \vdash E : \psi^*$, esto es por la regla $(RApp)$.

□

1.3.2. Pruebas en Lógica Intuicionista

Como estudiamos al principio de este capítulo la lógica intuicionista se basa en la idea de que la verdad significa la construcción de una prueba. Los juicios acerca de una fórmula se basan en la habilidad para construirla mediante una prueba explícita, el significado de las fórmulas compuestas se explica en términos de sus construcciones. Tal explicación se da de manera intuitiva mediante la llamada interpretación de Brouwer-Heyting-Kolmogorov (BHK) que es reconocida ampliamente como la semántica intensional de la lógica intuicionista, se define como:

- Una prueba de una variable proposicional p se supone conocida y dada por un contexto previamente definido.

- Una prueba de $\varphi \wedge \psi$ requiere de una prueba para φ y de una prueba para ψ.
- Una prueba de $\varphi \vee \psi$ consiste en una prueba de φ o una prueba de ψ indicando cual de las fórmulas es la que en efecto se ha probado.
- Una prueba de $\varphi \to \psi$ es una función que toma una prueba de φ y devuelve una prueba de ψ.
- El valor $\bot$ no tiene prueba alguna. No obstante, una prueba para $\neg\varphi$ es una construcción en la que a una prueba de φ se le asignaría (si existiese) una prueba de $\bot$, esto es $\varphi \to \bot$.

Esta semántica busca una manera de representar puntualmente las fórmulas correspondientes a la lógica intuicionista. Resulta conveniente utilizar al cálculo $\lambda^{\to,\bot,+,\times}$ como notación para las pruebas que busca la semántica BHK. Por el *isomorfismo de Curry-Howard* a cada prueba de una fórmula(tipo) φ, se le asociará un código único mediante un término E del cálculo $\lambda^{\to,\bot,+,\times}$, esta asociación se denota como $E : \varphi$. Ahora los contextos lógicos son de la forma $\Gamma = \{x_1 : \varphi_1, \ldots, x_n : \varphi_n\}$, donde x_i es una prueba para φ_i. Notemos que en este nuevo sistema de deducción figuran tanto los elementos de la lógica intuicionista como los términos del cálculo $\lambda^{\to,\bot,+,\times}$. Entonces $\Gamma \vdash E : \varphi$ tiene un significado dual. Por un lado, es la inferencia en la lógica intuicionista de φ con código de prueba E y contexto lógico Γ. Por otro lado, es la tipificación del término E del $\lambda^{\to,\bot,+,\times}$ con tipo φ y contexto de tipos Γ.

1.3.3. Reglas de reducción

Analizando la interpretación BHK y su representación en $\lambda^{\to,\bot,+,\times}$ nos damos cuenta de que es posible simplificar o reducir las pruebas, esto se debe a que las pruebas iniciales y las pruebas finales son redundantes. Veamos lo que sucede en la derivación siguiente donde a la fórmula χ se le asigna la prueba $\mathsf{case}\ \mathsf{inl}^{\varphi+\psi}(E)\ \ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\}$,

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash E : \varphi}}{\Gamma \vdash \mathsf{inl}^{\varphi\vee\psi}(E) : \varphi \vee \psi} \quad \dfrac{\vdots}{\Gamma, x : \varphi \vdash E_1 : \chi} \quad \dfrac{\vdots}{\Gamma, y : \psi \vdash E_2 : \chi}}{\Gamma \vdash \mathsf{case}\ \mathsf{inl}^{\varphi+\psi}(E)\ \ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\} : \chi}$$

Sabemos que $\Gamma, x : \varphi \vdash E_1 : \chi$ y también $\Gamma \vdash E : \varphi$, por lo que la proposición 1.5 implica que la sustitución $\Gamma \vdash E_1[x := E] : \chi$ es válida. Entonces podemos realizar la reducción,

$$\mathsf{case}\ \mathsf{inl}^{\varphi+\psi}(E)\ \ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\} \to E_1[x := E]$$

El término $\mathsf{case}\ \mathsf{inl}^{\varphi+\psi}(E)\ \ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\}$ es una prueba de χ del mismo modo que $E_1[x := E]$, aunque esta última es más sencilla; por lo tanto, la reducción que acabamos de dar es un proceso en el que las pruebas se simplifican.

Similarmente, si $\Gamma \vdash \mathsf{fst}\langle E_1, E_2\rangle : \varphi$ se debe cumplir la derivación siguiente,

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash E_1 : \varphi} \quad \dfrac{\vdots}{\Gamma \vdash E_2 : \varphi}}{\Gamma \vdash \langle E_1, E_2\rangle : \varphi \wedge \psi}}{\Gamma \vdash \mathsf{fst}\langle E_1, E_2\rangle : \varphi}$$

Vemos que si tenemos la prueba $\mathsf{fst}\langle E_1, E_2\rangle$ necesariamente se tiene la prueba E_1, además ambas son pruebas de la misma fórmula φ. Luego la reducción es,

$$\mathsf{fst}\langle E_1, E_2\rangle \to E_1$$

Siguiendo esta idea las demás reglas de reducción son,

$$\mathsf{case}\ \mathsf{inr}^{\varphi+\psi}(E)\ \ \{\mathsf{inl}(x).E_1 \,|\, \mathsf{inr}(y).E_2\} \to E_2[y := E]$$

$$\mathsf{snd}\langle E_1, E_2\rangle \to E_2$$

$$(\lambda x.E_1)E_2 \to E_1[x := E_2]$$

Estas simplificaciones forman parte de las reglas de evaluación para $\lambda^{\to,\bot,+,\times}$ utilizando una estrategia de *llamada-por-nombre*[9].

Se introdujeron en este capítulo la deducción natural y el cálculo lambda de Church. Además, se estableció el *isomorfismo de Curry-Howard* entre la lógica intuicionista y $\lambda^{\to,\bot,+,\times}$. En el siguiente capítulo se utilizarán todos estos conceptos para definir el cálculo $\lambda\mathcal{C}$, que proveerá de contenido computacional a la lógica clásica.

[9]En una estrategia de *llamada-por-nombre* la expresión se reduce directamente sin que antes se requieran obtener los valores de sus argumentos.

Capítulo 2

Cálculo $\lambda\mathcal{C}$

Una característica importante y poderosa en lenguajes de programación es la capacidad de alterar la secuencia de ejecución de un programa de acuerdo a determinadas condiciones. Actualmente se cuenta con una amplia gama de instrucciones que realizan esta tarea, como son if...then...else, return..., switch...case..., foreach..., entre otras. Estas instrucciones modifican de manera segura el orden de ejecución del programa, a diferencia de la instrucción goto que puede generar un comportamiento inestable y confuso, cuyo uso es poco recomendable y ha desaparecido por completo de ciertos lenguajes de programación como Java.

Un escenario común para modificar el orden de evaluación de un programa es cuando se presentan situaciones anormales como división entre cero, referencia a un apuntador nulo o lectura de un archivo inexistente. Para responder a esta anomalía necesitamos de una instrucción que reconozca cuando se ha presentado una excepción y salte a un bloque de código que maneje el error adecuadamente. Instrucciones que actúan de esa manera son try...catch...finally, abort, catch...throw... y handle err...with y se les llama operadores de control porque devuelven una expresión que salta a los demás términos del programa y se convierte en el valor final.

Una manera más general de operar sobre el contexto de ejecución es a través de *continuaciones*, las cuales son una representación del resto del cómputo a realizar después de que determinada tarea se efectúe. Por tanto, una continuación es una función que abstrae la evaluación del subtérmino actual presente en una expresión, una vez que se obtiene su valor los subtérminos restantes son evaluados; finalmente, los resultados se combinan para obtener el valor de la expresión original.

Como ejemplo considérese la siguiente expresión,

$$\mathsf{fst}\langle 5 + 3 * 4, 16/2\rangle$$

Tomando en cuenta que la multiplicación tiene mayor precedencia que la suma, entonces la continuación para la expresión $3*4$ es la función $\lambda k.\mathsf{fst}\langle 5 + k, 16/2\rangle$, es decir, una vez que se ha evaluado $k = 3 * 4$ el cálculo pendiente es $\mathsf{fst}\langle 5 + k, 16/2\rangle$.

Esta función continuación puede usarse de diversas maneras, incluso puede formar parte de los argumentos de otra función o ser el valor que regresa un método, lo que da lugar a conceptos como funciones de orden superior y *callbacks*[10].

Los operadores $\mathcal{C}$ de Felleisen y $\mathsf{call/cc}$[11] del lenguaje de programación SCHEME son ejemplos de funciones o abstracciones que manejan continuaciones, además, por su poder expresivo, se pueden tomar como base para definir otros operadores de control.

Analizaremos a detalle el cálculo $\lambda\mathcal{C}$ de Felleisen prestando gran interés en el operador $\mathcal{C}$, obteniendo su tipo y considerando sus propiedades. Asimismo lo utilizaremos para desarrollar algunos manejadores de excepciones, también servirá para expresar operaciones referentes a productos y sumas.

2.1. Cálculo $\lambda\mathcal{C}$ de Felleisen

Al estudiar el cálculo λ puro en el capítulo anterior, pudimos apreciar su funcionamiento, el cual está orientado exclusivamente a funciones y su aplicación, siendo éste su único mecanismo de evaluación. Esto satisface el objetivo que pretendía Alonzo Church; sin embargo, este lenguaje ofrece poca expresividad particularmente en lo referente al manejo del contexto de evaluaciones. Esta misma simplicidad hace al cálculo λ ineficiente. Supongamos que se ha desarrollado un programa recursivo usando el cálculo λ, nos gustaría poder finalizar el programa una vez que se ha encontrado una condición de escape

[10] Un *callback* es una función f pasada a otra función g tal que f es llamada en algún momento dentro de la función g.

[11] El significado de $\mathsf{call/cc}$ es *call with current continuation*, y recibe como argumento una función h a la que se le pasa la continuación actual, en el momento en que esta continuación es invocada en h se abandonará el contexto presente y se regresará al contexto donde el operador $\mathsf{call/cc}$ fue definido.

o término; lamentablemente, se deberá ejecutar toda la secuencia de instrucciones faltantes sin importar que ya no tiene caso seguir con la evaluación. De igual manera, en una situación de error lo que se desearía es suspender cualquier evaluación actual y procesar un código de recuperación que maneje la excepción. Resulta imprescindible el uso de continuaciones para solucionar estas deficiencias.

Podríamos adoptar un estilo de programación, por ejemplo CPS[12], en el que las continuaciones son tomadas como abstracciones lambda. De modo que los programas puedan accesar a la continuación actual al llamar a la función que la representa, obteniendo flexibilidad y control en el proceso de evaluación, decidiendo ejecutar o no la continuación actual, guardarla para un uso posterior o invocar una continuación previa. Aunque correcta, esta manera de usar continuaciones es difícil y engorrosa; por consiguiente, se tratarán las continuaciones a través de operadores de control. Los operadores call/cc y $\mathcal{C}$ actúan sobre las continuaciones obedeciendo reglas bien definidas, abstrayendo los detalles que estarían presentes al considerar la interpretación directa con funciones.

El cálculo $\lambda\mathcal{C}$ fue creado en los años ochenta por Matthias Felleisen [6], toma como base el cálculo λ del capítulo anterior aumentado con los operadores $\mathcal{C}$ (*Control*) y $\mathcal{A}$ (*Abort*), además de sus correspondientes aplicaciones. La sintáxis de los términos del cálculo $\lambda\mathcal{C}$ queda definida con la gramática,

$$M ::= x \mid \lambda x.M \mid MM \mid \mathcal{C}M \mid \mathcal{A}M$$

Identificamos al conjunto de valores V como la unión de variables y abstracciones lambda. A la definición de substitución dada previamente en el capítulo 1 se le añaden los casos para las aplicaciones $\mathcal{C}$ y $\mathcal{A}$.

- $(\mathcal{C}M)[x := N] = \mathcal{C}(M[x := N])$
- $(\mathcal{A}M)[x := N] = \mathcal{A}(M[x := N])$

La evaluación de los términos será de *llamada-por-valor*. Asimismo, se requiere de un nuevo concepto: los contextos de evaluación o contextos aplicativos. Éstos se definen recursivamente como,

$$E ::= [\,] \mid VE \mid EM$$

[12]Continuation Passing Style.

donde [ ] representa un *hueco*, de tal manera que si E es un contexto de evaluación y M un término entonces $E[M]$ denota el término obtenido al llenar el hueco de E con M. Formalmente $E[M]$ se define como,

$$\begin{array}{lcl} [\,][M] & = & M \\ (EN)[M] & = & (E[M])N \\ (VE)[M] & = & V(E[M]) \end{array}$$

Definición 2.1. *Un $\mathcal{C}$-redex es un β_v-redex [13], una aplicación $\mathcal{C}$ o una aplicación $\mathcal{A}$.*

Proposición 2.1. *Cualquier término cerrado M es un valor o puede ser expresado de manera única como $E[T]$ donde T es un $\mathcal{C}$-redex.*

Demostración. En [10] Griffin solo menciona este resultado sin dar demostración, nosotros procedemos por inducción sobre M.

- *Base de inducción.*

  Si M es una variable entonces no es un término cerrado, el antecedente es falso y, por consiguiente, la proposición es cierta.

- *Hipótesis de inducción.*

  Sea M un término cerrado, entonces podemos expresar a M como $E[T]$, con E contexto de evaluación y T un $\mathcal{C}$-redex.

- *Paso inductivo.* Por demostrar que la proposición es cierta para M'.

  Si M' es la abstracción lambda $\lambda x.M$ entonces se cumple la proposición debido a que M' es valor.

  Si tenemos $M' = \mathcal{C}M$ o $M' = \mathcal{A}M$ entonces M' es un $\mathcal{C}$-redex, así que tomamos el contexto vacio $E = [\,]$ y $T = M'$, con lo cual $M' = E[T]$.

  Si M' es una aplicación tenemos los siguientes casos,

  Caso 1. La aplicación es $M' = (\lambda x.M)V$, esto es un $\mathcal{C}$-redex. Tomando $E = [\,]$ y $T = (\lambda x.M)V$ se satisface que $M' = E[T]$

[13] En la estrategia de *llamada-por-valor*, un β_v-redex es una expresión de la forma $(\lambda x.M)V$, con V un valor.

Caso 2. $M' = M_0 M$ con M_0 valor. Como M' es cerrado M_0 es la abstracción lambda $\lambda x.N$ y M es una expresión cerrada. La hipótesis de inducción nos dice que la proposición es válida para M, es decir, existe un contexto de evaluación E_1 y un $\mathcal{C}$-redex T_1 tal que $M = E_1[T_1]$. Entonces, tomando el contexto $E = (\lambda x.N)E_1$ se tiene que,

$$\begin{aligned} E[T_1] &= ((\lambda x.N)E_1)[T_1] \\ &= (\lambda x.N)(E_1[T_1]) \\ &= (\lambda x.N)M = M' \end{aligned}$$

Caso 3. $M' = MM_1$ y M no es un valor. M debe ser una expresión cerrada así que la hipótesis de inducción dice que existen E_0 y T_0 tales que $M = E_0[T_0]$. Si consideramos $E = E_0 M_1$ entonces,

$$\begin{aligned} E[T_0] &= (E_0 M_1)[T_0] \\ &= (E_0[T_0])M_1 \\ &= MM_1 = M \end{aligned}$$

En los pasos de la inducción se nos da la técnica para encontrar E y T dado M. Esta construcción, con su caracter determinista, nos proporciona la unicidad del contexto E y de la representación de M como $E[T]$. □

Como ejemplo tomemos el término cerrado,

$$M = \Big((\lambda x.x)((\lambda yz.yyz)\mathcal{C}(\lambda v.v))\Big)(\lambda w.w)$$

así que de acuerdo a la construcción propuesta en la demostración tenemos,

$$\begin{aligned} \Big((\lambda x.x)((\lambda yz.yyz)\mathcal{C}(\lambda v.v))\Big)(\lambda w.w) \qquad & E = E_0(\lambda w.w) \\ (\lambda x.x)((\lambda yz.yyz)\mathcal{C}(\lambda v.v)) \qquad & E_0 = (\lambda x.x)E_1 \\ (\lambda yz.yyz)\mathcal{C}(\lambda v.v) \qquad & E_1 = (\lambda yz.yyz)E_2 \\ \mathcal{C}(\lambda v.v) \qquad & E_2 = [\,] \\ & T = \mathcal{C}(\lambda v.v) \end{aligned}$$

Por consiguiente, $E = \Big((\lambda x.x)((\lambda yz.yyz)[\,])\Big)(\lambda w.w)$ y $M = E[\mathcal{C}(\lambda v.v)]$

Por el procedimiento que se exhibe en la demostración afirmamos también que en $E[T]$ la T se refiere al $\mathcal{C}$-redex más a la izquierda de M fuera del alcance de una abstracción lambda. Esta situación se debe a que en la demostración, en el caso 2 para la aplicación, el contexto es $(\lambda x.N)E_1$, al estar el hueco en E_1 se encuentra fuera del alcance de la abstracción lambda. Mientras que en el caso 3 para la aplicación, el contexto es E_0M_1, de modo que el hueco está a la izquierda, en E_0, y todo $\mathcal{C}$-redex que se encuentre en M_1 será ignorado. Por lo tanto, el término final T estará fuera del alcance de una función y se localizará lo más a la izquierda en M.

Con los contextos de evaluación podemos dar la semántica operacional a través del *sistema de reescritura* $\mathcal{C}$ ($\mapsto_{\mathcal{C}}$) cuyas reglas se muestran enseguida,

$$E[(\lambda x.M)V] \mapsto_{\mathcal{C}} E[M[x := V]] \qquad (\mathcal{C}1)$$
$$E[\mathcal{C}M] \mapsto_{\mathcal{C}} M(\lambda z.\mathcal{A}(E[z])) \qquad (\mathcal{C}2)$$
$$E[\mathcal{A}M] \mapsto_{\mathcal{C}} M \qquad (\mathcal{C}3)$$

Sistema de reescritura $\mathcal{C}$

La regla de reescritura $(\mathcal{C}1)$ también se denota con $\mapsto_{\beta_v}$.

Tomemos la situación cuando $E = [\,]$, así que $E[\mathcal{A}M] = [\,][\mathcal{A}M] = \mathcal{A}M$, en este caso, usando la regla $(\mathcal{C}3)$, llegamos a,

$$[\,][\mathcal{A}M] = \mathcal{A}M \mapsto_{\mathcal{C}} M$$

Demos ahora el significado de las nuevas aplicaciones $\mathcal{C}$ y $\mathcal{A}$. El funcionamiento de la aplicación $\mathcal{A}M$ puede entenderse como *abortar*, detener o terminar la evaluación del programa, olvidando el contexto de evaluación actual y continuando con la ejecución de M. Esto es precisamente lo que se expresa con la regla $(\mathcal{C}3)$. Similarmente, el término $\mathcal{C}M$ se deshace del contexto de evaluación actual; no obstante, genera la aplicación de su argumento M con una abstracción del contexto deshechado, esta continuación es $\lambda z.\mathcal{A}(E[z])$, por lo que su regla de reducción es $(\mathcal{C}2)$. Decimos que una continuación es invocada, o lanzada, cuando se le aplica un valor. Notemos entonces que al invocar la continuación $\lambda z.\mathcal{A}(E[z])$ en un contexto de evaluación E_1 obtenemos,

$$\begin{aligned} E_1[(\lambda z.\mathcal{A}(E[z]))V] &\mapsto_{\mathcal{C}} E_1[\mathcal{A}(E[V])] \\ &\mapsto_{\mathcal{C}} E[V] \end{aligned} \qquad (Inv)$$

Esto es, el contexto actual E_1 es abandonado y se retoma la evaluación con $E[V]$.

Denotamos la cerradura reflexiva y transitiva de $\mapsto_{\mathcal{C}}$ con $\mapsto^*_{\mathcal{C}}$.

Sabemos por proposición 2.1 que cualquier programa se puede expresar de manera única en términos de un contexto de evaluación y un $\mathcal{C}$-redex, esto hace que el proceso de reducción sea determinístico puesto que siempre se evalúa el $\mathcal{C}$-redex más a la izquierda y fuera del alcance de una abstracción lambda. De lo cual se sigue que,

- Si $E[M] \mapsto^k_{\beta_v} E[N]$ entonces $M \mapsto^k_{\beta_v} N$, donde $\mapsto^k_{\beta_v}$ quiere decir que la regla $\mapsto_{\beta_v}$ se ha empleado k veces.
- Si $E[M] \mapsto^*_{\mathcal{C}} V$, entonces M se evalúa a M_0, donde M_0 es un valor, una aplicación $\mathcal{C}$ o una aplicación $\mathcal{A}$, tal que $E[M] \mapsto^*_{\mathcal{C}} E[M_0] \mapsto^*_{\mathcal{C}} V$.

Estos resultados nos indican la manera en la que se evalúan los términos, en el sentido de que si $M = E[N]$, la E se encuentra en un estado de espera hasta que el control pase a otro contexto, lo que ocurre si se ejecuta una aplicación $\mathcal{C}$ o $\mathcal{A}$, o aguarda hasta que la N se reduzca a un valor para poder continuar con la evaluación de subtérminos propios de E. Así, la E representa el *resto del cómputo faltante una vez que N se ha evaluado*, es decir, E es la *continuación de N*.

2.1.1. Sistema de reducciones

Si bien las reglas del *sistema de reescritura* $\mathcal{C}$ sirven para definir la semántica operacional del lenguaje, éstas no son propicias para razonar acerca de programas porque en realidad el *sistema de reescritura* $\mathcal{C}$ se define a través de una máquina abstracta. Más aún, desde un punto de vista computacional, el nuevo sistema de reducciones del que trata esta sección es más atractivo ya que muestra las reglas del cálculo y su naturaleza operacional. Lo que pretendemos ahora es encontrar un sistema en el cual la semántica operacional sea declarada en términos de reducciones compatibles con las reglas de formación de términos, para hallar estas reducciones estudiaremos la especificación y las razones que dieron lugar a las reglas del *sistema de reescritura* $\mathcal{C}$.

En dicho sistema de reescritura una transición depende de particionar un programa en un $\mathcal{C}$-redex y en un contexto de evaluación. Para establecer el nuevo sistema será primordial deshacernos de la dependencia existente hacia

los contextos de evaluación tanto como sea posible. Esto será sencillo para la regla ($\mathcal{C}1$) pero no será así para las demás reglas porque en ellas se tiene una dependencia relevante hacia los contextos de evaluación.

En ($\mathcal{C}1$) el contexto no interviene directamente en la reducción, no hay dependencia alguna y puede ser ignorado totalmente. La nueva regla es:

$$(\lambda x.M)V \xrightarrow{\beta_v} M[x := V], \text{ siendo } V \text{ un valor} \qquad (\beta_v)$$

Tomemos ahora la aplicación $\mathcal{A}$ y la regla ($\mathcal{C}3$). Al realizar un análisis por casos de los contextos de evaluación conseguiremos las reducciones deseadas.

- El primer caso es cuando la aplicación $\mathcal{A}M$ está a la izquierda del término N, es decir, se tiene $(\mathcal{A}M)N$, y esto se encuentra dentro del contexto E. Entonces podemos considerar a N como parte del contexto de $\mathcal{A}M$. De acuerdo a la regla ($\mathcal{C}3$) dicho contexto es olvidado, de manera que la N será expulsada y entonces el contexto E será eliminado. En otras palabras, $E[(\mathcal{A}M)N]$ se comporta como $E[\mathcal{A}M]$. Esto es cierto independientemente del contexto, por lo que formulamos la primera reducción para la aplicación $\mathcal{A}$ como:

$$(\mathcal{A}M)N \xrightarrow{\mathcal{A}_L} \mathcal{A}M \qquad (\mathcal{A}_L)$$

- El segundo caso es cuando $\mathcal{A}N$ está a la derecha del valor M y dentro del contexto E, es decir, la expresión a estudiar es $E[M(\mathcal{A}N)]$. Entonces ocurre un comportamiento análogo al caso anterior ya que M pertenece al contexto de evaluación para el $\mathcal{C}$-redex $\mathcal{A}N$; asi, en lugar de tomar $E[M(\mathcal{A}N)]$ consideramos solamente $E[\mathcal{A}N]$. La reducción que se le asocia es:

$$M(\mathcal{A}N) \xrightarrow{\mathcal{A}_R} \mathcal{A}N \qquad (\mathcal{A}_R)$$

- Falta el caso base para contextos de evaluación, cuando es vacío. Lo hemos dejado al último porque requiere de especial cuidado. Vimos que cuando el contexto es vacío ocurre que $\mathcal{A}M \mapsto_{\mathcal{C}} M$, pero esto no es una reducción propia. Esta regla se aplica únicamente cuando la aplicación $\mathcal{A}M$ no está en contacto con más términos, de otro modo el sistema de transición se vuelve inconsistente, como lo vemos con la expresión $(\mathcal{A}F)TT$, donde $T = \lambda xy.x$ y $F = \lambda xy.y$. Si se utiliza ($\mathcal{A}_L$) dos veces se obtiene,

$$((\mathcal{A}F)T)T \xrightarrow{\mathcal{A}_L} (\mathcal{A}F)T \xrightarrow{\mathcal{A}_L} \mathcal{A}F$$

usando $(\mathcal{C}3)$ concluimos que $\mathcal{A}F \longrightarrow F$. Por otro lado, si aplicamos $(\mathcal{C}3)$ al inicio de $(\mathcal{A}F)TT$ la reducción nos da FTT, luego

$$FTT = (\lambda xy.y)TT \xrightarrow{\beta_v}{}^{+} T^{14}$$

Llegamos a que $(\mathcal{A}F)TT \longrightarrow^{+} T$ y $(\mathcal{A}F)TT \longrightarrow^{+} F$ pero claramente T y F no son iguales, con esta inconsistencia perdemos la propiedad de confluencia, así que $(\mathcal{C}3)$ no puede ser una reducción propia. Por lo tanto, introducimos esta relación de *alto nivel* como una *regla de cómputo* denotada con $\triangleright$:

$$\mathcal{A}M \triangleright_{\mathcal{A}} M \qquad (\mathcal{A}_T)$$

Para encontrar las reducciones correspondientes a $\mathcal{C}$ y $(\mathcal{C}2)$ procedemos de manera similar a como lo hicimos con las aplicaciones $\mathcal{A}$. Los casos son:

- Para el caso $(\mathcal{C}M)N$ deseamos encontrar una expresión de la forma $\mathcal{C}X$, tal que la evaluación de ambos términos produzca el mismo resultado. El asunto consiste en encontrar la X idónea.

  Como la expresión $\mathcal{C}M$ está en el sitio que ocuparía una función en la aplicación $(\mathcal{C}M)N$, su continuación inmediata es la aplicación de su correspondiente función f, que aún es desconocida, con el término N, i.e. fN. Denotamos por k a la continuación global de la aplicación $(\mathcal{C}M)N$. Al componer las dos partes obtenemos la parte funcional de la continuación de $\mathcal{C}M$, ésta es $\lambda f.k(fN)$, la cual, siguiendo las directrices de la aplicación $\mathcal{C}$, será en su momento el argumento de M. Esta continuación debe abortar su contexto una vez invocada, como indica arriba la regla (Inv), para llevar a cabo esta tarea recurrimos al operador $\mathcal{A}$. El argumento final de M es $\lambda f.\mathcal{A}(k(fN))$ y la X es $\lambda k.M(\lambda f.\mathcal{A}(k(fN)))$.

  La reducción a la que llegamos es:

  $$(\mathcal{C}M)N \xrightarrow{\mathcal{C}_L} \mathcal{C}(\lambda k.M(\lambda f.\mathcal{A}(k(fN)))) \qquad (\mathcal{C}_L)$$

- Tomemos ahora el caso $M(\mathcal{C}N)$, donde M es un valor. La reducción que buscamos tiene la forma $\mathcal{C}X$.

[14] Al escribir $\xrightarrow{\beta_v}{}^{+}$ nos referimos a que la reducción β_v se aplica una o más veces.

Primero veamos que M se aplica al argumento desconocido v, así que la continuación de $\mathcal{C}N$ es Mv, este resultado será pasado a la continuación global de la aplicación $M(\mathcal{C}N)$, denotada por k, por lo que la parte funcional de la continuación de $\mathcal{C}N$ es $\lambda v.k(Mv)$. Con un argumento análogo al del caso anterior, la X adopta la forma $\lambda k.N(\lambda v.\mathcal{A}(k(Mv)))$. Con esto, la reducción buscada es:

$$N(\mathcal{C}M) \xrightarrow{\mathcal{C}_R} \mathcal{C}(\lambda k.N(\lambda v.\mathcal{A}(k(Mv)))) \qquad (\mathcal{C}_R)$$

- Para finalizar, tomemos el caso cuando el contexto es vacío. Se debe de producir la aplicación de M, argumento de $\mathcal{C}$, con la abstracción de procedimiento $\lambda z.\mathcal{A}(E[z])$. Como vimos para $\mathcal{A}$, esto no es una reducción propia sino una *regla de cómputo*:

$$\mathcal{C}M \rhd_{\mathcal{C}} M(\lambda x.\mathcal{A}x) \qquad (\mathcal{C}_T)$$

Con esto logramos simular al *sistema de reescritura* $\mathcal{C}$. Ahora deseamos definir la *relación de reducción de un-paso* $\longrightarrow_{\mathsf{C}}$, esta relación conecta términos que únicamente difieren en dos subtérminos pero donde uno conduce al otro a través de una regla de reducción.

Sea $\xrightarrow{C} = \xrightarrow{\mathcal{C}_L} \cup \xrightarrow{\mathcal{C}_R} \cup \xrightarrow{\mathcal{A}_L} \cup \xrightarrow{\mathcal{A}_R} \cup \xrightarrow{\beta_v}$. La *relación de reducción de un-paso* $\longrightarrow_{\mathsf{C}}$ es la cerradura de $\xrightarrow{C}$, y se define como,

$$\begin{aligned}
&M \xrightarrow{C} N \Rightarrow M \longrightarrow_{\mathsf{C}} N\\
&M \longrightarrow_{\mathsf{C}} N \Rightarrow \lambda x.M \longrightarrow_{\mathsf{C}} \lambda x.N\\
&M \longrightarrow_{\mathsf{C}} N \Rightarrow ZM \longrightarrow_{\mathsf{C}} ZN, MZ \longrightarrow_{\mathsf{C}} NZ, \text{ con } Z \text{ término de } \lambda\mathcal{C}\\
&M \longrightarrow_{\mathsf{C}} N \Rightarrow \mathcal{C}M \longrightarrow_{\mathsf{C}} \mathcal{C}N\\
&M \longrightarrow_{\mathsf{C}} N \Rightarrow \mathcal{A}M \longrightarrow_{\mathsf{C}} \mathcal{A}N
\end{aligned}$$

La *reducción*-C está representada por $\rightarrow^*_{\mathsf{C}}$ y se refiere a la cerradura transitiva y reflexiva de $\longrightarrow_{\mathsf{C}}$. La *igualdad*-$\mathsf{C}$, simbolizada por $=_{\mathsf{C}}$, indica la cerradura transitiva, reflexiva y simétrica de $\longrightarrow_{\mathsf{C}}$. Tomamos la unión de $\rightarrow^*_{\mathsf{C}}$ con las reglas de cómputo $\rhd_{\mathcal{A}}$ y $\rhd_{\mathcal{C}}$ para formar la *relación de cómputo* $\blacktriangleright_{\mathsf{C}}$. La relación $\overset{\blacktriangleright}{=}_{\mathsf{C}}$ es la relación de equivalencia que se obtiene de $\blacktriangleright_{\mathsf{C}}$, establece igualdad entre términos bajo reducciones y cómputos, esta es la *igualdad computacional*-C. La semántica del cálculo $\lambda\mathcal{C}$ está dada por la relación $\overset{\blacktriangleright}{=}_{\mathsf{C}}$; sin embargo, se optará por la regla $=_{\mathsf{C}}$, quien ignora a las reglas de cómputo y utiliza únicamente reducciones propias.

Entre las propiedades más relevantes que satisface el cálculo $\lambda\mathcal{C}$ mencionamos consistencia y estandarización. Consistencia quiere decir que se cumple la *confluencia* en las reducciones y cómputos que tienen un inicio en común. Mientras que estandarización se refiere a que el orden en el que se simplifican las *reducciones*-C es estándar, ya que se siguen ciertos patrones bien definidos. En nuestro caso la reducción que siempre se ejecuta primero es la que está más a la izquierda y no forma parte de una abstracción lambda. Estas propiedades las demuestran Felleisen y sus colaboradores en [6].

2.2. Operadores de control construidos con $\mathcal{C}$

El operador $\mathcal{C}$ posee gran expresividad, muestra de ello es la facilidad para definir operadores de control, en esta sección analizamos algunos de ellos.

2.2.1. Abort

En el cálculo $\lambda\mathcal{C}$ de Felleisen encontramos dos nuevos operadores $\mathcal{C}$ y $\mathcal{A}$. Como vimos $\mathcal{A}$ es el operador que aborta la ejecución de un programa, ignorando el contexto actual y continuando con la evaluación del argumento que se le pasa. La regla de la que hablamos es ($\mathcal{C}3$),

$$E[\mathcal{A}M] \mapsto_{\mathcal{C}} M$$

Esta misma acción se obtiene empleando el operador $\mathcal{C}$ de acuerdo a la siguiente definición,

$$\mathcal{A}M =_{def} \mathcal{C}(\lambda u.M)$$

donde u es una variable irrelevante que no es libre en M. Esto se debe a que,

$$\begin{aligned} E[\mathcal{A}M] &=_{def} E[\mathcal{C}(\lambda u.M)] \\ &\mapsto_{\mathcal{C}} (\lambda u.M)(\lambda z.\mathcal{A}(E[z])) \\ &\mapsto_{\mathcal{C}} M \end{aligned}$$

mismo efecto que produce la regla ($\mathcal{C}3$).

2.2.2. call/cc

Al considerar la semántica del operador call/cc del lenguaje SCHEME notamos que difiere de nuestro operador $\mathcal{C}$ en que la continuación actual E es

invocada con el valor que se obtiene de la aplicación $M(\lambda z.\mathcal{A}(E[z]))$, es decir, la evaluación del programa regresa al lugar de inicio. Si tuviéramos que dar una regla de reducción para $\mathsf{call/cc}$ ésta sería,

$$E[\mathsf{call/cc}\, M] \mapsto_{\mathsf{call/cc}} E[M(\lambda z.\mathcal{A}(E[z]))]$$

Sin embargo, Griffin define el operador $\mathsf{call/cc}$ en términos del operador $\mathcal{C}$ y también $\mathcal{C}$ en términos de $\mathsf{call/cc}$ como,

$$\mathsf{call/cc}\, M =_{def} \mathcal{C}(\lambda k.k(Mk)) \qquad\qquad \mathcal{C}M =_{def} \mathsf{call/cc}(\lambda k.\mathcal{A}(Mk))$$

En [10] Griffin no realiza las reducciones que nos aseguran que estas definiciones efectivamente funcionan, nosotros las damos a continuación.

Por un lado tenemos,

$$\begin{aligned} E[\mathcal{C}M] =_{def} E[\mathsf{call/cc}\,(\lambda k.\mathcal{A}(Mk))] &\mapsto_{\mathsf{call/cc}} E[(\lambda k.\mathcal{A}(Mk))(\lambda z.\mathcal{A}(E[z]))] \\ &\mapsto_{\mathcal{C}} E[\mathcal{A}(M(\lambda z.\mathcal{A}(E[z])))] \\ &\mapsto_{\mathcal{C}} M(\lambda z.\mathcal{A}(E[z])) \end{aligned}$$

este comportamiento es precisamente el de la regla ($\mathcal{C}2$).

Para corroborar $\mathsf{call/cc}\, M =_{def} \mathcal{C}(\lambda k.k(Mk))$ realizamos la derivación en una versión de *llamada-por-nombre* del cálculo $\lambda\mathcal{C}$ [15]

$$\begin{aligned} E[\mathsf{call/cc}\, M] =_{def} E[\mathcal{C}(\lambda k.k(Mk))] &\mapsto_{\mathcal{C}} (\lambda k.k(Mk))(\lambda z.\mathcal{A}(E[z])) \\ &\mapsto_{\mathcal{C}} (\lambda z.\mathcal{A}(E[z]))(M(\lambda z.\mathcal{A}(E[z]))) \\ &\mapsto_{\mathcal{C}} \mathcal{A}(E[M(\lambda z.\mathcal{A}(E[z]))]) \\ &\mapsto_{\mathcal{C}} E[M(\lambda z.\mathcal{A}(E[z]))] \end{aligned}$$

como lo indica la regla ($\mapsto_{\mathsf{call/cc}}$).

2.2.3. catch/throw

El operador $\mathsf{catch/throw}$ evalúa un término M, si se obtiene un valor V entonces tal es el resultado final de la expresión, este es un regreso normal. Por otro lado, si en la evaluación de M se lanza un valor V' a la continuación que

[15] En la versión de *llamada-por-nombre* los contextos de evaluación son $E ::= [\,] \mid EN$ y la reducción β_n es $E[(\lambda x.M)N] \mapsto_{\beta_n} E[M[x := N]]$, las demás reducciones son idénticas que en la versión de *llamada-por-valor*. Para mayor detalle refiérase a [10].

representa el contexto donde el operador catch/throw fue definido entonces se suspenden los cómputos y se devuelve V', este es un regreso por excepción. Para dar su definición utilizaremos la instrucción call/cc porque se busca regresar al punto del que se parte, ya sea para continuar normalmente con la ejecución del programa o para devolver un valor que es lanzado(throw) cuando se satisface cierta condición. Sabemos que call/cc se expresa en términos de $\mathcal{C}$, así catch/throw se define también a través de $\mathcal{C}$. Consideremos la evaluación de $E_0[\mathsf{call/cc}(\lambda j.M)]$ como un catch que identifica a la continuación actual E_0 con la variable j. Evaluando esta expresión y tomando $N = \lambda z.\mathcal{A}(E_0[z])$ se tiene,

$$\begin{aligned} E_0[\mathsf{call/cc}(\lambda j.M)] &= E_0[\mathcal{C}(\lambda k.k((\lambda j.M)k))] \\ &\mapsto_{\mathcal{C}} (\lambda k.k((\lambda j.M)k))N \\ &\mapsto_{\mathcal{C}} N((\lambda j.M)N) \\ &\mapsto_{\mathcal{C}} N(M[j := N]) \end{aligned}$$

Si la j nunca es invocada o lanzada entonces esta expresión regresa normalmente, esto es, si $M[j := N] \mapsto_{\mathcal{C}}^{*} V$ entonces la ejecución continúa su flujo normal,

$$\begin{aligned} N(M[j := N]) &\mapsto_{\mathcal{C}}^{*} NV \\ &\mapsto_{\mathcal{C}} \mathcal{A}(E_0[V]) \\ &\mapsto_{\mathcal{C}} E_0[V] \end{aligned}$$

Si por el contrario, al evaluar el término M la j es invocada con un valor V' bajo un contexto E_1, como en $E_1[jV']$, entonces olvidamos E_1 y V' es lanzada a la continuación que representa j obteniendo $E_0[V']$, esto representa un regreso por excepción. En este caso los pasos de la evaluación son,

$$\begin{aligned} N(M[j := N]) &\mapsto_{\mathcal{C}}^{*} E_1[jV'[j := N]] \\ &\mapsto_{\mathcal{C}} E_1[NV'] \\ &\mapsto_{\mathcal{C}} E_1[\mathcal{A}(E_0[V'])] \\ &\mapsto_{\mathcal{C}} E_0[V'] \end{aligned}$$

Vemos que el contexto E_1 es abandonado y que la evaluación prosigue con el valor V' dentro de la continuación inicial.

2.3. El tipo de los operadores $\mathcal{C}$ y $\mathcal{A}$

Hemos considerado a los operadores $\mathcal{C}$ y $\mathcal{A}$ sin tipos, tratemos ahora de encontrar una tipificación coherente para dichos operadores usando la idea que T.G. Griffin propone en [10]. Esto nos conducirá al *isomorfismo de Curry-Howard* en el que los tipos del cálculo $\lambda\mathcal{C}$ se corresponden con fórmulas de la lógica clásica. El estilo que se manejará para esta nueva tipificación es *a la Curry*. Debemos aclarar que la semántica en la que Griffin basa la obtención de tipos no en la reducción $\longrightarrow_{\mathsf{C}}$ dada por Felleisen sino en el *sistema de reescritura* $\mathcal{C}$.

Hasta este momento las reglas de tipificación con las que contamos son las de $\lambda^{\rightarrow}$,

$$\frac{}{\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}}\ (RVar) \qquad \frac{\Gamma, x : \mathsf{T} \vdash M : \mathsf{S}}{\Gamma \vdash \lambda x.M : \mathsf{T} \to \mathsf{S}}\ (RFun)$$

$$\frac{\Gamma \vdash M_1 : \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash M_2 : \mathsf{T}}{\Gamma \vdash M_1 M_2 : \mathsf{S}}\ (RApp)$$

Para asignar un tipo al operador de control $\mathcal{C}$ empecemos analizando la regla de reducción ($\mathcal{C}2$),

$$E[\mathcal{C}M] \mapsto_{\mathcal{C}} M(\lambda z.\mathcal{A}(E[z]))$$

Si suponemos que $E[\,] : \mathsf{T} \to \mathsf{S}$ entonces la expresión $E[Q] : \mathsf{S}$, donde $Q : \mathsf{T}$. Esto implica que podemos darle a la expresión $\lambda z.\mathcal{A}(E[z])$ el tipo $\mathsf{T} \to \mathsf{S}$, ya que para cualquier valor V de tipo T se aprecia que,

$$(\lambda z.\mathcal{A}(E[z]))V \mapsto_{\mathcal{C}} \mathcal{A}(E[V]) \mapsto_{\mathcal{C}} E[V],$$

y el tipo de $E[V]$ es S. Deseamos que la propiedad de reducción del sujeto se cumpla, es decir, buscamos que los tipos se preserven al reducir expresiones; por consiguiente, M debe ser de tipo función $(\mathsf{T} \to \mathsf{S}) \to \mathsf{S}$ para que $M(\lambda z.\mathcal{A}(E[z]))$ sea de tipo S, mismo tipo que $E[\mathcal{C}M]$. Ahora, asignamos a la aplicación $\mathcal{C}M$ el tipo T, porque es el tipo de la expresión que ocupa el *hueco* del contexto E, esto ocurre siempre y cuando M tenga tipo $(\mathsf{T} \to \mathsf{S}) \to \mathsf{S}$. La regla a la que llegamos es,

$$\frac{\Gamma \vdash M : (\mathsf{T} \to \mathsf{S}) \to \mathsf{S}}{\Gamma \vdash \mathcal{C}M : \mathsf{T}}$$

Consideremos detenidamente la tipificación anterior. Si el término cerrado N tiene tipo S, entonces a la expresión $\mathcal{A}N =_{def} \mathcal{C}(\lambda u.N)$ se le puede dar cualquier tipo T debido a que este tipo es obtenido de $\lambda u.N : (\mathsf{T} \to \mathsf{S}) \to \mathsf{S}$, donde a la variable u se les asigna el tipo $\mathsf{T} \to \mathsf{S}$; sin embargo, la u es irrelevante y no figura en N por lo que el tipo T no tiene restricción alguna. Lógicamente esto indica que cualquier proposición es inferible, lo cual arruina nuestro sistema de derivación; incluso, si consideramos por un momento los tipos como fórmulas lógicas tenemos la regla,

$$\frac{\Gamma \vdash (\mathsf{T} \to \mathsf{S}) \to \mathsf{S}}{\Gamma \vdash \mathsf{T}}$$

la cual es inconsistente, tómese T como falso y S como verdadero. Por lo tanto, para mantener la consistencia el tipo de S debe ser $\bot$[16]. Tomando el tipo $\bot$ como atómico y sobrecargando el conectivo lógico de negación para tipos del siguiente modo $\neg\mathsf{T} =_{def} \mathsf{T} \to \bot$, podemos asignar un tipo lógicamente consistente para $\mathcal{C}M$. Si el tipo de M es $\neg\neg\mathsf{T} =_{def} (\mathsf{T} \to \bot) \to \bot$ entonces el tipo de $\mathcal{C}M$ es T.

La nueva regla de tipificación es:

$$\frac{\Gamma \vdash M : \neg\neg\mathsf{T}}{\Gamma \vdash \mathcal{C}M : \mathsf{T}}\ (R\mathcal{C})$$

Aparentemente hemos logrado nuestro objetivo; sin embargo, la tipificación actual presenta un inconveniente, la regla $(\mathcal{C}2)$ requiere que la expresión entera $E[\mathcal{C}M]$ sea de tipo $\bot$, porque acordamos que el tipo de $E[Q]$ es S y concluimos que S tiene que ser $\bot$. Pero no hay términos cerrados de tipo $\bot$ y la regla $(\mathcal{C}2)$ no tiene caso. Griffin corrige este problema al modificar la semántica del *sistema de reescritura* $\mathcal{C}$. Al evaluar el término M con tipo T se procede a envolverlo en la expresión $\mathcal{C}(\lambda k.kM)$, siendo $\neg\mathsf{T}$ el tipo de k. Esta última expresión se evalúa usando las reglas del *sistema de reescritura* $\mathcal{C}$ únicamente dentro del cuerpo de la expresión $\mathcal{C}(\lambda k.\cdots)$, como el término que está representado por $\cdots$ es de tipo $\bot$ entonces es válido utilizar la reglas $\to_{\mathcal{C}}$ dentro de la abstracción sin provocar un conflicto de tipos. Formalmente, esta nueva semántica $(\mapsto_t)$ queda definida como la unión de las reglas:

$$\begin{aligned}
\mathcal{C}(\lambda k.E[(\lambda x.M)V]) &\mapsto_{t\beta_v} \mathcal{C}(\lambda k.E[M[x := V]]) \\
\mathcal{C}(\lambda k.E[\mathcal{C}M]) &\mapsto_{t\mathcal{C}} \mathcal{C}(\lambda k.M(\lambda z.\mathcal{A}(E[z]))) \\
\mathcal{C}(\lambda k.kV) &\mapsto_{\mathcal{C}_e} V \quad \text{con } k \notin FV(V)
\end{aligned}$$

[16]En este capítulo ocuparemos la notación $\bot$ para referirnos al tipo vacío o void.

Se ha omitido el operador $\mathcal{A}$ porque puede ser expresado en términos de $\mathcal{C}$. La expresión $E[\cdots]$ sigue siendo de tipo $\bot$ pero ahora el término final es de tipo T.

Una expresión está en forma t-normal si ninguna de las reglas anteriores puede ser usada.

Definición 2.2. *La evaluación del término cerrado M con tipo T es N si $\mathcal{C}(\lambda k.kM) \mapsto_t^* N$ y N está en forma normal t*

Griffin presenta el siguiente resultado que indica que la semántica definida por $\mapsto_t$ actúa de manera muy parecida a como lo hace el *sistema de reescritura* $\mathcal{C}$ y que la única violación de tipos se debe a la regla de cómputo ($\mathcal{C}_T$).

Lema 2.1. *Si $\mathcal{C}(\lambda k.kM) \mapsto_{\mathcal{C}}^* V$ entonces $\mathcal{C}(\lambda k.kM) \mapsto_t^* N$, donde N es V, $\mathcal{C}(\lambda k.kV')$ o $\mathcal{C}(\lambda k.V')$ con $V = V'[k := \lambda x.\mathcal{A}(x)]$.*

2.3.1. Isomorfismo de Curry-Howard

En su artículo, Griffin indica la tipificación para $\mathcal{C}$ pero no formaliza el cumplimiento del *isomorfismo de Curry-Howard* para el cálculo $\lambda\mathcal{C}$ y la lógica clásica, nosotros mostramos esta correspondencia de manera rigurosa en esta sección.

La transformación biunívoca de fórmulas a tipos $(-)^*$ se define como,

$$\bot^* = \bot$$
$$(\varphi \to \psi)^* = \varphi^* \to \psi^*$$

Debemos aclarar que en principio se tenía $\bot^* = \mathsf{void}$ pero hemos preferido la notación $\bot$ en lugar de void, por lo que $\bot^* = \bot$ está bien definida. El mapeo $(-)^\circ$, que toma tipos y regresa fórmulas, es la inversa de $(-)^*$. Estas transformaciones se pueden extender a contextos lógicos y contextos de tipos trivialmente.

El *isomorfismo de Curry-Howard* para el cálculo $\lambda\mathcal{C}$ y la lógica clásica se enuncia como sigue,

Teorema 2.1. (Isomorfismo de Curry-Howard) *Sea φ una fórmula de la lógica clásica, T un tipo de $\lambda\mathcal{C}$, Γ un contexto lógico y Δ un contexto de tipos.*

1. *Si $\Delta \vdash M : \mathsf{T}$ en $\lambda\mathcal{C}$ entonces $\Delta^\circ \vdash_c \mathsf{T}^\circ$ en $NK(\to, \bot)$.*

II. *Si* $\Gamma \vdash_c \varphi$ *en* $NK(\to, \bot)$ *entonces existe un término* M *tal que* $\Gamma^* \vdash M : \varphi^*$ *en* $\lambda\mathcal{C}$.

Demostración. La prueba de I es por inducción sobre la derivación de tipos $\Delta \vdash M : \mathsf{T}$. La base de inducción y la mayoría de los casos para el paso inductivo fueron vistos en la sección 1.3.1. Falta probar cuando la derivación termina con $\Delta \vdash \mathcal{C}M : \mathsf{T}$. Por hipótesis de inducción es cierto en $NK(\to, \bot)$ que $\Delta^\circ \vdash_c \neg\neg\mathsf{T}^\circ$, esto puede verse como $\Delta^\circ, \mathsf{T}^\circ \to \bot^\circ \vdash_c \bot^\circ$. Usando la regla $(\neg\neg E)$ se satisface $\Delta^\circ \vdash_c \mathsf{T}^\circ$, que es la derivación a la que queríamos llegar.

Para probar II realizamos inducción sobre la derivación lógica $\Gamma \vdash_c \varphi$. Gran parte de la demostración se encuentra en la sección 1.3.1. Debemos mostrar ahora que se cumple cuando la última regla usada es $(\neg\neg E)$. La hipótesis de inducción nos dice que existe M' tal que $\Gamma^* \vdash M' : \neg\neg\varphi^*$ en $\lambda\mathcal{C}$. Usando la regla $(R\mathcal{C})$ observamos que el término $\mathcal{C}M'$ cumple que $\Gamma^* \vdash \mathcal{C}M' : \varphi^*$, así tomando $M = \mathcal{C}M'$ terminamos. □

Apreciamos que la aplicación $\mathcal{C}M$ corresponde a una prueba clásica para la regla de la eliminación de la doble negación. Además, a partir de la regla $(R\mathcal{C})$ y del hecho de que $\mathcal{A}M =_{def} \mathcal{C}(\lambda u.M)$ vemos que el tipo para el operador $\mathcal{A}$ es,

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathcal{A}M : \mathsf{T}}\ (R\mathcal{A})$$

Es sencillo verificar la correspondencia entre $(R\mathcal{A})$ y la regla previamente definida como eliminación de $\bot$ o *ex-falso-quodlibet*.

2.4. Definición de tipos productos y sumas

En lógica clásica tenemos la capacidad de definir cualquier conectivo lógico mediante la negación y la implicación, lo cual no ocurre en la lógica minimal o intuicionista (véase el Corolario 6.7.7 de [31]). Ahora estamos interesados en encontrar e investigar las propiedades computacionales de los términos en $\lambda\mathcal{C}$ que definen, del lado de la lógica, a la conjunción y a la disyunción. Veremos que los operadores de par ordenado, proyección, inyección y análisis por casos, que por el *isomorfismo de Curry-Howard* no pueden ser definidos en $\lambda^\to$, pueden ser expresados en términos del operador $\mathcal{C}$ de Felleisen.

En esta sección en ocasiones usaremos anotaciones de tipos como superíndices para algunas variables.

Nos serviremos de las reglas de eliminación e introducción del $\wedge$ y $\vee$ en el sistema clásico, exportando dichas reglas del sistema intuicionista dado en el capítulo 1 para obtener el cálculo proposicional clásico $NK(\rightarrow, \bot, \wedge, \vee)$.

2.4.1. Conjunción

En [10] simplemente se define la conjunción como,

$$\varphi \wedge \psi =_{def} \neg(\varphi \rightarrow \neg\psi)$$

Nosotros profundizamos un poco más y comprobamos que $\vdash_c \varphi \wedge \psi \leftrightarrow \neg(\varphi \rightarrow \neg\psi)$, para esto necesitamos un resultado previo.

Proposición 2.2.

$$\vdash_i (\varphi \rightarrow \neg\psi) \rightarrow \psi \rightarrow \neg\varphi \qquad \text{(Prop 2.2)}$$

Demostración. Basta demostrar que $\Delta \vdash_i \bot$, con $\Delta = \{\varphi \rightarrow \neg\psi, \psi, \varphi\}$.

$$\dfrac{\dfrac{\dfrac{}{\Delta \vdash_i \varphi \rightarrow \neg\psi}(Ax) \quad \dfrac{}{\Delta \vdash_i \varphi}(Ax)}{\Delta \vdash_i \neg\psi}(\rightarrow E) \quad \dfrac{}{\Delta \vdash_i \psi}(Ax)}{\Delta \vdash_i \bot}(\rightarrow E)$$

Hemos probado que el secuente es derivable en la lógica intuicionista; por lo tanto, también lo es en la lógica clásica. □

Procedemos con la siguiente proposición,

Proposición 2.3. *Es cierto que* $\vdash_c \varphi \wedge \psi \leftrightarrow \neg(\varphi \rightarrow \neg\psi)$

Demostración. Se analizan por separado $\rightarrow$ y $\leftarrow$

- $\varphi \wedge \psi \vdash_c \neg(\varphi \rightarrow \neg\psi)$. Si consideramos $\Gamma = \{\varphi \wedge \psi, \varphi \rightarrow \neg\psi\}$, entonces,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_c \varphi \rightarrow \neg\psi}(Ax) \quad \dfrac{\dfrac{}{\Gamma \vdash_c \varphi \wedge \psi}(Ax)}{\Gamma \vdash_c \varphi}(\wedge_1 E)}{\Gamma \vdash_c \neg\psi}(\rightarrow E) \quad \dfrac{}{\Gamma \vdash_c \varphi \wedge \psi}(Ax)}{\Gamma \vdash_c \psi}(\wedge_2 E)}{\Gamma \vdash_c \bot}(\rightarrow E)}{\varphi \wedge \psi \vdash_c \neg(\varphi \rightarrow \neg\psi)}(\rightarrow I)$$

- $\neg(\varphi \rightarrow \neg\psi) \vdash_c \varphi \wedge \psi$. Se tienen las derivaciones,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(\varphi \to \neg\psi), \neg\psi, \varphi \vdash_c \neg\psi}\,(Ax)}{\neg(\varphi \to \neg\psi), \neg\psi \vdash_c \varphi \to \neg\psi}\,(\to I) \quad \dfrac{}{\neg(\varphi \to \neg\psi), \neg\psi \vdash_c \neg(\varphi \to \neg\psi)}\,(Ax)}{\neg(\varphi \to \neg\psi), \neg\psi \vdash_c \bot}\,(\to E)}{\neg(\varphi \to \neg\psi) \vdash_c \psi}\,(\neg\neg E)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(\varphi \to \neg\psi), \neg\varphi, \psi \vdash_c \neg\varphi}\,(Ax)}{\neg(\varphi \to \neg\psi), \neg\varphi \vdash_c \psi \to \neg\varphi}\,(\to I)}{\neg(\varphi \to \neg\psi), \neg\varphi \vdash_c \varphi \to \neg\psi}\,(\text{Prop 2.2}) \quad \dfrac{}{\neg(\varphi \to \neg\psi), \neg\varphi \vdash_c \neg(\varphi \to \neg\psi)}\,(Ax)}{\neg(\varphi \to \neg\psi), \neg\varphi \vdash_c \bot}\,(\to E)}{\neg(\varphi \to \neg\psi) \vdash_c \varphi}\,(\neg\neg E)$$

Por las dos derivaciones anteriores y la regla $(\wedge I)$ llegamos a que

$$\neg(\varphi \to \neg\psi) \vdash_c \varphi \wedge \psi$$

□

Si M_1 y M_2 tienen tipos φ y ψ, respectivamente, entonces de acuerdo con [10] definimos par ordenado como,

$$\langle M_1, M_2 \rangle =_{def} \lambda f^{\varphi \to \neg\psi}.f M_1 M_2$$

Siendo de tipo $\neg(\varphi \to \neg\psi)$ y representando la regla $(\wedge\, I)$.

La operación de proyección del término M con tipo $\neg(\varphi \to \neg\psi)$, equivale a la regla $(\wedge_i E)$. Griffin en [10] define proyección como,

$$\pi_i(M) =_{def} \mathcal{C}(\lambda v^{\neg\chi_i}.M(\lambda x_1^{\varphi} x_2^{\psi}.v x_i)) \quad \text{donde } i = \{1, 2\},\ \chi_1 = \varphi \text{ y } \chi_2 = \psi$$

Estudiemos esta definición. La expresión $R = \lambda x_1^{\varphi} x_2^{\psi}.v x_i$ es de tipo $\varphi \to \neg\psi$, luego la aplicación MR es de tipo $\bot$, así el tipo de $\lambda v^{\neg\chi_i}.MR$ es $\neg\neg\chi_i$, por último $\mathcal{C}(\lambda v^{\neg\chi_i}.MR)$ tiene tipo χ_i, con lo que obtenemos el resultado buscado.

Para la obtención de las propiedades computacionales de los términos introducidos en esta sección, y sólo para este fin, se usarán las reglas del *sistema de reescritura* $\mathcal{C}$ de la sección 2.1, bajo el entendido de que los términos tipificados deben ser evaluados usando las reglas $\to_t$, definidas en la sección 2.3.

Los operadores de par ordenado y proyección deben satisfacer que $E[\pi_i\langle M_1, M_2\rangle] \to_{\mathcal{C}} E[V_i]$, con V_i siendo el valor al que se reduce M_i. Tomando $N = \lambda z.\mathcal{A}(E[z])$ los pasos son,

$$\begin{aligned} E[\pi_i\langle M_1, M_2\rangle] &= E[\mathcal{C}(\lambda v.\langle M_1, M_2\rangle(\lambda x_1 x_2.vx_i))] \\ &\mapsto^*_{\mathcal{C}} \langle M_1, M_2\rangle(\lambda x_1 x_2.Nx_i) \\ &= (\lambda f.fM_1M_2)(\lambda x_1 x_2.Nx_i) \\ &\mapsto_{\mathcal{C}} (\lambda x_1 x_2.Nx_i)M_1M_2 \\ &\mapsto^*_{\mathcal{C}} Nx_i[x_1 := V_1][x_2 := V_2] \\ &= NV_i \\ &\mapsto_{\mathcal{C}} \mathcal{A}(E[V_i]) \\ &\mapsto_{\mathcal{C}} E[V_i] \end{aligned}$$

En nuestro análisis de *llamada-por-valor* estamos obligados a reducir primero M_1 y M_2 a valores V_1 y V_2, respectivamente, para continuar con las reducciones de los subtérminos de E, esto origina una evaluación bastante ineficiente debido a que se realizan varias reducciones inútiles ya que sólo ocuparemos uno de los dos valores. Para mejorar esta situación tomaremos la propuesta que Griffin ofrece en [10] pero daremos un explicación más convincente. En primer lugar observemos que,

$$\neg((T \to \varphi) \to \neg(T \to \psi)) \equiv \neg T \vee (\neg T \wedge \varphi) \vee (\neg T \wedge \psi) \vee (\varphi \wedge \psi)$$

Si la interpretación de T es $\top$ entonces lo anterior se simplifica a $\varphi \wedge \psi$. Para esto asignamos a T el tipo unit de la sección 1.2.8 con único valor $*$, entonces la conjunción queda redefinida como,

$$\varphi \wedge \psi =_{def} \neg((\mathsf{unit} \to \varphi) \to \neg(\mathsf{unit} \to \psi))$$

La redefinición de par ordenado y proyección se muestra a continuación,

$$\langle M_1, M_2\rangle =_{def} \lambda f.f(\lambda u.M_1)(\lambda u.M_2) \text{ con } u \notin FV(M_1) \cup FV(M_2)$$

$$\pi_i(M) =_{def} (\mathcal{C}(\lambda v.M\lambda x_1 x_2.vx_i))*$$

En la derivación siguiente $N = \lambda z.\mathcal{A}(E[z])$, además emplearemos la regla

($\mathcal{C}_L$) de la sección 2.1.1. Entonces tenemos,

$$\begin{aligned}
E[\pi_i\langle M_1, M_2\rangle] &= E[(\mathcal{C}(\lambda v.\langle M_1, M_2\rangle \lambda x_1 x_2.vx_i))*] \\
&\xrightarrow{\mathcal{C}_L} E[\mathcal{C}(\lambda k.(\lambda v.\langle M_1, M_2\rangle \lambda x_1 x_2.vx_i)(\lambda f.\mathcal{A}(k(f*))))] \\
&\mapsto_{\mathcal{C}} (\lambda k.(\lambda v.\langle M_1, M_2\rangle \lambda x_1 x_2.vx_i)(\lambda f.\mathcal{A}(k(f*))))N \\
&\mapsto_{\mathcal{C}} (\lambda v.\langle M_1, M_2\rangle \lambda x_1 x_2.vx_i)(\lambda f.\mathcal{A}(N(f*))) \\
&\mapsto_{\mathcal{C}} \langle M_1, M_2\rangle (\lambda x_1 x_2.(\lambda f.\mathcal{A}(N(f*)))x_i) \\
&= (\lambda g.g(\lambda u.M_1)(\lambda u.M_2))(\lambda x_1 x_2.(\lambda f.\mathcal{A}(N(f*)))x_i) \\
&\mapsto_{\mathcal{C}} (\lambda x_1 x_2.(\lambda f.\mathcal{A}(N(f*)))x_i)(\lambda u.M_1)(\lambda u.M_2) \\
&\mapsto^*_{\mathcal{C}} (\lambda f.\mathcal{A}(N(f*)))(\lambda u.M_i) \\
&\mapsto_{\mathcal{C}} \mathcal{A}(N((\lambda u.M_i)*)) \\
&\mapsto_{\mathcal{C}} N((\lambda u.M_i)*) \\
&\mapsto_{\mathcal{C}} N(M_i[u := *]) \\
&= N(M_i) \\
&\mapsto^*_{\mathcal{C}} N(V_i) \\
&\mapsto_{\mathcal{C}} \mathcal{A}(E[V_i]) \\
&\mapsto_{\mathcal{C}} E[V_i]
\end{aligned}$$

Así llegamos a la reduccion ($\mapsto_{\pi_i}$)

$$E[\pi_i\langle M_1, M_2\rangle] \mapsto_{\pi_i} E[V_i]$$

en la cual ya no se requiere calcular tanto el valor V_1 como V_2 de las expresiones M_1 y M_2, respectivamente, sino sólo el valor V_i correspondiente a M_i.

2.4.2. Disyunción

La disyunción está definida clásicamente en [10] como,

$$\varphi \vee \psi =_{def} \neg\varphi \to \neg\neg\psi$$

Nosotros corroboramos esta definición con la siguiente proposición.

Proposición 2.4. *El secuente* $\vdash_c \varphi \vee \psi \leftrightarrow \neg\varphi \to \neg\neg\psi$ *es derivable.*

Demostración. Se analizan dos situaciones.

- $\varphi \vee \psi \vdash_c \neg\varphi \to \neg\neg\psi$. Tomando $\Delta = \{\varphi \vee \psi, \neg\varphi, \neg\psi\}$ se tiene,

$$\dfrac{\dfrac{}{\Delta, \varphi \vdash_c \varphi}\,(Ax) \quad \dfrac{}{\Delta, \varphi \vdash_c \neg\varphi}\,(Ax)}{\Delta, \varphi \vdash_c \bot}\,(\to E)$$

  También,

$$\dfrac{\dfrac{}{\Delta, \psi \vdash_c \psi}\,(Ax) \quad \dfrac{}{\Delta, \psi \vdash_c \neg\psi}\,(Ax)}{\Delta, \psi \vdash_c \bot}\,(\to E)$$

  Además,

$$\dfrac{}{\Delta \vdash_c \varphi \vee \psi}\,(Ax)$$

  Por las tres derivaciones previas y la regla $(\vee E)$ obtenemos,

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta \vdash_c \bot}\,(\vee E)}{\varphi \vee \psi, \neg\varphi \vdash_c \neg\neg\psi}\,(\to I)}{\varphi \vee \psi \vdash_c \neg\varphi \to (\neg\neg\psi)}\,(\to I)$$

- $\neg\varphi \to \neg\neg\psi \vdash_c \varphi \vee \psi$. Sabemos que en la lógica clásica la regla de *tercero excluído* es válida,

$$\dfrac{}{\neg\varphi \to \neg\neg\psi \vdash_c \varphi \vee \neg\varphi}\,(TE)$$

  Observemos las siguientes dos derivaciones,

$$\dfrac{\dfrac{}{\neg\varphi \to \neg\neg\psi, \varphi \vdash_c \varphi}\,(Ax)}{\neg\varphi \to \neg\neg\psi, \varphi \vdash_c \varphi \vee \psi}\,(\vee I)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\varphi \to \neg\neg\psi, \neg\varphi \vdash_c \neg\varphi}\,(Ax) \quad \dfrac{}{\neg\varphi \to \neg\neg\psi, \neg\varphi \vdash_c \neg\varphi \to \neg\neg\psi}\,(Ax)}{\neg\varphi \to \neg\neg\psi, \neg\varphi \vdash_c \neg\neg\psi}\,(\to E)}{\neg\varphi \to \neg\neg\psi, \neg\varphi \vdash_c \psi}\,(\neg\neg E)}{\neg\varphi \to \neg\neg\psi, \neg\varphi \vdash_c \varphi \vee \psi}\,(\vee I)$$

Con estos resultados y la regla $(\vee E)$ concluimos,

$$\neg\varphi \to \neg\neg\psi \vdash_c \varphi \vee \psi$$

□

La operación de inyección, que representa la regla $(\vee_i I)$, se define en [10] como,

$$\mathsf{inj}_i(M^{\chi_i}) =_{def} \lambda f_1^{\neg\varphi} f_2^{\neg\psi}.f_i M \quad \text{con } \chi_1 = \varphi \text{ y } \chi_2 = \psi$$

es fácil ver que el tipo de $\mathsf{inj}_i(M^{\chi_i})$ es $\neg\varphi \to \neg\neg\psi$, esto no es más que, $\varphi \vee \psi$.

Suponiendo que la expresión M es de tipo $\neg\varphi \to \neg\neg\psi$, la operación de análisis por casos se expresa también en [10] como,

$$\mathsf{case}\, M\{F_1 \mid F_2\} =_{def} \mathcal{C}(\lambda v^{\neg\tau}.MG_1G_2)$$

dado que $F_i = \lambda x^{\chi_i}.P_i$ y $G_i = \lambda y^{\chi_i}.v(F_i y)$, con tipos respectivos $\chi_i \to \tau$ y $\neg\chi_i$, donde $\chi_1 = \varphi$ y $\chi_2 = \psi$.

Por ende,

$$\begin{aligned} MG_1G_2 &: \bot \\ \lambda v^{\neg\tau}.MG_1G_2 &: \neg\neg\tau \\ \mathcal{C}(\lambda v^{\neg\tau}.MG_1G_2) &: \tau \end{aligned}$$

Al contemplar los tipos de M, F_1 y F_2 verificamos que la expresión $\mathcal{C}(\lambda v^{\neg\tau}.MG_1G_2)$ se relaciona con la regla $(\vee E)$,

$$\frac{\Gamma \vdash_c \varphi \vee \psi \quad \Gamma, \varphi \vdash_c \tau \quad \Gamma, \psi \vdash_c \tau}{\Gamma \vdash_c \tau}$$

Los operadores de inyección y análisis por casos deben cumplir la propiedad computacional $E[\mathsf{case}\ (\mathsf{inj}_i\ (M_i))\{F_1 \mid F_2\}] \mapsto_{\mathsf{case}_i} E[V_i]$ donde $F_i M_i \mapsto_{\mathcal{C}}^* V_i$, puesto que estamos en un esquema de *llamada-por-valor*. Una vez más este procedimiento tiene un pobre desempeño, por lo cual realizamos una mejora como la que se obtuvo para la conjunción. La redefinición de disyunción es,

$$\varphi \vee \psi = \neg(\mathsf{unit} \to \varphi) \to \neg\neg(\mathsf{unit} \to \psi)$$

Ahora los operadores de inyección y análisis por casos toman la forma,

$$\mathsf{inj}_i(M^{\chi_i}) =_{def} \lambda f_1^{\neg(1\to\varphi)} f_2^{\neg(1\to\psi)}.f_i(\lambda u.M)$$

$$\mathsf{case}\, M\{F_1 \mid F_2\} = (\mathcal{C}(\lambda v^{\neg(1\to\tau)}.MG_1G_2))*$$

donde $G_i = \lambda y^{1\to\chi_i}.v(\lambda u.F_i(yu))$.

Si $N = \lambda z.\mathcal{A}(E[z])$ y considerando nuevamente la regla $(\mathcal{C}_L)$, entonces los pasos de la evaluación son,

$$\begin{aligned}
E[\mathsf{case}\ (\mathsf{inj}_i\ (M_i))\{F_1 \mid F_2\}] &= E[(\mathcal{C}(\lambda v.(\mathsf{inj}_i(M_i))G_1G_2))*] \\
&\xrightarrow{\mathcal{C}_L} E[\mathcal{C}(\lambda k(\lambda v.(\mathsf{inj}_i(M_i))G_1G_2)(\lambda f.\mathcal{A}(k(f*))))] \\
&\mapsto^*_{\mathcal{C}} (\lambda v.(\mathsf{inj}_i(M_i))G_1G_2)(\lambda f.\mathcal{A}(N(f*))) \\
&= (\lambda v.(\lambda f_1 f_2.f_i(\lambda u.M))G_1G_2)(\lambda f.\mathcal{A}(N(f*))) \\
&\mapsto_{\mathcal{C}} ((\lambda f_1 f_2.f_i(\lambda u.M_i))G_1G_2)[v := \lambda f.\mathcal{A}(N(f*))] \\
&\mapsto^*_{\mathcal{C}} (G_i[v := \lambda f.\mathcal{A}(N(f*))])(\lambda u.M_i) \\
&= (\lambda y.(\lambda f.\mathcal{A}(N(f*)))(\lambda w.F_i(yw)))(\lambda u.M_i) \\
&\mapsto_{\mathcal{C}} (\lambda f.\mathcal{A}(N(f*)))(\lambda w.F_i((\lambda u.M_i)w)) \\
&\mapsto_{\mathcal{C}} \mathcal{A}(N((\lambda w.F_i((\lambda u.M_i)w))*)) \\
&\mapsto_{\mathcal{C}} N((\lambda w.F_i((\lambda u.M_i)w))*) \\
&\mapsto_{\mathcal{C}} N(F_i((\lambda u.M_i)*)) \\
&\mapsto_{\mathcal{C}} N(F_i(M_i[u := *])) \\
&= N(F_iM_i) \\
&\mapsto^*_{\mathcal{C}} N(V_i) \\
&\mapsto_{\mathcal{C}} \mathcal{A}(E[V_i]) \\
&\mapsto_{\mathcal{C}} E[V_i)]
\end{aligned}$$

Así la reducción $(\mapsto_{\mathsf{case}_i})$

$$E[\mathsf{case}\ (\mathsf{inj}_i\ (M_i))\{F_1 \mid F_2\}] \mapsto_{case_i} E[V_i]$$

se satisface, donde $F_iM_i \mapsto^*_{\mathcal{C}} V_i$.

En este capítulo hemos estudiado el cálculo $\lambda\mathcal{C}$ de Felleisen y con base en el trabajo de Griffin se estableció el *isomorfismo de Curry-Howard* entre $\lambda\mathcal{C}$ y la lógica clásica.

En el capítulo siguiente estudiaremos el cálculo $\lambda\mu$ de Parigot. Su principal característica es el manejo explícito de continuaciones representadas por variables μ. Veremos que para este cálculo también se satisface el *isomorfismo de Curry-Howard* con la lógica clásica.

Capítulo 3

Cálculo $\lambda\mu$

A principios de la década de los noventas M. Parigot en [25] extiende el cálculo λ para formar el cálculo $\lambda\mu$ con el objetivo de proporcionar una interpretación algorítmica de las pruebas lógicas clásicas. Esto se debe a que los operadores de control caracterizan el aspecto algorítmico de la lógica clásica, por lo que Parigot abstrae el mecanismo de operación sobre el control de ejecución en términos μ, así en una abstracción μ se liga una variable que representa una continuación. En el cálculo $\lambda\mu$ es común referirnos directamente a contextos de evaluación y realizar invocación de continuaciones, por esto operadores de control que estudiamos en el capítulo anterior, como son $\mathsf{call/cc}$ y $\mathcal{C}$, pueden ser definidos empleando términos $\lambda\mu$.

Al considerar los tipos asociados a sus expresiones apreciamos que se satisface el *isomorfismo de Curry-Howard* para la lógica clásica y el cálculo $\lambda\mu$.

3.1. Definición

A las variables $x, y, z, \ldots$ las llamamos λ-variables ya que se asocian a la abstracción λ. Las variables $\alpha, \beta, \gamma, \ldots$ se ligan a la abstracción μ, por lo que éstas son μ-variables. En el cálculo $\lambda\mu$ se emplea una estrategia de *llamada-por-nombre* y un estilo *a la Church*. Sus términos son construídos a partir de la siguiente gramática,

$$M ::= x \mid MM \mid \lambda x : \mathsf{T}.\, M \mid [\alpha]M \mid \mu\alpha : \neg\mathsf{T}.\, M$$

Notemos que:

- Las μ-variables no son términos, a diferencia de las λ-variables.
- El símbolo μ es un operador de ligado, de tal suerte que en el término $\mu\alpha : \neg\mathsf{T}.\, M$ están ligadas todas las presencias libres de α que se encuentren en M.
- Se cuenta con dos expresiones nuevas, el término $\mu\alpha : \neg\mathsf{T}.\, M$ se conoce como *abstracción* μ y la expresión $[\alpha]M$ como *aplicación canalizada*.

Para asignar tipos a las expresiones $\lambda\mu$ recurrimos a las siguientes reglas de tipificación[17],

$$\frac{}{\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}}(RVar)$$

$$\frac{\Gamma, x : \mathsf{T} \vdash M : \mathsf{S}}{\Gamma \vdash \lambda x : \mathsf{T}.\, M : \mathsf{T} \to \mathsf{S}}(RFun) \qquad \frac{\Gamma \vdash M : \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash N : \mathsf{T}}{\Gamma \vdash MN : \mathsf{S}}(RApp)$$

$$\frac{\Gamma, \alpha : \neg\mathsf{T} \vdash M : \bot}{\Gamma \vdash \mu\alpha : \neg\mathsf{T}.\, M : \mathsf{T}}(\mu RFun) \qquad \frac{\Gamma, \alpha : \neg\mathsf{T} \vdash M : \mathsf{T}}{\Gamma, \alpha : \neg\mathsf{T} \vdash [\alpha]M : \bot}(\mu RApp)$$

Reglas de tipificación para el cálculo $\lambda\mu$

Una observación importante es que las μ-variables siempre tienen tipo negado, esto nos recuerda el tipo $\mathsf{T} \to \bot$ que Griffin asignó a los contextos de evaluación E del cálculo $\lambda\mathcal{C}$ y es que, efectivamente, las μ-variables se asocian con los contextos de evaluación debido a que ambos representan continuaciones.

Veamos un ejemplo de inferencia de tipos. Si $\Delta = \{x : (\mathsf{T} \to \mathsf{S}) \to \mathsf{T}, \alpha : \neg\mathsf{T}\}$ y ω representa la expresión $\mu\alpha : \neg\mathsf{T}.[\alpha](x(\lambda z : \mathsf{T}.\mu\beta : \neg\mathsf{S}.[\alpha]z))$ entonces se tiene la siguiente derivación, donde el tipo final representa la *ley de Peirce* $((\mathsf{T} \to \mathsf{S}) \to \mathsf{T}) \to \mathsf{T}$,

$$\dfrac{\dfrac{}{\Delta \vdash x : (\mathsf{T} \to \mathsf{S}) \to \mathsf{T}}(RVar) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Delta, z : \mathsf{T}, \beta : \neg\mathsf{S} \vdash z : \mathsf{T}}(RVar)}{\Delta, z : \mathsf{T}, \beta : \neg\mathsf{S} \vdash [\alpha]z : \bot}(\mu RApp)}{\Delta, z : \mathsf{T} \vdash \mu\beta : \neg\mathsf{S}.[\alpha]z : \mathsf{S}}(\mu RFun)}{\Delta \vdash \lambda z : \mathsf{T}.\mu\beta : \neg\mathsf{S}.[\alpha]z : \mathsf{T} \to \mathsf{S}}(RFun)}{\dfrac{\Delta \vdash x(\lambda z : \mathsf{T}.\mu\beta : \neg\mathsf{S}.[\alpha]z) : \mathsf{T}}{\dfrac{\Delta \vdash [\alpha](x(\lambda z : \mathsf{T}.\mu\beta : \neg\mathsf{S}.[\alpha]z)) : \bot}{\dfrac{x : (\mathsf{T} \to \mathsf{S}) \to \mathsf{T} \vdash \omega : \mathsf{T}}{\vdash \lambda x : (\mathsf{T} \to \mathsf{S}) \to \mathsf{T}.\omega : ((\mathsf{T} \to \mathsf{S}) \to \mathsf{T}) \to \mathsf{T}}(RFun)}(\mu RFun)}(\mu RApp)}(RApp)}$$

[17]En este capítulo el tipo nulo o void será denotado con $\bot$.

El significado computacional de las nuevas expresiones es el siguiente, para el término $\mu\alpha : \neg\mathsf{T}.\cdots[\alpha]M\cdots$ la μ-variable α es un canal por el cual los valores son transmitidos y el subtérmino $[\alpha]M$ representa el lanzamiento de M a través del canal $[\alpha]$ para ser recibido por $\mu\alpha : \neg\mathsf{T}.\cdots$. Por consiguiente, es de esperarse que el valor de dicha abstracción μ sea M. A estas expresiones se les conoce como *saltos* u *operadores de control* ya que M brinca todos los demás subtérminos de la expresión y resulta ser el valor final.

A continuación mostramos dos lemas que serán de utilidad más adelante.

Lema 3.1. *Se satisface que:*

$$\Gamma, \alpha : \neg\mathsf{T} \vdash \lambda x : \mathsf{T}.[\alpha]x : \neg\mathsf{T}$$

Demostración.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \alpha : \neg\mathsf{T}, x : \mathsf{T} \vdash x : \mathsf{T}}\,(RVar)}{\Gamma, \alpha : \neg\mathsf{T}, x : \mathsf{T} \vdash [\alpha]x : \bot}\,(\mu RApp)}{\Gamma, \alpha : \neg\mathsf{T} \vdash \lambda x : \mathsf{T}.[\alpha]x : \neg\mathsf{T}}\,(\mu RFun)$$

□

Lema 3.2. *Si* $\Gamma, x : \mathsf{T} \vdash M : \mathsf{S}$ *y* $\Gamma \vdash N : \mathsf{T}$ *entonces* $\Gamma \vdash M[x := N] : \mathsf{S}$.

Demostración. La demostración es por inducción sobre los términos del cálculo $\lambda\mu$. Es practicamente la misma que se dio para la proposición 1.5 del capítulo 1 en la página 17, sólo se añaden los siguientes casos.

- Para la abstracción μ suponemos $\Gamma, x : \mathsf{T} \vdash \mu\alpha : \neg\mathsf{S}.M : \mathsf{S}$ y $\Gamma \vdash N : \mathsf{T}$. Además, observamos que la sustitución $(\mu\alpha : \neg\mathsf{S}.M)[x := N]$ consta de reemplazar las presencias libres de x por N en M. Dicha sustitución se define como $\mu\alpha : \neg\mathsf{S}.(M[x := N])$. Por hipótesis de inducción se tiene $\Gamma, \alpha : \neg\mathsf{S} \vdash M[x := N] : \bot$, entonces de acuerdo a la regla $(\mu RFun)$ la tipificación correcta es:

  $$\Gamma \vdash \mu\alpha : \neg\mathsf{S}.(M[x := N]) : \mathsf{S}$$

  así,

  $$\Gamma \vdash (\mu\alpha : \neg\mathsf{S}.M)[x := N] : \mathsf{S}$$

- Para la aplicación μ suponemos $\Gamma, \alpha : \neg\mathsf{S}, x : \mathsf{T} \vdash [\alpha]M : \bot$ y $\Gamma, \alpha : \neg\mathsf{S} \vdash N : \mathsf{T}$. La sustitución $([\alpha]M)[x := N]$ no se ocupa de α, así que se expresa como $[\alpha](M[x := N])$. Aplicando la hipótesis de inducción a M llegamos a $\Gamma, \alpha : \neg\mathsf{S} \vdash M[x := N] : \mathsf{S}$, y considerando la regla ($\mu RApp$) se tiene el tipo,

$$\Gamma, \alpha : \neg\mathsf{S} \vdash [\alpha](M[x := N]) : \bot$$

de aquí que,

$$\Gamma, \alpha : \neg\mathsf{S} \vdash ([\alpha]M)[x := N] : \bot$$

□

Para dar las reducciones del cálculo $\lambda\mu$ necesitamos primero de ciertas ideas preliminares,

- Los contextos de evaluación o contextos aplicativos $\lambda\mu$ que se dieron en el cálculo $\lambda\mathcal{C}$ de Felleisen, ahora se denotan por $\mathcal{E}$ y se forman a partir de la gramática,

$$\mathcal{E} ::= [\,] \mid \mathcal{E}M \mid [\alpha]\mathcal{E}$$

Como vimos, las reglas para llenar el hueco en $\mathcal{E}$ con M son,

$$\begin{aligned}
[\,][M] &= M \\
(\mathcal{E}N)[M] &= (\mathcal{E}[M])N \\
([\alpha]\mathcal{E})[M] &= [\alpha](\mathcal{E}[M])
\end{aligned}$$

- El conjunto de variables libres de $\mathcal{E}$, denotado $FV(\mathcal{E})$, se define como,

$$\begin{aligned}
FV([\,]) &= \varnothing \\
FV(\mathcal{E}M) &= FV(\mathcal{E}) \cup FV(M) \\
FV([\alpha]\mathcal{E}) &= \{\alpha\} \cup FV(\mathcal{E})
\end{aligned}$$

- La operación de sustitución $M[\alpha := \mathcal{E}]$ queda definida a partir de las siguientes reglas con $y, \beta \notin FV(\mathcal{E})$ y $\beta \neq \alpha$,

$$\begin{aligned}
x[\alpha := \mathcal{E}] &= x \\
(\lambda y : \mathsf{T}.M)[\alpha := \mathcal{E}] &= \lambda y : \mathsf{T}.(M[\alpha := \mathcal{E}]) \\
(M_1 M_2)[\alpha := \mathcal{E}] &= M_1[\alpha := \mathcal{E}]M_2[\alpha := \mathcal{E}] \\
(\mu\beta : \neg\mathsf{T}.M)[\alpha := \mathcal{E}] &= \mu\beta : \neg\mathsf{T}.(M[\alpha := \mathcal{E}]) \\
([\alpha]M)[\alpha := \mathcal{E}] &= \mathcal{E}[M[\alpha := \mathcal{E}]] \\
([\beta]M)[\alpha := \mathcal{E}] &= [\beta](M[\alpha := \mathcal{E}])
\end{aligned}$$

Ahora estamos en posibilidad de dar las reglas de reducción para el cálculo $\lambda\mu$, que están constituidas por las reglas $\rightarrow_\beta$, $\rightarrow_{\eta_\mu}$, $\rightarrow_{\beta_\mu}$ y $\rightarrow_\varsigma$, cuya unión se denota $\rightarrow_\mu$,

$$\begin{aligned}
(\lambda x : \mathsf{T}.M)N &\rightarrow_\beta M[x := N] \\
\mu\alpha : \neg\mathsf{T}.[\alpha]M &\rightarrow_{\eta_\mu} M \text{ si } \alpha \notin FV(M) \\
[\beta](\mu\alpha : \neg\mathsf{T}.M) &\rightarrow_{\beta_\mu} M[\alpha := [\beta][\,]] \\
(\mu\alpha : \neg(\mathsf{T} \rightarrow \mathsf{S}).M)N &\rightarrow_\varsigma \mu\beta : \neg\mathsf{S}.M[\alpha := [\beta]([\,]N)] \\
\text{para } \rightarrow_\varsigma \text{ se requiere } \alpha \neq \beta &\text{ y } \beta \notin FV(MN)
\end{aligned}$$

Reglas de reducción $\rightarrow \mu$

La regla $\rightarrow_\varsigma$ se conoce como *reducción estructural* y $\rightarrow_{\beta_\mu}$ como *renombramiento.*

Discutamos cada una de las reglas por separado

a. La regla $\rightarrow_\beta$ expresa simplemente la conocida reducción β.

b. La regla $\rightarrow_{\eta_\mu}$ transmite el resultado a lo largo del canal. Teóricamente nos dice que si una hipótesis $\neg\mathsf{T}$, esto es $[\alpha]$, se aplica a una prueba de T, digamos M, que no utiliza la hipótesis $[\alpha]$ entonces la prueba para T, $\mu\alpha : \neg\mathsf{T}.[\alpha]M$, puede reemplazarse por M.

c. En la regla $\rightarrow_{\beta_\mu}$ se optimiza el salto, ya que en el término $[\beta](\mu\alpha : \neg\mathsf{T}.M)$ puede suceder que en la M alguna expresión M' sea transmitida a α, recibida por $\mu\alpha$ y retransmitida a β. Es mejor si transmitimos M' directamente a β en los lugares donde se transmitiría a α, esto se logra si realizamos la sustitución $M[\alpha := [\beta][\,]]$.

d. En la regla $\rightarrow_\varsigma$ se indica que en lugar de aplicar el término N de tipo T a la abstracción $\mu\alpha : \neg(\mathsf{T} \rightarrow \mathsf{S}).M$ es más conveniente aplicar N al término que está siendo transmitido en M, así obtenemos $\mu\beta : \neg\mathsf{S}.M[\alpha := [\beta]([\,]N)]$. Desde un punto de vista teórico, al asumir $\neg(\mathsf{T} \rightarrow \mathsf{S})$ y concluir $\perp$ se consigue una prueba por contradicción de $\mathsf{T} \rightarrow \mathsf{S}$, esto es $\mu\alpha : \neg(\mathsf{T} \rightarrow \mathsf{S}).M$, luego a esta expresión se le aplica una prueba N de T para obtener una prueba de S, la cual es $(\mu\alpha : \neg(\mathsf{T} \rightarrow \mathsf{S}).M)N$. Comúnmente en una prueba por contradicción de $\mathsf{T} \rightarrow \mathsf{S}$ asumimos T y $\neg\mathsf{S}$ y concluímos $\perp$, la prueba buscada es $\lambda x : \mathsf{T}.\mu\beta : \neg\mathsf{S}.(M[\alpha := [\beta]([\,]x)])$. Como N es una prueba de T, al término anterior le pasamos la N y obtenemos la prueba $\mu\beta : \neg\mathsf{S}.(M[\alpha := [\beta]([\,]N)])$ para S.

3.2. Isomorfismo de Curry-Howard

La estrecha similitud que existe entre la tipificación de los términos del cálculo $\lambda\mu$ y el cálculo proposicional clásico $NK(\to, \bot)$ se exhibe en este apartado. De capítulos pasados retomamos las transformaciones $(-)^*$ y $(-)^\circ$, de fórmulas a tipos y de tipos a fórmulas, respectivamente. También consideramos estas transformaciones extendidas a contextos lógicos y contextos de tipos como las hemos manejado anteriormente.

Teorema 3.1. (Isomorfismo de Curry-Howard) *Sean φ una fórmula clásica, T un tipo correspondiente al cálculo $\lambda\mu$, Γ un contexto lógico y Δ un contexto de tipos. Luego,*

I. *Si $\Delta \vdash M : \mathsf{T}$ en el cálculo $\lambda\mu$ entonces $\Delta^\circ \vdash_c \mathsf{T}^\circ$ en $NK(\to, \bot)$*

II. *Si $\Gamma \vdash_c \varphi$ se cumple en $NK(\to, \bot)$ entonces existe un término M tal que $\Gamma^* \vdash M : \varphi^*$ en $\lambda\mu$*

Demostración. La demostración del teorema que se da a continuación es una versión ligeramente modificada de la prueba hecha por Sørensen y Urzyczyn en [31].

Para la demostración de I procedemos por inducción sobre la inferencia de tipos $\Delta \vdash M : \mathsf{T}$ del cálculo $\lambda\mu$. La base de inducción y la mayoría de los casos para el paso inductivo han sido probados en el teorema 1.1 de la sección 1.3.1.

- Si la inferencia de tipos termina usando la regla $(\mu RFun)$ suponemos por hipótesis de inducción que se tiene $\Delta^\circ, \neg\mathsf{T}^\circ \vdash_c \bot$, luego usando la regla $(\neg\neg E)$, que forma parte de $NK(\to, \bot)$, llegamos a que $\Delta^\circ \vdash_c \mathsf{T}^\circ$, que es lo que queríamos probar.

- Para la regla $(\mu RApp)$ no se tiene una en $NK(\to, \bot)$ a la que se le relacione directamente. No obstante, si en el último paso de la inferencia se utiliza $(\mu RApp)$ sabemos por hipótesis de inducción que $\Delta^\circ, \neg\mathsf{T}^\circ \vdash_c \mathsf{T}^\circ$, es decir $\Delta^\circ, \mathsf{T}^\circ \to \bot^\circ \vdash_c \mathsf{T}^\circ$. También es cierto que $\Delta^\circ, \mathsf{T}^\circ \to \bot^\circ \vdash_c \mathsf{T}^\circ \to \bot^\circ$ puesto que es un axioma. Ahora, usando estos últimos dos resultados y la regla $(\to E)$ de $NK(\to, \bot)$ concluimos $\Delta^\circ, \neg\mathsf{T}^\circ \vdash_c \bot^\circ$, así la proposición es cierta para $(\mu RApp)$.

Para el inciso II llevamos a cabo una prueba por inducción sobre la inferencia lógica $\Gamma \vdash_c \varphi$. La base de inducción y buena parte de los casos para el

paso inductivo se encuentran en el teorema 1.1 de la sección 1.3.1 del capítulo 1. Cuando la inferencia termina con la regla $(\neg\neg E)$,

$$\frac{\Gamma, \varphi \to \bot \vdash_c \bot}{\Gamma \vdash_c \varphi}$$

analizamos dos posibilidades,

Caso 1. Si $\neg\varphi \in \Gamma$.

Por hipótesis de inducción existe un término M' tal que $\Gamma^* \vdash M' : \bot^*$, como el agregar a Γ^* una μ-variable no afecta la inferencia actual se tiene que $\Gamma^*, \alpha : \neg\varphi^* \vdash M' : \bot^*$. Por lo tanto, gracias a la regla $(\mu RFun)$, si tomamos $M = \mu\alpha : \neg\varphi^*.M'$ entonces $\Gamma^* \vdash M : \varphi^*$.

Caso 2. Si $\neg\varphi \notin \Gamma$.

Por hipótesis de inducción existe un término M' tal que $\Gamma^*, x : \neg\varphi^* \vdash M' : \bot^*$. Al agregar la declaración $\alpha : \neg\varphi^*$ obtenemos $\Gamma^*, \alpha : \neg\varphi^*, x : \neg\varphi^* \vdash M' : \bot^*$. Por el lema 3.1 sabemos que $\Gamma^*, \alpha : \neg\varphi^* \vdash \lambda y : \varphi^*.[\alpha]y : \neg\varphi^*$ y por el lema 3.2 tenemos $\Gamma^*, \alpha : \neg\varphi^* \vdash M'[x := \lambda y : \varphi^*.[\alpha]y\,] : \bot^*$. Por consiguiente, ocupando la regla $(\mu RFun)$ y escogiendo $M = \mu\alpha : \neg\varphi^*.M'[x := \lambda y : \varphi^*.[\alpha]y\,]$ se tiene $\Gamma^* \vdash M : \varphi^*$, lo que muestra que la proposición es válida para $(\neg\neg E)$.

□

3.3. Propiedades de las reducciones-μ

Esta sección se enfoca en probar tres propiedades importantes para el cálculo $\lambda\mu$, las cuales enlazan el sistema de tipos con las reducciones μ.

3.3.1. Reducción del sujeto

Para lo cual necesitaremos de los siguientes lemas que se mencionan en [31] pero que nosotros demostramos paso a paso.

Lema 3.3. *Si* $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \mathsf{S}$, $z \notin FV(\mathcal{E})$ *y* $\Gamma \vdash N : \mathsf{T}$ *entonces* $\Gamma \vdash \mathcal{E}[N] : \mathsf{S}$.

Demostración. Por inducción sobre el contexto $\mathcal{E}$.

- *Base de inducción* $\mathcal{E} = [\,]$. Por hipótesis se tiene $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \mathsf{S}$ y $\Gamma \vdash N : \mathsf{T}$, como $\mathcal{E}[z] = z$ se sigue que $\Gamma, z : \mathsf{T} \vdash z : \mathsf{S}$ y $\mathsf{T} = \mathsf{S}$. Luego, $\mathcal{E}[N] = N$ así que $\Gamma \vdash \mathcal{E}[N] : \mathsf{S}$.

- *Hipótesis de inducción.* La proposición es cierta para $\mathcal{E}'$. Es decir que dado $\Gamma, z : \mathsf{T} \vdash \mathcal{E}'[z] : \mathsf{S}$, $z \notin FV(\mathcal{E}')$ y $\Gamma \vdash N : \mathsf{T}$ entonces se cumple $\Gamma \vdash \mathcal{E}'[N] : \mathsf{S}$.

- *Paso inductivo.*

  - $\mathcal{E} = \mathcal{E}'M$. Suponemos que $\Gamma, z : \mathsf{T} \vdash (\mathcal{E}'M)[z] : \mathsf{S}$, i.e. $\Gamma, z : \mathsf{T} \vdash \mathcal{E}'[z]M : \mathsf{S}$ con $z \notin FV(\mathcal{E})$. De modo que la regla $(RApp)$ nos indica que existe un tipo S' tal que,

    $$\Gamma, z : \mathsf{T} \vdash \mathcal{E}'[z] : \mathsf{S}' \to \mathsf{S} \qquad \Gamma, z : \mathsf{T} \vdash M : \mathsf{S}'$$

    como $z \notin FV(\mathcal{E})$ en particular es cierto que $z \notin FV(M)$, entonces se tiene $\Gamma \vdash M : \mathsf{S}'$. Usando este hecho, la hipótesis de inducción $\Gamma \vdash \mathcal{E}'[N] : \mathsf{S}' \to \mathsf{S}$, la regla $(RApp)$ y que $\mathcal{E}[N] = (\mathcal{E}'[N])M$, llegamos a que $\Gamma \vdash \mathcal{E}[N] : \mathsf{S}$.

  - $\mathcal{E} = [\alpha]\mathcal{E}'$. Suponemos $\Gamma, \alpha : \neg\mathsf{S}', z : \mathsf{T} \vdash ([\alpha]\mathcal{E}')[z] : \bot$, es decir, $\Gamma, \alpha : \neg\mathsf{S}', z : \mathsf{T} \vdash [\alpha](\mathcal{E}'[z]) : \bot$, de modo que por la regla de tipificación $(\mu RApp)$ afirmamos que,

    $$\Gamma, \alpha : \neg\mathsf{S}', z : \mathsf{T} \vdash \mathcal{E}'[z] : \mathsf{S}'$$

    Por hipótesis de inducción obtenemos $\Gamma, \alpha : \neg\mathsf{S}' \vdash \mathcal{E}'[N] : \mathsf{S}'$. Por $(\mu RApp)$ y puesto que $\mathcal{E}[N] = [\alpha](\mathcal{E}'[N])$ se tiene $\Gamma, \alpha : \neg\mathsf{S}' \vdash \mathcal{E}[N] : \bot$.

□

Lema 3.4. *Si* $\Gamma, \alpha : \neg\mathsf{T} \vdash M : \mathsf{S}$ *y* $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \bot$ *entonces* $\Gamma \vdash M[\alpha := \mathcal{E}] : \mathsf{S}$ *dado que* $z \notin FV(\mathcal{E})$

Demostración. Inducción sobre M.

- *Base de inducción.* Consideremos el término x. Si $\Gamma, \alpha : \neg\mathsf{T} \vdash x : \mathsf{S}$ y $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \bot$ entonces $\Gamma \vdash x[\alpha := \mathcal{E}] : \mathsf{S}$, ya que $x[\alpha := \mathcal{E}] = x$ y $\Gamma \vdash x : \mathsf{S}$ por la regla $(RVar)$.

- *Hipótesis de inducción.* El lema es cierto para M', es decir, si $\Gamma, \alpha : \neg\mathsf{T} \vdash M' : \mathsf{S}$ y $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \bot$ entonces $\Gamma \vdash M'[\alpha := \mathcal{E}] : \mathsf{S}$ dado que $z \notin FV(\mathcal{E})$.

- *Paso inductivo.* Se demuestra para cada uno de los casos en el que el término M es formado usando las reglas de la gramática que define al cálculo $\lambda\mu$. Como ejemplo tomemos $M = [\alpha]M'$. Suponemos que $\Gamma, z : \mathsf{T} \vdash \mathcal{E}[z] : \bot\,(^*)$ y $\Gamma, \alpha : \neg\mathsf{T} \vdash [\alpha]M' : \bot$, por ($\mu RApp$) es cierto que $\Gamma, \alpha : \neg\mathsf{T} \vdash M' : \mathsf{T}$, usando la hipótesis de inducción llegamos a $\Gamma \vdash M'[\alpha := \mathcal{E}] : \mathsf{T}\,(^{**})$. Finalmente, por $(^*)$, $(^{**})$, el lema 3.3 y la sustitución $([\alpha]M')[\alpha := \mathcal{E}] = \mathcal{E}[M'[\alpha := \mathcal{E}]$ se cumple que $\Gamma \vdash ([\alpha]M')[\alpha := \mathcal{E}] : \bot$.

□

Teorema 3.2. (Reducción del sujeto) *Si* $M \to_\mu N$ *y* $\Gamma \vdash M : \mathsf{T}$ *entonces* $\Gamma \vdash N : \mathsf{T}$

Demostración. Inducción sobre $M \to_\mu N$ usando los lemas 3.2 y 3.4.

Como ejemplo tomaremos la reducción $\to_{\beta_\mu}$,

$$[\beta](\mu\alpha : \neg\mathsf{T}.M) \to_{\beta_\mu} M[\alpha := [\beta][]]$$

Es fácil ver que $\Gamma, \beta : \neg\mathsf{T} \vdash [\beta](\mu\alpha : \neg\mathsf{T}.M) : \bot$, de donde por la regla ($\mu RApp$) se obtiene $\Gamma, \beta : \neg\mathsf{T} \vdash \mu\alpha : \neg\mathsf{T}.M : \mathsf{T}$. Luego, por la regla ($\mu RFun$) debe cumplirse,

$$\Gamma, \beta : \neg\mathsf{T}, \alpha : \neg\mathsf{T} \vdash M : \bot$$

Además el lector puede cerciorarse de que,

$$\Gamma, \beta : \neg\mathsf{T}, z : \mathsf{T} \vdash ([\beta][])[z] : \bot$$

Por las dos últimas tipificaciones y el lema 3.4 tenemos $\Gamma, \beta : \neg\mathsf{T} \vdash M[\alpha := [\beta][]] : \bot$, con lo que se cumple el teorema. □

3.3.2. Normalización fuerte

La idea es utilizar el hecho de que en $\lambda^\to$ se satisface la propiedad de normalización fuerte (véase proposición 1.7 capítulo 1), por lo que se convertirán términos $\lambda\mu$ en términos $\lambda^\to$ respetando las reducciones de términos. El estilo que se ha ocupado para el cálculo $\lambda\mu$ es *a la Church*, para simplificar la conversión se obtendrán términos del $\lambda^\to$ *a la Curry*. Suponemos también que $\bot$ es un tipo simple.

Definición 3.1. *Asociamos a cada μ-variable α de tipo $\neg(\mathsf{U}_1 \to \ldots \to \mathsf{U}_n \to \mathsf{V})$ un vector de variables nuevas $\vec{y}^{\,\alpha} = y_1^\alpha, \ldots y_n^\alpha$, una por cada argumento U_i de α.*

Por consiguiente, la conversión de $\lambda\mu$-términos a $\lambda^\to$-términos obedece las siguientes reglas,

$$\begin{aligned} \underline{x} &= x \\ \underline{\lambda x : \mathsf{T}.M} &= \lambda x.\underline{M} \\ \underline{MN} &= \underline{M}\,\underline{N} \\ \underline{\mu\alpha : \neg\mathsf{T}.M} &= \lambda\vec{y}^{\,\alpha}\underline{M} \\ \underline{[\alpha]M} &= \underline{M}\vec{y}^{\,\alpha} \end{aligned}$$

Adicionalmente, si T es un tipo se define $\underline{\mathsf{T}}$ reemplazando todas las variables de tipo que aparezcan en T por $\bot$.

Lema 3.5. *Si M es un término $\lambda\mu$ con tipo T entonces se satisface que $\Gamma \vdash \underline{M} : \underline{\mathsf{T}}$ en $\lambda^\to$, donde los elementos de Γ son:*

- *$x : \underline{\mathsf{S}}$, por cada λ-variable libre x en M de tipo S.*
- *$y_1^\alpha : \underline{\mathsf{S}}_1, \ldots y_n^\alpha : \underline{\mathsf{S}}_n$, por cada μ-variable libre α de tipo $\neg(\mathsf{S}_1 \to \cdots \to \mathsf{S}_n \to \mathsf{U})$.*

Demostración. Inducción sobre la derivación de tipos $\Gamma \vdash M : \mathsf{T}$ en el cálculo $\lambda\mu$. □

Lema 3.6. *Se cumple lo siguiente,*

I. $\underline{P[x := Q]} = \underline{P}[x := \underline{Q}]$

II. $\underline{P[\alpha := [\beta][\,]]} = \underline{P}[\vec{y}^{\,\alpha} := \vec{y}^{\,\beta}]$

III. $\underline{P[\alpha := [\beta]([\,]Q)]} = \underline{P}[\vec{y}^{\,\alpha} := \underline{Q}, \vec{y}^{\,\beta}]$ *Obsérvese que esta sustitución proviene de la regla $\to_\varsigma$, luego el número de argumentos para α es uno más que el número de argumentos para β, por lo que la sustitución $\underline{P}[\vec{y}^{\,\alpha} := \underline{Q}, \vec{y}^{\,\beta}]$ está bien definida.*

Demostración. Por inducción sobre los términos P del cálculo $\lambda\mu$.

Tomemos como ejemplo el inciso III y veamos qué pasa en el caso $P = [\alpha]M$.

$$\begin{aligned}
\underline{([\alpha]M)[\alpha := [\beta]([\,]Q)]} &= \underline{([\beta]([\,]Q))\overbrace{[M[\alpha := [\beta]([\,]Q)]}^{M'}]} \\
&= \underline{[\beta](M'Q)} \\
&= \underline{(M'Q)\,\vec{y}^{\,\beta}} \\
&= (\underline{M'}\underline{Q})\,\vec{y}^{\,\beta} \\
&= (\underline{M}[\vec{y}^{\,\alpha} := \underline{Q}, \vec{y}^{\,\beta}]\underline{Q})\,\vec{y}^{\,\beta} \;\text{ por } HI \text{ en } M' \\
&= (\underline{M\,\vec{y}^{\,\alpha}})[\vec{y}^{\,\alpha} := \underline{Q}, \vec{y}^{\,\beta}] \\
&= \underline{[\alpha]M}[\vec{y}^{\,\alpha} := \underline{Q}, \vec{y}^{\,\beta}]
\end{aligned}$$

□

El comportamiento de la reducción μ cuando sus μ-términos son transformados a $\lambda^{\to}$-términos se estudia a contiuación.

Proposición 3.1. *Si $M \to_\mu N$ en $\lambda\mu$, entonces $\underline{M} \to^*_{\beta\eta} \underline{N}$, donde $\to_{\beta\eta} \stackrel{def}{=} \to_\beta \cup \to_\eta$. Además se cumple que si $M \to_{\beta\varsigma} N$ entonces $\underline{M} \to_\beta \underline{N}$.*

Demostración. Inducción sobre la reducción $M \to_\mu N$.

Como ilustración consideremos el caso:

$$M = (\mu\alpha : \neg(\mathsf{T} \to \mathsf{S}).P)Q \to_\varsigma \mu\beta : \neg\mathsf{S}.P[\alpha := [\beta]([\,]Q)] = N$$

Si $\vec{y}^{\,\alpha} = x, \vec{y}$ donde $\vec{y}$ es el vector de variables que se asocia al tipo S, entonces, usando el lema anterior tenemos,

$$\begin{aligned}
\underline{M} &= (\lambda x\vec{y}.\underline{P})\underline{Q} \to_\beta \lambda\vec{y}.\underline{P}[x := \underline{Q}] \\
&= \lambda\vec{z}.\underline{P}[x, \vec{y} := \underline{Q}, \vec{z}] \quad \text{cambio del vector } \vec{y} \text{ por el vector } \vec{z} \\
&= \lambda\vec{z}.\underline{P}[\vec{y}^{\,\alpha} := \underline{Q}, \vec{z}] \\
&= \lambda\vec{z}.\underline{P[\alpha := [\beta]([\,]Q)]} \quad \text{por el lema 3.6, III} \\
&= \underline{\mu\beta : \neg\mathsf{S}.P[\alpha := [\beta]([\,]Q)]} = \underline{N}
\end{aligned}$$

□

Ahora estamos en posibilidades de probar la propiedad de normalización fuerte.

Teorema 3.3. (Normalización fuerte) *Si* $\vdash M : \mathsf{T}$ *en* $\lambda\mu$ *entonces no existe secuencia infinita de reducciones* $M \to_\mu \ldots$

Demostración. Para llegar a una contradicción supongamos que $\vdash M : \mathsf{T}$ en $\lambda\mu$ y que existen una infinidad de reducciones μ para M,

$$M \to_\mu M' \to_\mu M'' \to_\mu \ldots \qquad (*)$$

Por la proposición 3.1 sabemos que,

$$\underline{M} \to^*_{\beta\eta} \underline{M'} \to^*_{\beta\eta} \underline{M''} \to^*_{\beta\eta} \ldots \qquad (**)$$

Como las reducciones β_μ y η_μ decrementan el número de presencias μ en el término, entonces no puede haber una infinidad de dichas reducciones en (*). Luego, debe de haber una infinidad de reducciones $\beta\varsigma$ en (*), por la proposición 3.1 existe una infinidad de reducciones $\to_\beta$ en (**). Por el lema 3.5, $\underline{M}$ es un término del cálculo $\lambda^\to$ para el cual existe una infinidad de reducciones $\to_\beta$, esto contradice la propiedad de normalización fuerte para $\lambda^\to$ que se vio en la proposición 1.7 capítulo 1. □

3.3.3. Confluencia

La última propiedad que estudiaremos es confluencia y se enuncia como sigue.

Teorema 3.4. (Confluencia o Propiedad de Church-Rosser) *Si* $M_1 \to^* M_2$ *y* $M_1 \to^* M_3$ *con* $\vdash M : \mathsf{T}$ *en* $\lambda\mu$*, entonces existe término* N *tal que* $M_2 \to^* N$ *y* $M_3 \to^* N$*.*

Demostración. La demostración no es trivial y se omite en esta tesis; sin embargo, puede encontrarse en [31]. □

3.4. Operadores de control

En esta sección expresaremos en términos del cálculo $\lambda\mu$ algunos operadores de control con los que ya estamos familiarizados.

3.4.1. catch-throw

En la expresión

$$\mathbf{catch}\,\alpha\,\mathbf{in}\,M \qquad (+)$$

se evalúa en primer lugar la M. Si ésta produce algún valor entonces tal valor es el resultado de la expresión $(+)$. En cuyo caso tenemos un regreso normal. Si por el contrario, en M encontramos el término,

$$\mathbf{throw}\,N\,\mathbf{to}\,\alpha$$

la evaluación de M término abruptamente. Por lo que el resultado de la expresión $(+)$ es el resultado de evaluar N. Esto se conoce como regreso por excepción.

La definición en el cálculo $\lambda\mu$ de **catch** y **throw** es,

$$\begin{aligned}\mathbf{catch}\,\alpha\,\mathbf{in}\,M &= \mu\alpha : \neg\mathsf{T}.[\alpha]M \\ \mathbf{throw}\,M\,\mathbf{to}\,\alpha &= \mu\beta : \neg\mathsf{U}.[\alpha]M \quad \beta \notin FV([\alpha]M)\end{aligned}$$

donde el tipo de M es T y el tipo U depende del contexto. Con esta definición se cumple que $\mathbf{catch}\,\alpha\,\mathbf{in}\,\mathcal{E}[\mathbf{throw}\,N\,\mathbf{to}\,\alpha] \to_\mu^* N$ suponiendo que N y $\mathcal{E}$ no tienen variables libres.

Primeramente veamos un caso especial de $\to_\varsigma$, en el que $\beta \notin FV(P)$; por consiguiente, $(\mu\beta : \neg(\mathsf{T} \to \mathsf{S}).P)Q \to_\varsigma \mu\beta' : \neg\mathsf{S}.P$. Observamos también que si $\mathcal{E}$ no tiene variables libres entonces es un contexto de la forma $[\,]N_1 \ldots N_n$, donde $FV(N_1) = \ldots = FV(N_n) = \varnothing$; luego, usando el caso especial de $\to_\varsigma$ tenemos $\mathcal{E}[\mu\beta : \neg\mathsf{U}.P] \to_\mu^* \mu\gamma : \neg\mathsf{U}'.P$ para tipos U y U' acordes al contexto $\mathcal{E}$.

Entonces, considerando que $\mathcal{E}$ y N no tienen variables libres, tenemos,

$$\begin{aligned}\mathbf{catch}\,\alpha\,\mathbf{in}\,\mathcal{E}[\mathbf{throw}\,N\,\mathbf{to}\,\alpha] &= \mu\alpha : \neg\mathsf{T}.[\alpha]\mathcal{E}[\mu\beta : \neg\mathsf{U}.[\alpha]N] \\ &\to_\mu^* \mu\alpha : \neg\mathsf{T}.[\alpha](\mu\gamma : \neg\mathsf{U}'.[\alpha]N) \\ &\to_{\beta_\mu} \mu\alpha : \neg\mathsf{T}.[\alpha]N \\ &\to_{\eta_\mu} N\end{aligned}$$

3.4.2. Abort

Recordemos que el cálculo $\lambda\mathcal{C}$ de Felleisen cuenta con el operador $\mathcal{A}$, el cual al aplicarle la expresión M aborta el cómputo actual, ignora el contexto

presente, retoma el contexto de M y comienza a evaluar esta expresión, por lo que la reducción correspondiente es,

$$E[\mathcal{A}M] \mapsto_{\mathcal{C}} M \qquad (\diamond)$$

Siguiendo la definición para el operador *abort* dada en [13] y [32] debemos considerar una versión ligeramente diferente del cálculo $\lambda\mu$, donde el estilo que se sigue es *a la Curry* y los términos son *restringidos*, como lo muestra la siguiente gramática,

$$M ::= x \mid MM \mid \lambda x.M \mid \mu\alpha.[\beta]M$$

Además la regla $\rightarrow_{\beta_\mu}$ queda como,

$$\mu\alpha'.[\beta](\mu\alpha.M) \rightarrow_{\beta_\mu} \mu\alpha'.M[\alpha := [\beta][\,]]$$

En [14] Herbelin menciona que con esta definición los términos del cálculo $\lambda\mu$ no se corresponden con la lógica clásica, pero si con la llamada *lógica clásica minimal*, en la que la ley de Peirce es válida pero no la regla *EFQ*. Desde una perspectiva computacional esto nos dice que en el cálculo $\lambda\mu$ *restringido* no se permite abortar cómputos, a pesar de que cuenta con variables que representan directamente continuaciones, así que para lograr expresar $\mathcal{A}$ debemos definir una continuación especial con la que podamos manejar el contexto de la manera en la que lo hace la regla ($\diamond$). Esta continuación será una constante de *alto nivel* que se denota como $\mathbf{tp}$[18], así se podrá diferenciar entre abortar un cómputo, con $\mathbf{tp}$, e invocar una continuación, con las μ-variables ordinarias. Con la adición de $\mathbf{tp}$ también se añade la regla,

$$\mu\alpha'.[\mathbf{tp}](\mu\alpha.M) \rightarrow_{\beta'_\mu} \mu\alpha'.M[\alpha := [\mathbf{tp}][\,]]$$

Por consiguiente, la definición *a la Curry* de $\mathcal{A}$ es,

$$\mathcal{A} = \lambda x.\mu\beta.[\mathbf{tp}]x$$

En las fuentes (véase [13] y [32]) de donde nos estamos basando para definir el operador *abort* los contextos de evaluación E tienen la forma $[\,]N$,

[18]El nombre `tp` proviene de top level. Al decir que es una continuación de alto nivel nos referimos a que al evaluar cualquier término la continuación `tp` será la que abstrae la evaluación del término en su totalidad sin dejar cómputos pendientes, además puede considerarse como el fondo de la pila de la máquina abstracta subyacente a $\lambda\mathcal{C}$.

que es tomada del cálculo $\lambda\mathcal{C}$ en su versión de *llamada-por-nombre*, la misma estrategia de evaluación es utilizada por el cálculo $\lambda\mu$ aquí tratado. Sin embargo, ninguno de los dos artículos muestra la derivación de $E[\mathcal{A}M]$, nosotros la damos en seguida,

$$\begin{aligned} E[\mathcal{A}M] &= E[(\lambda x.\mu\beta.[\mathbf{tp}]x)M] \to_\beta E[\mu\beta.[\mathbf{tp}]M] \\ &= ([\,]N)[\mu\beta.[\mathbf{tp}]M] = (\mu\beta.[\mathbf{tp}]M)N \\ &\to_\varsigma \mu\sigma.[\mathbf{tp}]M \\ &\text{con } \sigma \text{ } \mu\text{-variable nueva} \end{aligned}$$

Vemos que la ejecución actual se ha abortado y se continúa con la evaluación de M en el contexto vacío representado por $\mathbf{tp}$, logrando capturar de forma precisa el comportamiento de $\mathcal{A}$. Como el contexto es vacío en $\mu\sigma.[\mathbf{tp}]M$ podemos ver esta expresión como M.

3.4.3. call/cc

El operador $\mathsf{call/cc}$ se introduce en el cálculo $\lambda\mathcal{C}$ de Felleisen con la reducción,

$$E[\mathsf{call/cc}\, M] \mapsto_{\mathsf{call/cc}} E[M(\lambda z.\mathcal{A}(E[z]))]$$

En el cálculo $\lambda\mu$ *restringido* se define,

$$\mathsf{call/cc} =_{def} \lambda f.\mu\alpha.[\alpha](f\lambda y.\mu\beta.[\alpha]y)$$

esta definición se encuentra en [13] y [32]; aunque en tales artículos no se proporciona la debida reducción, nosotros la ofrecemos enseguida, donde $E = [\,]N$, como ya se había mencionado antes. Entonces,

$$\begin{aligned} E[\mathsf{call/cc}\, M] &= E[(\lambda f.\mu\alpha.[\alpha](f\lambda y.\mu\beta.[\alpha]y))M] \\ &\to_\beta E[\mu\alpha.[\alpha](M\lambda y.\mu\beta.[\alpha]y)] \\ &= \big(\mu\alpha.[\alpha](M\lambda y.\mu\beta.[\alpha]y)\big)N \\ &\to_\varsigma \mu\gamma.\big([\gamma](M\lambda y.\mu\beta.[\gamma](yN))N\big) \end{aligned}$$

Formalmente debemos hacer explícita la presencia del fondo de la pila ($\mu\sigma.[\mathbf{tp}]\cdots$) en la expresión anterior, con lo cual tenemos,

$$\begin{aligned} \mu\sigma.[\mathbf{tp}]E[\mathsf{call/cc}\, M] &\to^*_\mu \mu\sigma.[\mathbf{tp}]\Big(\mu\gamma.\big([\gamma](M\lambda y.\mu\beta.[\gamma](yN))N\big)\Big) \\ &\to_{\beta'_\mu} \mu\sigma.[\mathbf{tp}](M\lambda y.\mu\beta.[\mathbf{tp}](yN))N \end{aligned}$$

regresando a la representación convencional, donde el fondo de la pila está implícito, escribimos,

$$= (M\lambda y.\mu\beta.[\mathbf{tp}](yN))N$$

Por otro lado, si empleamos la definición tradicional de $\mathsf{call/cc}$ tenemos,

$$\begin{aligned} E[\mathsf{call/cc}\, M] &\mapsto_{\mathsf{call/cc}} E[M\lambda z.\mathcal{A}(E[z])] \\ &= E[M\big(\lambda z.(\lambda x.\mu\beta.[\mathbf{tp}]x)(E[z])\big)] \\ &\to_{\beta} E[M\big(\lambda z.\mu\beta.[\mathbf{tp}]E[z]\big)] \\ &= \big(M(\lambda z.\mu\beta.[\mathbf{tp}](zN))\big)N \end{aligned}$$

Debido a que las expresiones finales son equivalentes entonces la definición que hemos mostrado para $\mathsf{call/cc}$ en el cálculo $\lambda\mu$ *restringido* es adecuada.

3.4.4. Operador $\mathcal{C}$ de Felleisen

La reducción que conocemos es,

$$E[\mathcal{C}M] \mapsto_{\mathcal{C}} M(\lambda z.\mathcal{A}(E[z]))$$

La nueva definición en el cálculo $\lambda\mu$ *restringido* es la siguiente,

$$\mathcal{C} =_{def} \lambda f.\mu\alpha.[\mathbf{tp}]f(\lambda x.\mu\beta.[\alpha]x)$$

la reducción correspondiente, que no aparece en [32] de donde es tomada la definición anterior, es,

$$\begin{aligned} E[\mathcal{C}M] &= E[(\lambda f.\mu\alpha.[\mathbf{tp}]f(\lambda x.\mu\beta.[\alpha]x))M] \\ &\to_{\beta} E[\mu\alpha.[\mathbf{tp}]M(\lambda x.\mu\beta.[\alpha]x)] \\ &= \big(\mu\alpha.[\mathbf{tp}]M(\lambda x\mu\beta.[\alpha]x)\big)N \\ &\to_{\varsigma} \mu\gamma.[\mathbf{tp}]M\big(\lambda x.\mu\beta.[\gamma](xN)\big) \end{aligned}$$

Una vez más hacemos explícita la existencia de $\mu\sigma.[\mathbf{tp}]\cdots$ en la expresión anterior, por lo que,

$$\begin{aligned} \mu\sigma.[\mathbf{tp}]E[\mathcal{C}M] &\to_{\mu}^{*} \mu\sigma.[\mathbf{tp}]\Big(\mu\gamma.[\mathbf{tp}]M\big(\lambda x.\mu\beta.[\gamma](xN)\big)\Big) \\ &\to_{\beta'_{\mu}} \mu\sigma.[\mathbf{tp}]M\big(\lambda x.\mu\beta.[\mathbf{tp}](xN)\big) \end{aligned}$$

esta expresión, sin mostrar el fondo de la pila, queda como,

$$= M\big(\lambda x.\mu\beta.[\mathbf{tp}](xN)\big)$$

Usando la definición de $\mathcal{C}$ proveniente del cálculo $\lambda\mathcal{C}$ de Felleisen tenemos,

$$\begin{aligned} E[\mathcal{C}M] &\mapsto_{\mathcal{C}} M\big(\lambda z.\mathcal{A}(E[z])\big) \\ &= M\big(\lambda z.(\lambda x.\mu\beta.[\mathbf{tp}]x)(E[z])\big) \\ &\to_{\beta} M\big(\lambda z.\mu\beta.[\mathbf{tp}]E[z]\big) \\ &= M\big(\lambda z.\mu\beta.[\mathbf{tp}](zN)\big) \end{aligned}$$

Por consiguiente, la definición propuesta para $\mathcal{C}$ en el cálculo $\lambda\mu$ *restringido* funciona correctamente. Notemos que la posibilidad de expresar al operador $\mathcal{C}$ en términos $\lambda\mu$ hace al cálculo creado por Parigot más poderoso que el cálculo $\lambda\mathcal{C}$.

Por el momento, dejaremos el cálculo $\lambda\mu$ *restringido* a un lado; empero, lo retomaremos en el siguiente capítulo.

3.5. Tipos productos y sumas

Por el Corolario 6.7.7 de [31] sabemos que los conectivos $\wedge$ y $\vee$ pueden ser integrados en $NK(\to, \bot)$, así que en esta sección estamos interesados en definir las operaciones para los tipos productos y sumas en el cálculo $\lambda\mu$, para este fin se seguirán las construcciones que ofrecen Sørensen y Urzyczyn en [31]. Recordemos que en el capítulo anterior esta tarea ya fue realizada para el cálculo $\lambda\mathcal{C}$ de Felleisen usando el operador $\mathcal{C}$.

La definición que manejamos para conjunción es la misma del capítulo anterior,

$$\mathsf{T}_1 \wedge \mathsf{T}_2 =_{def} \neg(\mathsf{T}_1 \to \neg\mathsf{T}_2)$$

Las operaciones de par ordenado y proyección, correspondientes al tipo producto, se expresan de la siguiente manera, donde w y α son variables nuevas,

Par ordenado. Sean P_1 y P_2 términos $\lambda\mu$ con tipos T_1 y T_2, respectivamente. Definimos,

$$\langle P_1, P_2\rangle =_{def} \lambda w : \mathsf{T}_1 \to \neg\mathsf{T}_2.wP_1P_2$$

Proyección. Tomamos Q con tipo $\neg(\mathsf{T}_1 \to \neg\mathsf{T}_2)$, luego,

$$\pi_i(Q) =_{def} \mu\alpha : \neg\mathsf{T}_i.Q(\lambda w_1 : \mathsf{T}_1 \lambda w_2 : \mathsf{T}_2.[\alpha]w_i)$$

Considérese como ejemplo la reducción,

$$\begin{aligned}\pi_2(\langle P_1, P_2\rangle) &= \mu\alpha : \neg\mathsf{T}_2.(\lambda w : \mathsf{T}_1 \to \neg\mathsf{T}_2.wP_1P_2)(\lambda w_1 : \mathsf{T}_1 \lambda w_2 : \mathsf{T}_2.[\alpha]w_2)\\ &\to_\beta \mu\alpha : \neg\mathsf{T}_2.(\lambda w_1 : \mathsf{T}_1 \lambda w_2 : \mathsf{T}_2.[\alpha]w_2)P_1P_2\\ &\to^*_\beta \mu\alpha : \neg\mathsf{T}_2.[\alpha]P_2\\ &\to_{\beta_\mu} P_2\end{aligned}$$

Para la disyunción retomamos la definición que se dio en el capítulo 2,

$$\mathsf{T}_1 \vee \mathsf{T}_2 =_{def} \neg\mathsf{T}_1 \to \neg\neg\mathsf{T}_2$$

A continuación se muestran la definiciones de las operaciones para el tipo suma, éstas son inyección y análisis por casos, además introducimos z_1, z_2 y α como variables nuevas,

Inyección. Sean P_1 y P_2 términos $\lambda\mu$ con tipos T_1 y T_2, respectivamente. Entonces,

$$\mathsf{inj}_i(P_i^{\mathsf{T}_i}) =_{def} \lambda z_1 : \neg\mathsf{T}_1 \lambda z_2 : \neg\mathsf{T}_2.z_iP_i \text{ con } i = \{1, 2\}$$

Análisis por casos. Si R tiene tipo $\neg\mathsf{T}_1 \to \neg\neg\mathsf{T}_2$ y S_1, S_2 son ambos de tipo S entonces,

$$\mathsf{case}\ R\ \{S_1 \mid S_2\} =_{def} \mu\alpha : \neg\mathsf{S}.R(\lambda x : \mathsf{T}_1.[\alpha]S_1)(\lambda y : \mathsf{T}_2.[\alpha]S_2)$$

La verificación de que estas definiciones son correctas se deja como ejercicio al lector.

Este capítulo se ha dedicado al cálculo $\lambda\mu$, el cual ha servido principalmente para mostrar el contenido computacional de la lógica clásica a través del *isomorfismo de Curry-Howard*. Se estudiaron sus propiedades y se exhibieron expresiones $\lambda\mu$ que definen algunos operadores de control y las operaciones correspondientes a los tipos sumas y productos.

En el capítulo siguiente se muestra la traducción de Kolmogorov, de la lógica clásica a la intuicionista, y se investiga la transformación CPS, del cálculo $\lambda\mu$ al cálculo $\lambda^\to$. Se indica además la relación que guardan entre si estas traducciones. Finalmente, se consideran de nueva cuenta los términos $\lambda\mu$ *restringidos* y se estudia lo que acontece cuando son transformados con CPS.

Capítulo 4

Traducciones lógicas

Es interesante analizar las traducciones que existen entre sistemas lógicos porque nos permiten interpretar fórmulas de un sistema bajo un enfoque lógico perteneciente a otro sistema, por ejemplo, es posible expresar a las proposiciones de la lógica clásica dentro de la lógica intuicionista; asimismo, son importantes las traducciones entre cálculos para conocer que la expresividad de un cálculo puede ser simulada en otro, digamos términos del cálculo $\lambda\mu$ se traducen en expresiones del cálculo $\lambda^{\rightarrow}$, como en el caso tratado en el capítulo anterior al probar la propiedad de normalización fuerte para $\lambda\mu$.

Este capítulo lo dedicaremos a investigar algunas traducciones entre sistemas lógicos y también entre cálculos. Comenzaremos con la *traducción de la doble negación de Kolmogorov*, o simplemente *traducción de Kolmogorov*, que da un sentido intuicionista a las fórmulas lógicas clásicas. En seguida se presenta la traducción CPS[19], en la que las continuaciones en una expresión son vistas como funciones, transformando términos $\lambda\mu$ en términos $\lambda^{\rightarrow}$. El capítulo continúa dando a conocer la interpretación computacional de la traducción de Kolmogorov, ya que resulta ser un caso especial de la traducción CPS sobre tipos. Por último, mostramos que la igualdad entre términos $\lambda\mu$ *restringidos* se conserva en $\lambda^{\rightarrow}$ al aplicar la transformación CPS.

4.1. Traducción de Kolmogorov

A partir de la década de los veinte el estudio de interpretaciones de la lógica clásica dentro de la lógica intuicionista tomó un gran impulso, como

[19]Continuation Passing Style.

se menciona en [31], Kolmogorov observó que al tener el prefijo $\neg\neg$ en cada subfórmula de una fórmula clásica válida se obtiene una fórmula intuicionista válida. Formalmente,

Definición 4.1. *Sea ρ una proposición atómica. La traducción de Kolmogorov se lleva a cabo de la siguiente manera,*

$$k(\rho) = \neg\neg\rho$$
$$k(\varphi \to \psi) = \neg\neg(k(\varphi) \to k(\psi))$$

Observemos que al definir $\varphi^\star$ como,

$$\rho^\star = \rho$$
$$(\varphi \to \psi)^\star = \neg\neg\varphi^\star \to \neg\neg\psi^\star$$

entonces la traducción de Kolmogorov toma la forma $k(\varphi) = \neg\neg\varphi^\star$. Además, convenimos para el contexto de tipos clásicos Γ que $k(\Gamma) = \{k(\varphi) \mid \varphi \in \Gamma\}$.

Lema 4.1. *Los siguientes secuentes son derivables en la lógica intuicionista.*

a) $\vdash \varphi \to \neg\neg\varphi$

b) $\vdash (\varphi \to \psi) \to \neg\psi \to \neg\varphi$

c) $\vdash \neg\neg\neg\varphi \to \neg\varphi$

d) $\vdash \neg\neg(\varphi \to \psi) \to (\varphi \to \neg\neg\psi)$

e) $\vdash \neg\neg(\neg\neg\varphi \to \neg\neg\psi) \leftrightarrow \neg\neg\varphi \to \neg\neg\psi$

f) $\vdash \neg\neg(\neg\neg\varphi \to \neg\neg\psi) \leftrightarrow \neg\neg(\varphi \to \psi)$

Demostración.

- Las demostraciones para a) y b) son triviales.
- La prueba de c) consta en derivar $\Pi \vdash_i \bot$, donde $\Pi = \{((\varphi \to \bot) \to \bot) \to \bot, \varphi\}$. Entonces,

$$\dfrac{\dfrac{\dfrac{}{\Pi, \varphi \to \bot \vdash_i \varphi \to \bot}\,(Ax) \quad \dfrac{}{\Pi, \varphi \to \bot \vdash_i \varphi}\,(Ax)}{\Pi, \varphi \to \bot \vdash_i \bot}\,(\to E)}{\Pi \vdash_i (\varphi \to \bot) \to \bot}\,(\to I)$$

Por *Modus Ponens* de este resultado y $\Pi \vdash_i ((\varphi \to \bot) \to \bot) \to \bot$ obtenemos $\Pi \vdash_i \bot$.

- Para la prueba de d) consideramos $\Gamma = \{((\varphi \to \psi) \to \bot) \to \bot, \varphi, \psi \to \bot\}$. De modo que buscamos derivar $\Gamma \vdash_i \bot$. Si $\Phi = \Gamma \cup \{\varphi \to \psi\}$, entonces,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Phi \vdash_i \varphi \to \psi}(Ax) \quad \dfrac{}{\Phi \vdash_i \varphi}(Ax)}{\Phi \vdash_i \psi}(\to E) \quad \dfrac{}{\Phi \vdash_i \psi \to \bot}(Ax)}{\Phi \vdash_i \bot}(\to E)}{\Gamma \vdash_i (\varphi \to \psi) \to \bot}(\to I)$$

Usando esta derivación y el axioma $\Gamma \vdash_i ((\varphi \to \psi) \to \bot) \to \bot$, logramos, por *Modus Ponens*, $\Gamma \vdash_i \bot$.

- La prueba $\leftarrow$ de e) es un caso particular de a). Para la demostración $\to$ de e) usaremos los incisos c) y d). La derivación a mostrar es $\Delta \vdash_i \neg\neg\psi$, con $\Delta = \{\neg\neg(\neg\neg\varphi \to \neg\neg\psi), \neg\neg\varphi\}$. Así,

$$\dfrac{\dfrac{\dfrac{}{\Delta \vdash_i \neg\neg(\neg\neg\varphi \to \neg\neg\psi)}(Ax)}{\Delta \vdash_i \neg\neg\varphi \to \neg\neg\neg\neg\psi}\ \textit{inciso d)}}$$

Hemos obtenido que $\Delta \vdash_i \neg\neg\varphi \to \neg\neg\neg\neg\psi$; asimismo, por inciso c) es cierto que $\Delta \vdash_i \neg\neg\neg\neg\psi \to \neg\neg\psi$, usando transitividad de la implicación de estas dos derivaciones tenemos $\Delta \vdash_i \neg\neg\varphi \to \neg\neg\psi$ $(*)$. Finalmente,

$$\dfrac{\dfrac{}{\Delta \vdash_i \neg\neg\varphi \to \neg\neg\psi}(*) \quad \dfrac{}{\Delta \vdash_i \neg\neg\varphi}(Ax)}{\Delta \vdash_i \neg\neg\psi}(\to E)$$

- La demostración $\leftarrow$ de f) es la siguiente, si tomamos $\Pi = \{\neg\neg(\varphi \to \psi), (\neg\neg\varphi \to \neg\neg\psi) \to \bot\}$, entonces debemos mostrar que $\Pi \vdash_i \bot$. Luego,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Pi \vdash_i \neg\neg(\varphi \to \psi)}(Ax)}{\Pi \vdash_i \varphi \to \neg\neg\psi}\ \textit{inciso d)}}{\Pi \vdash_i \neg\psi \to \neg\varphi}\ \textit{(Prop 2.2) pág. 46}}{\Pi \vdash_i \neg\neg\varphi \to \neg\neg\psi}\ \textit{inciso b)} \quad \dfrac{}{\Pi \vdash_i (\neg\neg\varphi \to \neg\neg\psi) \to \bot}(Ax)}{\Pi \vdash_i \bot}(\to E)$$

Para demostrar la implicación de izquierda a derecha de f) es suficiente probar, por inciso e), que $(\neg\neg\varphi \to \neg\neg\psi) \to \neg\neg(\varphi \to \psi)$. Sea $\Gamma = \{\neg\neg\varphi \to \neg\neg\psi, \neg(\varphi \to \psi)\}$, deseamos derivar $\Gamma \vdash_i \bot$.

Por un lado,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi, \neg\varphi \vdash_i \neg\varphi}\,(Ax) \quad \dfrac{}{\Gamma, \varphi, \neg\varphi \vdash_i \varphi}\,(Ax)}{\Gamma, \varphi, \neg\varphi \vdash_i \bot}\,(\to E)}{\Gamma, \varphi, \neg\varphi \vdash_i \psi}\,(EFQ)}{\Gamma, \neg\varphi \vdash_i \varphi \to \psi}\,(\to I) \qquad \dfrac{}{\Gamma, \neg\varphi \vdash_i \neg(\varphi \to \psi)}\,(Ax)}{\Gamma, \neg\varphi \vdash_i \bot}\,(\to E)}{\Gamma \vdash_i \neg\neg\varphi}\,(\to I)$$

A partir de esta derivación y el axioma $\Gamma \vdash_i \neg\neg\varphi \to \neg\neg\psi$, la regla $(\to E)$ indica que $\Gamma \vdash_i \neg\neg\psi$.

Por otro lado,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi, \psi \vdash_i \psi}\,(Ax)}{\Gamma, \psi \vdash_i \varphi \to \psi}\,(\to I) \qquad \dfrac{}{\Gamma, \psi \vdash_i \neg(\varphi \to \psi)}\,(Ax)}{\Gamma, \psi \vdash_i \bot}\,(\to E)}{\Gamma \vdash_i \neg\psi}\,(\to I)$$

Así, hemos llegado a que $\Gamma \vdash_i \neg\neg\psi$ y $\Gamma \vdash_i \neg\psi$, por $(\to E)$ obtenemos el resultado buscado $\Gamma \vdash_i \bot$.

□

El teorema que se presenta a continuación nos dice que la traducción de Kolmogorov traduce fórmulas clásicas válidas en fórmulas intuicionistas válidas.

Teorema 4.1. *Es verdad que,*

1. $\vdash_c \varphi \to k(\varphi)$ *y* $\vdash_c k(\varphi) \to \varphi$ *en* $NK(\to, \bot)$
2. $\vdash_c \varphi$ *en* $NK(\to, \bot)$ *si y solo si* $\vdash_i k(\varphi)$ *en* $NJ(\to, \bot)$

Demostración. La prueba del inciso 1 requiere del teorema de completud, para mayores detalles consulte [31].

Para la implicación de derecha a izquierda del inciso 2 suponemos $\vdash_i k(\varphi)$ en $NJ(\to,\bot)$, luego $\vdash_c k(\varphi)^{(\clubsuit)}$ en $NK(\to,\bot)$ por la proposición 1.2 del capítulo 1. Por el inciso 1 de este teorema, $\vdash_c k(\varphi)\to\varphi^{(\spadesuit)}$ en $NK(\to,\bot)$, y por *Modus Ponens* de $(\clubsuit)$ y $(\spadesuit)$ concluimos $\vdash_c \varphi$ en $NK(\to,\bot)$.

Para la implicación de izquierda a derecha del inciso 2 procedemos por inducción sobre la derivación $\Gamma\vdash_c\varphi$ en $NK(\to,\bot)$, el argumento que sigue es más claro que el que encuentra en [31] donde se prueba el mismo teorema.

- *Base de inducción.* Si $\Gamma,\varphi\vdash_c\varphi$, entonces $k(\Gamma),k(\varphi)\vdash_i k(\varphi)$ es cierto en $NJ(\to,\bot)$ ya que corresponde a la regla (Ax).

- *Paso inductivo.* Por demostrar que la afirmación es cierta para la inferencia $\Gamma\vdash_c\varphi$ en $NK(\to,\bot)$.

  - Si la derivación termina usando la regla $(\to I)$ se tiene,

$$\dfrac{\dfrac{\dfrac{}{k(\Gamma),k(\varphi)\vdash_i k(\psi)}\,HI}{k(\Gamma)\vdash_i k(\varphi)\to k(\psi)}\,(\to I)}{k(\Gamma)\vdash_i \neg\neg(k(\varphi)\to k(\psi))}\,\text{lema 4.1.a)}$$

    Como $\neg\neg(k(\varphi)\to k(\psi))=k(\varphi\to\psi)$ concluimos $k(\Gamma)\vdash_i k(\varphi\to\psi)$.

  - Cuando termina con la regla $(\to E)$ ocupamos que $k(\varphi)=\neg\neg\varphi^\star$, luego,

$$\dfrac{\dfrac{}{k(\Gamma)\vdash_i \neg\neg\varphi^\star}\,HI \qquad \dfrac{\dfrac{}{k(\Gamma)\vdash_i \neg\neg(\neg\neg\varphi^\star\to\neg\neg\psi^\star)}\,HI}{k(\Gamma)\vdash_i \neg\neg\varphi^\star\to\neg\neg\psi^\star}\,\text{lema 4.1.e)}}{k(\Gamma)\vdash_i \neg\neg\psi^\star}\,(\to E)$$

    Esto es $k(\Gamma)\vdash_i k(\psi)$.

  - Finalmente, si la última regla en ocuparse es $(\neg\neg E)$ entonces,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{k(\Gamma),\neg\neg(k(\varphi)\to\neg\neg\bot)\vdash_i\neg\neg\bot}\,HI}{k(\Gamma)\vdash_i\neg\neg(k(\varphi)\to\neg\neg\bot)\to\neg\neg\bot}\,(\to I)}{k(\Gamma)\vdash_i\neg\neg(\neg\neg(k(\varphi)\to\neg\neg\bot)\to\neg\neg\bot)}\,\textit{lema 4.1.e)}}{k(\Gamma)\vdash_i\neg\neg((k(\varphi)\to\neg\neg\bot)\to\bot)}\,\textit{lema 4.1.f)}$$

Lo cual se puede ver como $k(\Gamma) \vdash_i \neg\big(\neg\neg(\neg\neg\varphi^\star \to \neg\neg\bot)\big)^{(\#)}$. Luego,

$$\dfrac{\dfrac{\dfrac{}{k(\Gamma) \vdash_i \neg\big(\neg\neg(\neg\neg\varphi^\star \to \neg\neg\bot)\big)}\,(\#)}{k(\Gamma) \vdash_i \neg\neg\neg(\varphi^\star \to \bot)}\,\textit{lema 4.1.f)}}{k(\Gamma) \vdash_i \neg\neg\varphi^\star}\,\textit{lema 4.1.c)}$$

La última tipificación puede considerarse como $k(\Gamma) \vdash_i k(\varphi)$.

□

En la demostración del inciso 2 se tiene una traducción implícita de pruebas clásicas a pruebas intuicionistas. Desde el punto de vista del cálculo $\lambda\mu$ esta traducción elimina presencias de μ, justo como lo hace la transformación CPS que tratamos a continuación.

4.2. Programación CPS

La idea consiste en que dado un término M cada subtérmino N sea parametrizado por una función, una *continuación*, que al recibir el valor de N regrese el valor total del término M. Para ilustrar la transformación CPS se admitirán números naturales y sumas como términos del cálculo $\lambda\mu$. Por ejemplo, el término 17 se convierte en $\lambda l.l17$, el número 17 es un valor y no require de cómputos adicionales, así que a la continuación l se le aplica directamente el valor 17.

La transformación CPS de $17 + 23$ da como resultado,

$$\lambda h.\underline{17}(\lambda u.\underline{23}(\lambda v.h(u + v)))$$

en donde $\underline{17}$ y $\underline{23}$ son las tranformaciones CPS de 17 y 23, respectivamente, de modo que $\underline{17} = \lambda l.l17$ y $\underline{23} = \lambda k.k23$. La suma total es un término parametrizado por la continuación h, ya que se tiene esa continuación entonces la evaluación produce el valor 17, el cual se pasa a la continuación $\lambda u \ldots$, para dar lugar a la producción del número 23 y de su aplicación con la continuación $\lambda v \ldots$. Para terminar, la continuación global h es invocada con la suma $17 + 23$.

El término $(17 + 23) + 1$ en CPS es

$$\lambda k.\big(\lambda h.\underline{17}(\lambda u.\underline{23}(\lambda v.h(u + v)))\big)(\lambda m.\underline{1}(\lambda n.k(m + n)))$$

En esta ocasión h representa el cómputo pendiente $[\,]+1$, por lo que las presencias de h son sustituidas por la continuación $\lambda m\ldots$. Después m recibe el valor 40 resultado de 17+23, el valor 1 es producido y pasado a la continuación $\lambda n\ldots$, obteniendo el $40+1$ como resultado final y la continuación global k invocada con este valor. Si $(17+23)+1$ es la expresión a traducir entonces debemos aplicar el término CPS que acabamos de obtener a la continuación de *alto nivel* $\lambda g.g$[20]. Así la k se sustituirá por $\lambda g.g$ y 41 será el valor que se regrese. Es decir,

$$\Big(\lambda k.\big(\lambda h.\underline{17}(\lambda u.\underline{23}(\lambda v.h(u+v)))\big)(\lambda m.\underline{1}(\lambda n.k(m+n)))\Big)(\lambda g.g) \to_\beta^* 41$$

Tratemos ahora los términos que definen al cálculo $\lambda\mu$.

Para la traducción de la abstracción $\lambda x.M$ observamos que se trata de un valor, no necesitando de cómputos adicionales para que se lleve a cabo la reducción de $(\lambda x.M)N$; no obstante, se requiere traducir los subtérminos que existan en M. La traducción queda como,

$$\underline{\lambda x.M} = \lambda k.k(\lambda x.\underline{M})$$

En la aplicación MN requerimos que M se reduzca primeramente a un valor m para producir la aplicación mN. Por lo tanto, la forma de la traducción es $\underline{MN} = \lambda k.\underline{M}(\lambda m\ldots m\underline{N}\ldots)$. Pero en $m\underline{N}$ es posible que más reducciones se ejecuten para obtener un valor. Los valores, como cualquier otro término que se traduce, esperan que se les pase una continuación, así que una vez calculado el valor recibirá la continuación k. La traducción final es,

$$\underline{MN} = \lambda k.\underline{M}(\lambda m.m\underline{N}k)$$

Notemos que,

$$\begin{aligned}\underline{(\lambda x.M)N} = \lambda k.(\lambda k'.k'(\lambda x.\underline{M}))(\lambda m.m\underline{N}k) &\to_\beta^* \lambda k.(\lambda x.\underline{M})\underline{N}k\\ &\to_\beta \lambda k.\underline{M}[x := \underline{N}]k\end{aligned}$$

entonces x se sustituye por el término traducido $\underline{N}$, pero los términos traducidos esperan su aplicación con una continuación k, luego,

$$\underline{x} = \lambda k.xk$$

[20]La continuación $\lambda g.g$ es de alto nivel en el sentido de que el subtérmino al que parametriza es el término final, sin que haya algún otro cómputo pendiente a realizar.

Para la traducción de la abstracción y aplicación μ considérese la expresión,

$$2 + \mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)$$

donde $[\alpha]13$ ignora el contexto $[\alpha](11 + \mu\beta.[\,])$, de modo que la traducción de $\mu\alpha\ldots$ introduce una continuación a correspondiente al contexto $2 + [\,]$ y la traducción del término $[\alpha]13$ desprecia la continuación que se asocia a su contexto y usa la continuación a. Ahora bien, al traducir $\mu\alpha.M$ se llevará a cabo la traducción de M en $\underline{M}$, esta última aguarda una continuación k que será ignorada debido a que en M se espera que aparezca $[\alpha]N$ quien desechará la continuación k, así que cualquier continuación que se le pase a $\underline{M}$ funciona, digamos $\lambda v.v$. Las traducciones buscadas son,

$$\underline{\mu\alpha.M} = \lambda a.\underline{M}(\lambda v.v)$$
$$\underline{[\alpha]M} = \lambda y.\underline{M}a$$

Como ejemplo la traducción de $2 + \mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)$ y su derivación correspondiente se muestran enseguida, para este fin usaremos que,

$$\underline{\mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)} = \lambda a.\Big(\lambda y.\Big(\lambda w.\underline{11}\big(\lambda z.(\lambda b.(\lambda x.\underline{13}a)(\lambda e.e))(\lambda f.w(z + f))\big)\Big)a\Big)(\lambda d.d)$$

entonces,

$$\begin{aligned}
&\underline{2 + \mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)} = \\
&\qquad\Big(\lambda h.\underline{2}\Big(\lambda u.\underline{\mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)}(\lambda v.h(u + v))\Big)\Big)(\lambda g.g)\\
\to^*_\beta\ &\underline{\mu\alpha.[\alpha](11 + \mu\beta.[\alpha]13)}m\\
\to^*_\beta\ &\Big(\lambda w.\underline{11}\big(\lambda z.(\lambda b.(\lambda x.\underline{13}m)(\lambda e.e))(\lambda f.w(z + f))\big)\Big)m\\
\to^*_\beta\ &(\lambda b.(\lambda x.\underline{13}m)(\lambda e.e))(\lambda f.m(11 + f))\\
\to_\beta\ &(\lambda x.\underline{13}m)(\lambda e.e)\\
\to_\beta\ &\underline{13}m\\
\to^*_\beta\ &(\lambda g.g)(2 + 13)\\
\to_\beta\ &15
\end{aligned}$$

donde,

$$m = \lambda v.(\lambda g.g)(2 + v)$$

4.2.1. Transformación CPS sobre tipos

Si $\mathcal{P}$ es un término $\lambda\mu$ de tipo **Nat** y $\mathcal{Q}$ un subtérmino de $\mathcal{P}$ con tipo **Nat**, entonces $\underline{\mathcal{Q}}$ está parametrizado por una continuación que eventualmente tomará el valor de $\mathcal{Q}$ y dará como resultado el valor de $\mathcal{P}$. Esta continuación tendrá tipo $\mathbf{Nat} \to \mathbf{Nat}$, así que el tipo de $\underline{\mathcal{Q}}$, que está parametrizado por dicha continuación, será de tipo $(\mathbf{Nat} \to \mathbf{Nat}) \to \mathbf{Nat}$. Si el tipo de $\mathcal{Q}$ fuera **Bool**, entonces $\underline{\mathcal{Q}}$ sería de tipo $(\mathbf{Bool} \to \mathbf{Nat}) \to \mathbf{Nat}$. Si abstraemos el tipo de $\mathcal{Q}$ en Q, entonces $\underline{\mathcal{Q}}$ tendría tipo $T(\mathsf{Q})$, donde,

$$T(\mathsf{Q}) = (\mathsf{Q} \to \mathbf{Nat}) \to \mathbf{Nat}$$

Tomemos el caso cuando $\mathcal{Q}$ es una función de tipo $\mathsf{R} \to \mathsf{S}$. El argumento que se le pasa a la función ya transformada también está en CPS, de modo que tendrá tipo $T(\mathsf{R})$. Además los cómputos que se ejecutan dentro de la función siguen el estilo de programación CPS, por lo que el tipo de regreso será $T(\mathsf{S})$. Consecuentemente, el tipo de $\underline{\mathcal{Q}}$ será,

$$T(\mathsf{R} \to \mathsf{S}) = ((T(\mathsf{R}) \to T(\mathsf{S})) \to \mathbf{Nat}) \to \mathbf{Nat}$$

Si el tipo del término $\mathcal{P}$ es $\bot$ y si pensamos en la traducción de pruebas clásicas a intuicionistas que induce el teorema 4.1 como una transformación CPS sobre términos, entonces la función T que se muestra arriba se asocia a la traducción de Kolmorogov k, la cual puede ser considerada un caso especial de la transformación CPS sobre tipos que acabamos de estudiar.

Hasta ahora la transformación CPS ha manipulado los términos del cálculo $\lambda\mu$ en un estilo *a la Curry*, pero una vez que se ha analizado lo que sucede con los tipos de las expresiones $\lambda\mu$ podemos dar la definición formal en un estilo *a la Church*.

Definición 4.2 (Transformación CPS)**.** *Retomando el significado de* $\mathsf{T}^\star$ *que se dio en la definición 4.1 de la traducción de Kolmogorov, los términos* $\lambda\mu$ *se traducen en términos* $\lambda^\to$ *del siguiente modo,*

$$\begin{aligned}
\underline{x^{\mathsf{T}}} &= \lambda h : \neg\mathsf{T}^\star.x^{k(\mathsf{T})}h \\
\underline{\lambda x : \mathsf{T}.M^{\mathsf{S}}} &= \lambda h : \neg(k(\mathsf{T}) \to k(\mathsf{S})).h(\lambda x : k(\mathsf{T}).\underline{M}^{k(\mathsf{S})}) \\
\underline{MN} &= \lambda h : \neg\mathsf{T}^\star.\underline{M}^{\neg\neg(k(\mathsf{S})\to k(\mathsf{T}))}(\lambda m : k(\mathsf{S}) \to k(\mathsf{T}).m\underline{N}^{k(\mathsf{S})}h) \\
\underline{\mu\alpha : \neg\mathsf{T}.M^{\bot}} &= \lambda a : \neg\mathsf{T}^\star.\underline{M}^{\neg\neg\bot}(\lambda v : \bot.v) \\
\underline{[\alpha^{\neg\mathsf{T}}]M^{\mathsf{T}}} &= \lambda y : \neg\bot.\underline{M}^{k(\mathsf{T})}a^{\neg\mathsf{T}^\star}
\end{aligned}$$

La traducción CPS se extiende a contextos como sigue,

$$\underline{\Gamma} = \{x : k(\mathsf{T}) \,|\, x : \mathsf{T} \in \Gamma\} \cup \{a : \neg\mathsf{T}^\star \,|\, \alpha : \neg\mathsf{T} \in \Gamma\}$$

El teorema que viene a continuación expresa el contenido computacional de la traducción de Kolmogorov por medio de la transformación CPS.

Teorema 4.2. *Si* $\Gamma \vdash M : \mathsf{T}$ *en* $\lambda\mu$*, entonces* $\underline{\Gamma} \vdash \underline{M} : k(\mathsf{T})$ *en* $\lambda^{\to}$*.*

Demostración. Inducción sobre la derivación de tipos $\Gamma \vdash M : \mathsf{T}$ del cálculo $\lambda\mu$.

- *Base de inducción.* Cuando transformamos $\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}$ usando CPS obtenemos,

  $$\underline{\Gamma}, x : k(\mathsf{T}) \vdash \lambda h : \neg\mathsf{T}^\star.x^{k(\mathsf{T})}h : \neg\mathsf{T}^\star \to \bot$$

  Podemos escribir esto como,

  $$\underline{\Gamma}, x : k(\mathsf{T}) \vdash \underline{x} : k(\mathsf{T})$$

- *Paso inductivo.* Por demostrar que el teorema se cumple para $\Gamma \vdash M : \mathsf{T}$.

  Como muestra tomemos el caso cuando la derivación termina con $\Gamma \vdash \mu\alpha : \neg\mathsf{T}.M' : \mathsf{T}$. Sabemos que $\underline{\mu\alpha : \neg\mathsf{T}.M'} = \lambda a : \neg\mathsf{T}^\star.\underline{M'}(\lambda v : \bot.v)$. Por hipótesis de inducción se satisface que $\underline{\Gamma}, a : \neg\mathsf{T}^\star \vdash \underline{M'} : k(\bot)$ en $\lambda^{\to}$ con $k(\bot) = \neg\neg\bot$, además es fácil ver que $\underline{\Gamma}, a : \neg\mathsf{T}^\star \vdash \lambda v : \bot.v : \bot \to \bot$, luego $\underline{\Gamma}, a : \neg\mathsf{T}^\star \vdash M'(\lambda v : \bot.v) : \bot$. Usando la regla $(RFun)$ se tiene,

  $$\underline{\Gamma} \vdash \lambda a : \neg\mathsf{T}^\star.\underline{M'}(\lambda v : \bot.v) : \neg\mathsf{T}^\star \to \bot$$

  Es decir,

  $$\underline{\Gamma} \vdash \underline{\mu\alpha : \neg\mathsf{T}.M'} : k(\mathsf{T})$$

□

4.2.2. Términos restringidos

En el capítulo anterior se definió el cálculo $\lambda\mu$ *restringido*, recordemos que sus términos se forman de la siguiente manera, usando un estilo *a la Church*,

$$M ::= x \mid MM \mid \lambda x : \mathsf{T}.M \mid \mu\alpha : \neg\mathsf{T}.[\beta]M$$

Esta definición simplemente nos indica que la formación de términos $\lambda\mu$ es la adecuada, en el sentido de que no se concibe una abstracción μ sin su correspondiente aplicación, y viceversa. Es decir, en una abstración μ necesariamente aparecerá una aplicación μ y siempre que se tenga una aplicación $[\beta]M$ entonces estará acompañada del operador de ligado $\mu\alpha$, donde puede suceder que $\alpha = \beta$, en cuyo caso α es el canal por el cual el valor es transmitido; o bien, $\alpha \neq \beta$, entonces $\mu\alpha : \neg\mathsf{T}.[\beta]M$ sirve para introducir la aplicación μ al contexto de la expresión sin incurrir en una violación de tipos.

Definición 4.3. *Las relaciones $=_\beta$(igualdad-β) e $=_\mu$(igualdad-μ) denotan las menores relaciones de equivalencia conteniendo $\to_\beta$ y $\to_\mu$, respectivamente.*

Considérense los siguientes ejemplos,

- $(\lambda x.x)yz =_\beta y((\lambda x.x)z)$.

  Esto es porque $(\lambda x.x)yz \to_\beta yz$, y también $y((\lambda x.x)z) \to_\beta yz$.

- $[\beta](\mu\alpha_0 : \neg\mathsf{T}.M) =_\mu \mu\alpha_1 : \neg\mathsf{T}.[\alpha_1]M$ con $\alpha_0, \alpha_1 \notin FV(M)$.

  Esta igualdad-μ se debe a que $[\beta](\mu\alpha_0 : \neg\mathsf{T}.M) \to_{\beta_\mu} M[\alpha_0 := [\beta][]] = M$ ya que $\alpha_0 \notin FV(M)$. Por otro lado, $\mu\alpha_1 : \neg\mathsf{T}.[\alpha_1]M \to_{\eta_\mu} M$ porque $\alpha_1 \notin FV(M)$.

Lema 4.2. *Si P y Q son términos $\lambda\mu$ restringidos entonces es cierto que,*

$$\begin{array}{lll} \lambda k : \mathsf{T}.\underline{P}k & =_\beta \underline{P} & (\#) \\ \underline{P[x := Q]} & =_\beta \underline{P}[x := \underline{Q}] & \\ \underline{P[\alpha := [\beta][\,]]} & =_\beta \underline{P}[a := b] & \\ \underline{P[\alpha := [\beta]([\,]Q)} & =_\beta \underline{P}[a := \lambda m^{k(\mathsf{T})\to k(\mathsf{S})}.m\underline{Q}b] & \end{array}$$

Demostración. La propiedad # es válida porque $\underline{P}$ es siempre una abstracción, digamos $\underline{P} = \lambda f : \mathsf{T}.M$, entonces

$$\begin{aligned} \lambda k : \mathsf{T}.\underline{P}k &= \lambda k : \mathsf{T}.(\lambda f : \mathsf{T}.M)k \\ &\to_\beta \lambda k : \mathsf{T}.M[f := k] \end{aligned}$$

la sustitución nos arroja un término que es α-equivalente a $\underline{P}$; por consiguiente, es válido decir que,

$$\lambda k : \mathsf{T}.\underline{P}k =_\beta \underline{P}$$

Para probar las demás propiedades se procede por inducción sobre P usando la propiedad #. □

Lema 4.3. *Si $P \to_\mu Q$ entonces $\underline{P} \to_\beta^* \underline{Q}$, con P y Q términos $\lambda\mu$ restringidos.*

Demostración. La prueba es por inducción sobre $P \to_\mu Q$ usando el lema 4.2. Como ejemplo tomemos el caso base para $\to_{\beta_\mu}$, notemos que como los términos son restringidos debemos hacer explícita la presencia del operador de ligado $\mu\alpha'$ en la expresión. La reducción, usando el estilo *a la Curry*, es,

$$P = \mu\alpha'.[\beta](\mu\alpha.M) \to_{\beta_\mu} \mu\alpha'.M[\alpha := [\beta][\,]] = Q$$

entonces,

$$\begin{aligned}
\underline{P} = \underline{\mu\alpha'.[\beta](\mu\alpha.M)} &= \lambda a'.\Big(\lambda y.\big(\lambda a.\underline{M}(\lambda u.u)\big)b\Big)(\lambda v.v) \\
&\to_\beta^* \lambda a'.\big((\underline{M}(\lambda u.u))[a := b]\big)[y := \lambda v.v] \\
&= \lambda a'.\underline{M}[a := b](\lambda u.u) \\
&= \lambda a'.\underline{M[\alpha := [\beta][\,]]}(\lambda u.u) \text{ por lema 4.2} \\
&= \underline{\mu\alpha'.M[\alpha := [\beta][\,]]} = \underline{Q}
\end{aligned}$$

□

Como resultado importante se cumple que la transformación CPS preserva la igualdad de términos restringidos como lo estipula el teorema que sigue,

Teorema 4.3. *Si $M =_\mu N$ entonces $\underline{M} =_\beta \underline{N}$, dados cualesquiera términos $\lambda\mu$ M y N restringidos.*

Demostración. La idea de la prueba fue tomada de [31], sólo que aquí se ha desglosado en los lemas 4.2 y 4.3, y en el argumento siguiente.

Al suponer $M =_\mu N$ entonces existe $\lambda\mu$-término L tal que $M \to_\mu^* L$ y $N \to_\mu^* L$, por transitividad de $=_\mu$ es suficiente analizar el caso cuando $M \to_\mu^* N$. Notemos que los términos $\lambda\mu$ restringidos son cerrados bajo la reducción, así que podemos usar el lema 4.3 en cada paso de $M \to_\mu^* N$, entonces llegamos a que $\underline{M} \to_\beta^* \underline{N}$. Luego, $\underline{M} =_\beta \underline{N}$. □

El hecho de que M y N sean términos restringidos en el teorema 4.3 es vital. Examinemos la situación siguiente, donde por comodidad las expresiones están en el estilo *a la Curry*,

$$M = [\beta]\mu\alpha.x \quad =_\mu \quad x = N$$

por otro lado,

$$\underline{M} = \lambda y.(\lambda a.(\lambda h.xh)(\lambda v.v))b \quad \neq_\beta \quad \lambda h.xh = \underline{N}$$

En este capítulo se dieron a conocer la traducción lógica de Kolmogorov, de lógica clásica a la intuicionista, y la transformación CPS, que convierte términos $\lambda\mu$ a términos $\lambda^{\rightarrow}$. Se mostró cómo es que el tipo de un término $\lambda\mu$ que se transforma utilizando CPS cambia siguiendo la traducción de Kolmogorov. Finalmente, se presentó un resultado para términos $\lambda\mu$ restringidos que nos dice que la transformación CPS preserva la igualdad en $\lambda^{\rightarrow}$.

Capítulo 5

Conclusiones

La importancia que la lógica tiene dentro de las Ciencias de la Computación es evidente al estudiar formalmente los lenguajes de programación, basta observar cómo es que los sistemas de deducción natural poseen un significado computacional; por ejemplo, en el capítulo 1 se investigó la correspondencia existente entre el cálculo proposicional intuicionista, donde los conectivos involucrados fueron: la implicación, disyunción y conjunción; y el cálculo λ de Church con tipos simples extendido con tipos sumas y productos, este hecho es conocido como *isomorfismo de Curry-Howard*.

Surge entonces la siguiente interrogante: ¿existirá algún lenguaje o formalismo matemático que se corresponda con la lógica clásica? Por un tiempo prolongado se pensó que no existía tal lenguaje y se consideraba a la lógica clásica irrelevante computacionalmente hablando. Afortunadamente, la respuesta a la pregunta planteada es afirmativa. En este trabajo se ha mostrado el contenido computacional de la lógica clásica, con base en el trabajo realizado por Griffin y Parigot, se ha logrado establecer el *ismorfismo de Curry-Howard* entre la lógica clásica y los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$. El capítulo 2 introdujo el cálculo $\lambda\mathcal{C}$ de Felleisen para el cual se investigó una tipificación que permitió la correspondencia con la lógica clásica cumpliéndose el *isomorfismo de Curry-Howard*. En el capítulo 3 encontramos otro lenguaje que también satisface este isomorfismo, nos referimos al cálculo $\lambda\mu$, en donde las continuaciones son abstraídas en variables μ, lo que hace a este lenguaje propicio para la definición de operadores de control. Como se apreció a lo largo de este trabajo, la capacidad en los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$ de manipular el contexto de evaluación es el ingrediente esencial para formular la correspondencia con la lógica clásica, la razón es que a las reglas de tipificación para el operador $\mathcal{C}$

y para las abstracciones μ se les asocia la regla de eliminación de la doble negación de NK. Consecuentemente, el poder definir operadores de control en estos lenguajes nos permite estudiar formalmente su comportamiento usando lógica matemática.

Continuando con el paralelismo entre fórmulas lógicas y tipos en lenguajes de programación, se indagó el contenido computacional de la traducción de Kolmogorov en el capítulo 4. Vimos que la transformación CPS convierte términos $\lambda\mu$ en términos $\lambda^{\rightarrow}$, pero lo que ocurre con los tipos de los términos antes y después de ser transformados es sorprendente, ya que cambian de acuerdo a la traducción de Kolmogorov.

La investigación que se ha realizado acerca de la relación entre la lógica clásica y los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$, además de la definición de operadores de control en estos lenguajes, es muestra de la aplicación que tiene la lógica matemática dentro del ámbito de Ciencias de la Computación.

Este trabajo puede ser de interés como una primera aproximación a aquellos estudiantes que desean conocer los fundamentos de lenguajes de programación, en particular lo referente al sistema de tipos y su relación con la deducción natural, profundizando después en el estudio de continuaciones y operadores de control al extender el cálculo λ a otros lenguajes que controlan el contexto de evaluación como son $\lambda\mathcal{C}$ y $\lambda\mu$, para los cuales se satisface el *isomorfismo de Curry-Howard* con la lógica clásica.

Una futura investigación podría enfocarse en el desarrollo y evolución del cálculo $\lambda\mu$, así como en sus múltiples variantes (véase [14]). También es interesante analizar la relación entre los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$ que de Groote propone en [11], donde exhibe una traducción entre ambos cálculos empleando cálculo de secuentes. Por último, Ariola y Herbelin en [1] realizan una clasificación interesante de los axiomas que definen a la lógica clásica, dividiéndolos en axiomas clásicos débiles, minimales y completos, lo que origina distintas versiones de los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$.

Apéndice A

Definición de los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$

Este apéndice tiene la finalidad de ser una referencia rápida y accesible para la definición de los cálculos $\lambda\mathcal{C}$ y $\lambda\mu$, actores principales en todo este trabajo, mostrando la grámatica que forma a los términos de cada cálculo, así como las reglas de tipificación y reducciones respectivas.

A.1. Cálculo $\lambda\mathcal{C}$

La gramática que define al cálculo $\lambda\mathcal{C}$, en un estilo *a la Curry*, se muestra a continuación,

$$M ::= x \mid \lambda x.M \mid MM \mid \mathcal{C}M \mid \mathcal{A}M$$

Las reglas de tipificación correspondientes son,

$$\frac{}{\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}}\ (RVar) \qquad \frac{\Gamma, x : \mathsf{T} \vdash M : \mathsf{S}}{\Gamma \vdash \lambda x.M : \mathsf{T} \to \mathsf{S}}\ (RFun)$$

$$\frac{\Gamma \vdash M_1 : \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash M_2 : \mathsf{T}}{\Gamma \vdash M_1 M_2 : \mathsf{S}}\ (RApp)$$

$$\frac{\Gamma \vdash M : \neg\neg\mathsf{T}}{\Gamma \vdash \mathcal{C}M : \mathsf{T}}\ (R\mathcal{C}) \qquad \frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathcal{A}M : \mathsf{T}}\ (R\mathcal{A})$$

Suponiendo que V es un valor, es decir, una variable o abstracción lambda,

entonces el *sistema de reescritura* $\mathcal{C}$ es el siguiente,

$$E[(\lambda x.M)V] \mapsto_{\mathcal{C}} E[M[x := V]] \quad (\mathcal{C}1)$$
$$E[\mathcal{C}M] \mapsto_{\mathcal{C}} M(\lambda z.\mathcal{A}(E[z])) \quad (\mathcal{C}2)$$
$$E[\mathcal{A}M] \mapsto_{\mathcal{C}} M \quad (\mathcal{C}3)$$

donde E es un contexto de evaluación y su definición está dada por la gramática $E ::= [\,] \mid VE \mid EM$, de tal suerte que para $E[M]$ se tiene,

$$\begin{array}{lcl} [\,][M] & = & M \\ (EN)[M] & = & (E[M])N \\ (VE)[M] & = & V(E[M]) \end{array}$$

A.2. Cálculo $\lambda\mu$

El cálculo $\lambda\mu$ se define, en un estilo *a la Church*, a partir de la siguiente gramática,

$$M ::= x \mid MM \mid \lambda x : \mathsf{T}.\, M \mid [\alpha]M \mid \mu\alpha : \neg\mathsf{T}.\, M$$

El conjunto de reglas de tipificación para los términos del cálculo $\lambda\mu$ es,

$$\frac{}{\Gamma, x : \mathsf{T} \vdash x : \mathsf{T}}(RVar)$$

$$\frac{\Gamma, x : \mathsf{T} \vdash M : \mathsf{S}}{\Gamma \vdash \lambda x : \mathsf{T}.\, M : \mathsf{T} \to \mathsf{S}}(RFun) \qquad \frac{\Gamma \vdash M : \mathsf{T} \to \mathsf{S} \quad \Gamma \vdash N : \mathsf{T}}{\Gamma \vdash MN : \mathsf{S}}(RApp)$$

$$\frac{\Gamma, \alpha : \neg\mathsf{T} \vdash M : \bot}{\Gamma \vdash \mu\alpha : \neg\mathsf{T}.\, M : \mathsf{T}}(\mu RFun) \qquad \frac{\Gamma, \alpha : \neg\mathsf{T} \vdash M : \mathsf{T}}{\Gamma, \alpha : \neg\mathsf{T} \vdash [\alpha]M : \bot}(\mu RApp)$$

Para dar las reglas de reducción necesitamos de los contextos de evaluación $\mathcal{E}$, que se definen en el cálculo $\lambda\mu$ como $\mathcal{E} ::= [\,] \mid \mathcal{E}M \mid [\alpha]\mathcal{E}$, con $\mathcal{E}[M]$ obedeciendo las siguientes reglas,

$$\begin{array}{c} [\,][M] = M \\ (\mathcal{E}N)[M] = (\mathcal{E}[M])N \\ ([\alpha]\mathcal{E})[M] = [\alpha]\mathcal{E}[M] \end{array}$$

También requerimos especificar el conjunto de variables libres de $\mathcal{E}$, denotado con $FV(\mathcal{E})$, cuya definición es,

$$\begin{aligned} FV([\,]) &= \varnothing \\ FV(\mathcal{E}M) &= FV(\mathcal{E}) \cup FV(M) \\ FV([\alpha]\mathcal{E}) &= \{\alpha\} \cup FV(\mathcal{E}) \end{aligned}$$

Con esto, la reducción $\to_\mu$ se forma al unir las reducciones $\to_\beta$, $\to_{\eta_\mu}$, $\to_{\beta_\mu}$ y $\to_\varsigma$, las cuales se muestran a continuación,

$$\begin{aligned} (\lambda x : \mathsf{T}.M)N &\to_\beta M[x := N] \\ \mu\alpha : \neg\mathsf{T}.[\alpha]M &\to_{\eta_\mu} M \text{ si } \alpha \notin FV(M) \\ [\beta](\mu\alpha : \neg\mathsf{T}.M) &\to_{\beta_\mu} M[\alpha := [\beta][\,]] \\ (\mu\alpha : \neg(\mathsf{T} \to \mathsf{S}).M)N &\to_\varsigma \mu\beta : \neg\mathsf{S}.M[\alpha := [\beta]([\,]N)] \\ \text{para } \to_\varsigma \text{ se requiere } \alpha \neq \beta &\text{ y } \beta \notin FV(MN) \end{aligned}$$

donde la sustitución $M[\alpha := \mathcal{E}]$ toma el contexto de evaluación $\mathcal{E}$ y lo coloca en todos los lugares donde α figura libre en M. Formalmente se tiene que si $y, \beta \notin FV(\mathcal{E})$ y $\beta \neq \alpha$ entonces,

$$\begin{aligned} x[\alpha := \mathcal{E}] &= x \\ (\lambda y : \mathsf{T}.M)[\alpha := \mathcal{E}] &= \lambda y : \mathsf{T}.(M[\alpha := \mathcal{E}]) \\ (M_1 M_2)[\alpha := \mathcal{E}] &= (M_1[\alpha := \mathcal{E}])(M_2[\alpha := \mathcal{E}]) \\ (\mu\beta : \neg\mathsf{T}.M)[\alpha := \mathcal{E}] &= \mu\beta : \neg\mathsf{T}.(M[\alpha := \mathcal{E}]) \\ ([\alpha]M)[\alpha := \mathcal{E}] &= \mathcal{E}[M[\alpha := \mathcal{E}]] \\ ([\beta]M)[\alpha := \mathcal{E}] &= [\beta](M[\alpha := \mathcal{E}]) \end{aligned}$$

Bibliografía

[1] Zena M. Ariola and Hugo Herbelin. *Minimal Classical Logic and Control Operators*, Thirtieth International Colloquium on Automata, Languages and Programming, volume 2719 of *Lecture Notes in Computer Science*, pp. 871-885. Springer-Verlag. 2003.

[2] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2nd revised edition. 1984.

[3] Torben Braüner. *Introduction to Linear Logic*. Basic Research in Computer Science (BRICS) Lectures Series. 1996. URL = `http://www.brics.dk/LS/96/6/BRICS-LS-96-6.pdf`. Consultado: el 4 de abril de 2011.

[4] Haskell Curry. *Functionality in Combinatory Logic*. Proceedings of the National Academy of Sciences, Vol. 20, pp. 584-590. 1934.

[5] Haskell Curry, Robert Feys, William Craig. *Combinatory Logic, Vol. I*. Amsterdam: North-Holland, with two sections by William Craig, see paragraph 9E. 1958.

[6] M. Felleisen, D.P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205-237. 1987.

[7] M. Felleisen and R. Hieb. The revised report on the syntactic theory of sequential control and state. *Theoretical Computer Science*, 103:235-271. 1992.

[8] Gilda Ferreira and Paulo Oliva. *On Various Negative Translations*, Poceedings of CL&C2010, Electronic Proceedings in Theoretical Computer Science, Vol. 47, pp. 21-33. 2011.

[9] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press. 1989.

[10] T.G. Griffin. *A formulae-as-types Notion of Control*, Conference Record of the Annual ACM Symposium on Principles of Programming Languages, pp. 47-58. ACM Press. 1990.

[11] Philippe de Groote. *On the relation between the lambda-mu calculus and the syntactic theory of sequential control*. In LPAR, volume 822 of *Lecture Notes of Artificial Intelligence*, pp. 31-43. Springer-Verlag. 1994.

[12] John Harrison. *Lecture notes for the course: Introduction to Functional Programming*, University of Cambridge. 1997. URL = `http://www.cl.cam.ac.uk/teaching/Lectures/funprog-jrh-1996/`. Consultado: el 9 de mayo de 2011.

[13] Hugo Herbelin. *Introductive talk on call-by-name delimited continuations*. Selected Talks. July 2007. URL = `http://yquem.inria.fr/~herbelin/talks/index-eng.html`. Consultado el: 8 de noviembre de 2011.

[14] Hugo Herbelin and Alexis Saurin. *$\lambda\mu$-calculus and $\Lambda\mu$-calculus: a Capital Difference*. Elsevier. 2010. URL = `http://www.pauillac.inria.fr/ herbelin/publis/apal-HerSau10-lambda-mu-Lambda-mu.pdf`. Consultado el: 10 de noviembre de 2011.

[15] William Howard. *The formulae-as-types notion of construction [original paper manuscript from 1969]*, in Seldin, Jonathan P.; Hindley, J. Roger, To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Boston, MA: Academic Press, pp. 479-490. 1980.

[16] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press. 1986.

[17] Georg P. Loczewski. *Lambda Calculus & A++. Basic Concepts*. 2003. URL = `http://www.lambda-bound.com/book/lambdacalc/`. Consultado el: 10 de febrero de 2012.

[18] Favio E. Miranda Perea. *Lógica proposicional de segundo orden*. Aportaciones Matemáticas, Memorias 40, Sociedad Matemática Mexicana. 2009.

[19] Favio E. Miranda Perea. *Notas de clase para el curso de Análisis Lógico*, 2011. URL = `http://www.matematicas.unam.mx/favio/cursos.html`. Consultado el: 28 de abril de 2011.

[20] Favio E. Miranda Perea. *Notas de clase para el curso de Lenguajes de Programación y sus Paradigmas*, 2010. URL = `http://www.matematicas.unam.mx/favio/cursos.html`. Consultado el: 28 de abril de 2011.

[21] Favio E. Miranda Perea, Lourdes del Carmen González Huesca, Araceli Liliana Reyes Cabello. *Una semántica constructiva para la lógica de segundo orden AF2*. Enviado a las Memorias de la Sociedad Matemática Mexicana. Enero 2012.

[22] John C. Mitchell. *Concepts in Programming Languages*. Cambridge University Press. 2003.

[23] Joan Moschovakis. *Intuitionistic Logic*, The Stanford Encyclopedia of Philosophy, Edward N. Zalta (ed.), Summer 2010 Edition. URL = `http://plato.stanford.edu/archives/sum2010/entries/logic-intuitionistic/`. Consultado el: 28 de abril de 2011.

[24] Eduardo G. Pacheco Gómez. *El isomorfismo de Curry-Howard, un fundamento lógico para la programación funcional*, Tesis de Licenciatura. UNAM, Facultad de Ciencias. 2008.

[25] M. Parigot. *$\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction*, Pro. International Conference on Logic Programming and Automated Reasoning, volume 624 of *Lecture Notes in Artificial Intelligence*, pp. 190-201. Spring-Verlag. 1992.

[26] F.J. Pelletier. A Brief History of Natural Deduction, *History and Philosophy of Logic*, Vol. 20, pp. 1-31. 1999.

[27] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press. 2002.

[28] Raúl Rojas. *A Tutorial Introduction to the Lambda Calculus*, Freie Universität Berlin, WS-97/98. URL=`http://www.inf.fu-berlin.de/lehre/WS03/alpi/lambda.pdf`. Consultado el 28 de marzo de 2011.

[29] Marcus du Sautoy. *A brief history of mathematics*, 2010. BBC Podcasts, URL = `http://www.bbc.co.uk/podcasts/series/maths`. Consultado el: 14 de julio de 2011.

[30] Gert Smolka and Chad E. Brown. *Introduction to Computational Logic*, Lecture Notes, Saarland University. July 2008. URL = `http://www.ps.uni-sb.de/courses/cl-ss08/script/icl.pdf`. Consultado el: 20 de julio de 2011.

[31] M.H.B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, Studies in Logic and the Foundations of Mathematics, Vol. 149, Elsevier. 2006.

[32] Th. Streicher, B. Reus. *Classical Logic, Continuation Semantics and Abstract Machines*. Journal of Functional Programming. Cambridge University Press, 8:543-572. 1998.

[33] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics. An Introduction*, volume 121 of Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam. 1988.

[34] Philip Wadler. *Proofs are Programs: 19th Century Logic and 21st Century Computing*, Avaya Labs. 2000. URL = `http://homepages.inf.ed.ac.uk/wadler/papers/frege/frege.pdf`. Consultado el 27 de octubre de 2011.

[35] Anita Wasilewska. *Lecture notes for the logic course*, Department of Computer Science. Stony Brook University. 2007. URL = `http://www.cs.sunysb.edu/~cse371/`. Consultado el: 11 de abril de 2011.