

¡The Matrix!

Apuntes sobre cálculo lambda

**Jorge A. Peri
Jorge Scucimarri**

**Universidad Nacional de Luján
Departamento de Ciencias Básicas**

**Edita: Fundación Encuentro
Edición previa
Luján, Bs. As.
2001**

Agradecemos a Silvia Alonso la revisión del original

Esta obra es propiedad intelectual de los autores. Los derechos de publicación en lengua española han sido cedidos a la Fundación Encuentro para esta edición. Prohibida su reproducción parcial o total por cualquier medio, sin permiso por escrito de Fundación Encuentro.

Edición previa, 2001

El genio del grupo

Siempre intuimos que tarde o temprano Carlos iba a meternos en líos. Su personalidad era un cóctail explosivo: una inteligencia sobrenatural, unida a una voluntad de granito y a una curiosidad imposible de satisfacer. El no entraba en la famosa dicotomía entre intelectuales y hacedores, él era las dos cosas a la vez, ¡y en su máxima expresión!. Nunca le conocimos límites, no tenemos dudas de que si le hubiera interesado el dinero hubiera sido el hombre más rico del mundo. Pero no, su ambición era saber, y saber todo.

Lo conocemos desde muy chico, fuimos juntos al jardín de infantes, y desde allí fuimos amigos durante toda nuestra vida. Siempre nos sorprendía con su inteligencia, siempre se adelantaba a los temas, siempre sabía las respuestas, fácilmente podría haber adelantado sus estudios y haberse recibido mucho antes, pero decía que prefería seguir nuestro ritmo para no separarse de nosotros. A los dos años, su madre, doña Inés, se extrañaba de verlo a menudo absorto contemplando los envases de los comestibles, ¡más adelante supo que en realidad estaba leyendo!.

A los tres años había desarrollado una habilidad manual sorprendente, más o menos por esa época construyó una estación meteorológica completa siguiendo las instrucciones que había encontrado en un libro de la UNESCO, y utilizando frasquitos, globos de goma y otros elementos comunes. Recuerdo que para el higrómetro necesitaba un cabello que fuera lo más largo posible, la víctima fué Ana, una rubieza con el pelo hasta la cintura. En un operativo comando, nosotros la entretuvimos, mientras él se alzaba con el preciado botín de cuatro o cinco cabelllos que logró cortarle con una tijera. Aún hoy, en un rincón del laboratorio de Carlos el higrómetro sigue marcando la humedad relativa como hace cuarenta años atrás. Así fué esta parte de su vida, a los cinco años nos sorprendió con su primer radio a galena y su primer teléfono, hechos con unos trozos de hojalata, cables y pedazos de carbón, y a los siete construyó su primer transmisor de radio.

Durante la adolescencia, su personalidad cambió radicalmente. De pronto dejó de interesarse por las cuestiones tecnológicas para dedicarse a la lectura de libros de filosofía y religión, a los quince ya había leído la Biblia, los Vedas, el Corán y otros libros sagrados cuyos nombres ya no recuerdo. Comenzó a estudiar varias carreras universitarias: física, matemáticas, informática, etc. pero no terminó ninguna porque perdía el interés.

En su juventud se obsesionó por la naturaleza de la materia y de la energía: "la materia es energía concentrada" nos decía, "de la misma forma que la energía es información concentrada", y se iba dejándonos absortos. Treinta años después entenderíamos el significado de esta última frase.

Ya de adulto, comenzó a desarrollar una vida más normal, yo diría casi mediocre, con un trabajo normal, una mujer normal y una casa normal. Lo único distinto en su vida era aquél galpón en el fondo de su casa, que él llamaba "el laboratorio", que para mi gusto de laboratorio tenía poco: unas cuantas computadoras en red, algunos dispositivos parecidos a los de realidad virtual y algunos pocos libros. Fué en ese galpón donde transcurrieron los hechos asombrosos que vamos a relatar.

La virtualidad real

Aquel día Carlos estaba eufórico.

- Los invito a vivir una experiencia con mi nuevo sistema de virtualidad real, dijo con una sonrisa de triunfo.
- Querrás decir de realidad virtual, lo corregí.
- No, a éste yo lo bauticé con el nombre de "virtualidad real".

- ¿Y cuál es la diferencia? pregunté.

- Muy sencillo, dijo Carlos, en la realidad virtual las cosas que ocurren parecen reales, pero son virtuales. En mi sistema es exactamente al revés, las cosas parecen virtuales pero son reales.

Acepté la explicación casi automáticamente, sin darme cuenta que en ella estaba la clave de la increíble experiencia que estaba por comenzar a vivir.

La cita fué al otro día, en el laboratorio. Estaba la red de computadoras de siempre y cuatro camas, arriba de los cuales había cascos conectados a la red. Al lado de la cama de Carlos había un monitor y un teclado. La explicación fué sencilla:

- Solo tienen que acostarse en las camas y ponerse los cascos, yo haré el resto. Van a sentir una cierta somnolencia, déjense llevar por el sueño.

Así fué, muy lentamente, me invadiendo un cierto sopor. ¿Me pareció, o en la pared del costado recién había un cuadro?. La sensación de sueño fue aumentando gradualmente, miré a mi alrededor y noté algo extraño, era como si en la habitación "faltaran cosas", ¿no había un monitor al costado de la cama de Carlos?, ¿y la lámpara? ¿dónde está la lámpara?. La última vez que abrí los ojos, ya al borde del sueño total, no me quedaron dudas, ¡había desaparecido la mayoría de los objetos de la habitación!. Me quedé profundamente dormido.

¡Un mundo increíble!

- ¡Despierten! la voz sin sonido de Carlos retumbó dentro mío.

- ¡Bienvenidos al universo real!

- ¿Dónde estamos?, pregunté mentalmente.

- No es fácil contestar a esa pregunta, respondió Carlos, estamos en ninguna parte, porque aquí no hay espacio. Tampoco hay tiempo, por eso no podemos contestar ni cuando ni donde está ocurriendo esto. Aquí no hay nada de lo que conocemos, ni planetas, ni estrellas, ni materia, ni energía. Tampoco hay sonidos, nosotros no estamos hablando, simplemente pensamos. Todo lo que crean "ver" aquí, en realidad será percibido por ustedes a través de mi sistema de virtualidad real, que quedó funcionando allá en el mundo.

- Espero que no se corte la energía eléctrica, pensé.

- No hay riesgos, pensó Carlos, allá en la tierra tenemos UPS's con suficientes baterías de respaldo.

Me dediqué a tratar de percibir el entorno. La oscuridad era total, el silencio, absoluto, todo era una quietud exasperante. Sin embargo "algo" estaba percibiendo.

- Aquí no es necesaria la luz, ni el sonido, ni el movimiento, pensó Carlos, aquí la información reside en estado puro. No necesita ni códigos ni ondas. Incluso la componente temporal en nuestro "diálogo" es virtual, ¡no podríamos dialogar sin tiempo!. De la misma forma, todas las sensaciones que perciben parecidas a las que percibían en el mundo son simuladas por mi sistema de virtualidad real. Nuestro cerebro está adecuado para pensar en términos de espacio y tiempo, y no podría soportar la percepción directa de este mundo, tal vez enloqueceríamos instantáneamente. Por eso bauticé a mi sistema como de "virtualidad real", es porque toma conocimiento e información de este mundo real, y nos lo presenta como si fuera virtual, esto es exactamente al revés que un sistema de realidad virtual.

- Algo percibo, pensé, ¡pero no alcanzo a darme cuenta qué es!.

- Serena tus pensamientos y concéntrate más, pensó Carlos.

¿Máquinas?:

De pronto las percibí, allí estaban, fantásticas, increíbles, grandiosas, imponentes.

- ¡¿Qué son esas cosas?!, pensé entre asombrado y curioso.

- Conocimiento en estado puro, pensó Carlos, sin códigos, ni cerebros, ni procesadores, solo conocimiento. Y eso es todo lo que existe en este universo.

- Pero parecen...

- ¿Máquinas?, preguntó Carlos percibiendo mis sensaciones. Tal vez podríamos imaginarlas mentalmente como máquinas, pero recuerda que tus sensaciones, como las mías, son recreadas por el sistema de virtualidad real, él siempre trata de presentarnos las cosas utilizando análogos de este universo real en nuestro mundo virtual.

- ¡¿Cómo virtual!?

- Ya hablaremos de eso, pensó Carlos. Con respecto a estas cosas, si te gusta nos referiremos a ellas como máquinas. Mira eso, pensó señalando con un dedo inexistente hacia un lugar inexistente. Por supuesto que vi sin mirar.

Lo que percibí está mas allá de cualquier experiencia cotidiana, aquí no hay nada que se asemeje en algo a nuestro mundo. por esta razón mis descripciones serán necesariamente confusas e imprecisas. ¿Dónde queda esto?, ¿Cuándo ocurre?, imposible saberlo. Tal vez para contestar dónde, debiéramos decir algo así como "fuera del espacio", y para contestar cuándo, deberíamos decir "antes del tiempo", pero, ¿qué es "fuera" donde no hay espacio? o ¿qué es "antes" donde no hay tiempo?. Tal vez las expresiones mas adecuadas sean "antes del espacio" y "fuera del tiempo", a nosotros nos gustó más decir "debajo del tiempo", pero sólo por razones estéticas.

Vi a las prodigiosas máquinas trabajar permanentemente, máquinas muy diminutas, infinitesimalmente elementales. Las percibí como máquinas porque tienen algo así como una entrada y una salida, pareciera que por la entrada reciben algo que usan como materia prima, y por la salida emiten algo como producto, y su existencia consiste en estar permanentemente produciendo.

- ¿Ven?, preguntó Carlos, en la oscuridad total, en el silencio mas absoluto, en el medio de una quietud exasperante, nuestras máquinas trabajan sin parar, tal vez tengan una tarea muy importante que cumplir... Estas máquinas son todo lo que existe en este universo real.

El mundo de las máquinas

- Ahora, si en este mundo lo único que hay son máquinas, ¿que procesan?, pregunté

- ;Por supuesto que procesan a otras máquinas!, contestó Carlos. Cada máquina lo que hace es fabricar máquinas a partir de otras máquinas. Para ello, acepta una máquina como materia prima, y despacha otra máquina como producto. Ahora bien, es bueno que recuerdes que en matemática existe el concepto de "función". Esta palabra tiene dos interpretaciones: por un lado se puede ver a una función como un tipo especial de relación, y como tal es un conjunto de pares ordenados.

- Hasta aquí es lo que yo conozco, pensé.

- Pero este concepto también tiene una interpretación procedural, una función puede ser pensada como una "caja negra" que devuelve un valor cada vez que se le da un dato como argumento. En este sentido nuestras máquinas pueden ser vistas como funciones, cuyos argumentos son otras funciones!

- Pero, ¿Quién las fabricó?, ¿Para qué hacen todo esto?

- La primera pregunta decididamente no te la puedo responder porque no tengo la menor idea. Respecto de la segunda, tengo una somera percepción, pero tal vez nos aproximemos más a la respuesta todos juntos si las estudiamos ahora que estamos entre ellas.

Percibimos que el comportamiento parecía ser general, podemos describirlo diciendo que cuando una máquina A procesa a otra máquina B, el producto es una tercera máquina C.

- Para resumir: una máquina está compuesta por máquinas, y fabrica máquinas a partir de otras máquinas, pensó Carlos.

La única forma posible que encuentro para describir este universo es mediante símbolos. Traduciéndola a nuestro lenguaje, representaremos a una máquina como una secuencia de caracteres como puede ser:

fa.fb.a b fa.b

o cualquier otra parecida, siempre que la secuencia haya sido construida con ciertas reglas de formación que veremos más adelante. Cada uno de los caracteres se corresponde con una "parte" de la máquina. En esta traducción, la secuencia de caracteres tiene tres interpretaciones:

- es el nombre de la máquina.
- es una descripción de lo que la máquina hace.
- es la máquina en sí misma.

- En este universo extravagante, en el cuál lo único que existe son máquinas, estas tres interpretaciones se confunden en una sola: una máquina es una secuencia de caracteres que es el nombre de la máquina y la descripción de lo que ella hace, pensó Carlos.

Como una máquina está compuesta por otras máquinas, la secuencia de caracteres está compuesta por sub-secuencias correspondientes a las máquinas componentes.

- En general, cuando una máquina acepta a otra para "procesar" lo que hace es introducir el código de la máquina entrante entre su propio código, mezclando a ambos para obtener un código nuevo, algo parecido a lo que hacemos los seres vivos con el código genético, nos aclaró Carlos.

Abstracciones

"Cuando tus oídos se hayan cerrado para todos los sonidos del mundo, entonces, y solo entonces, podrás escuchar la voz del silencio"

La Voz del Silencio

Decidimos utilizar algunas convenciones para ver claramente como ocurren estos procesos. Para nosotros, una máquina es una expresión de la forma:

{<letra>.<expresión>

por ejemplo:

fb.abc

donde, en este caso:

- a) la letra griega f indica que se trata de una máquina.
- b) la letra "b" representa a la entrada de la máquina.
- c) la expresión "abc" corresponde a la salida, también indica qué es lo que hace la máquina con la entrada para obtener la salida.
- d) el punto separa los símbolos correspondientes a la entrada de los correspondientes a la salida.

Acordamos en denominar "abstracción" a la descripción de la máquina, y a la expresión "abc" la llamamos "cuerpo" de la abstracción.

Como segundo ejemplo, podemos imaginar una máquina un poco más compleja:

fa.fb.abc

o también:

fa.(fb.abc)

en este caso la máquina "fb.abc" es componente de la máquina "fa.fb.abc".

Aplicaciones

Cuando una máquina procesa a otra lo indicaremos simplemente colocando una máquina al lado de la otra entre paréntesis:

(<máquina1> <máquina2>)

donde estamos indicando que la <máquina1> está por procesar a la <máquina2>, por ejemplo:

(fa.fb.a b fx.x) *Estos parentesis no se usan*

aquí los paréntesis no son estrictamente necesarios y se usan solamente para mejorar la legibilidad. A esta expresión la denominamos "aplicación". De esta manera, para nosotros una expresión "M", que llamaremos "término", puede referirse a una de tres cosas, o es una letra suelta como "x", que representa a una máquina (una variable), o es una abstracción del la forma "fa.N", o es una aplicación de la forma "(M1 M2)", donde N, M1 y M2 a su vez pueden ser variables, abstracciones o aplicaciones. Formalmente, podemos describir a esta gramática como sigue:

M ::= x | (M1 M2) | (fx.N)

Usamos letras minúsculas para nombrar a las variables, y mayúsculas para nombrar a los términos, pero siempre recordando que la diferencia entre variable y término es sólo gramatical, ya que ambas se refieren a máquinas. La gramática descripta anteriormente define con exactitud qué expresiones se refieren a máquinas y cuáles no. Llamaremos "bien formadas" a las expresiones formadas de acuerdo con estas reglas de formación.

Algunas observaciones:

a) En nuestra convención, en ausencia de paréntesis, supondremos que las aplicaciones son asociativas a la izquierda, de forma que una expresión como:

M1 M2 M3

debe ser considerada como:

((M1 M2) M3)

esto es, la máquina M1 está por procesar a M2, y la máquina (M1 M2) está por procesar a M3.

b) En el cuerpo de una abstracción, y en ausencia de paréntesis, los símbolos se asocian a la derecha, tan lejos como sea posible, una expresión como:

fv.M1 M2 M3

la consideraremos como:

fv.(M1 M2 M3)

c) La aplicación tiene precedencia frente a la abstracción, con lo cual una expresión como:

fx.xy

la interpretaremos como:

fx.(x y)

d) Las abstracciones anidadas pueden ser abreviadas removiendo las ocurrencias extra de los símbolos f, por ejemplo la expresión:

fx.fy.fz.M

puede ser escrita como:

Se desal conseja

fixyz.M

En las expresiones que no sean evidentes, nosotros trataremos de evitar estas simplificaciones para garantizar la claridad, aunque ello implique escribir con muchos paréntesis.

- Deberemos entrenarnos para entender mejor este inaudito mundo de las máquinas, pensó Carlos: Así que les preparé algunos ejercicios para ejercitarse la mente.

Ejercicios:

En las siguientes expresiones, identificar todas las abstracciones y aplicaciones que hay. Completar los paréntesis cuando convenga para aumentar la legibilidad. Las abstracciones y aplicaciones pueden indicarse como mostramos en el siguiente ejemplo:

fa.ab

completamos los paréntesis y marcamos donde comienza y donde termina cada aplicación y abstracción

(fa.(a b))
<---> aplicación

<-----> abstracción

Ejercicio 1:

(fa.a(x y))

Ejercicio 2:

a (fa.a b) a

Ejercicio 3:

((fx.(x y)) (fz.z))

Ejercicio 4:

(fx.xxx) (fx.xxx)

Ejercicio 5:

(fx.xx) a (fx.xx)

Ejercicio 6:

((fx.((fy.(xy))x)) (fz.w))

Ejercicio 7: De acuerdo con las reglas anteriores, y dada la siguiente expresión:

fx.M1 fy.M2 M3 M4

¿cuál de las siguientes expresiones con paréntesis es equivalente?:

- a) (fx. (M1 (fy. ((M2 M3) M4))))
- b) ((fx. (M1)) (fy. ((M2 M3) M4)))
- c) (((fx.M1) (fx.M2)) M3) M4}

VARIABLES LIBRES Y LIGADAS:

En nuestra descripción de las máquinas mediante símbolos, en una expresión como la siguiente:

fx.fb.axyb

distinguimos dos tipos de variables: por un lado "x" y "b", que aparecen indicando las entradas de las máquinas, y por otro lado "a" y "y" que aparecen sólo en el cuerpo de la abstracción; llamaremos "ligadas" a las primeras, y "libres" a las segundas. Veremos que su significado es radicalmente distinto. Llamaremos "combinadores" a las máquinas que no tengan variables libres.

Ejercicios:

Indicar cuales de las siguientes expresiones son combinadores, y cuáles son las variables libres y las ligadas en cada expresión:

Ejercicio 8: $fx.fy.fz.abxyzc$

Ejercicio 9: $fx.fy.(axb)$

Ejercicio 10: $(fx.b(xy))$

Ejercicio 11: $(fx.ax(fa.xb))$

Ejercicio 12: $((fx.bax) (fb.by))$

Ejercicio 13: $((fx.x) (fa.fb.ab))$

Ejercicio 14: $((fa.fb.fc.b(abc)) (ff.fx.x))$

β -reducción:

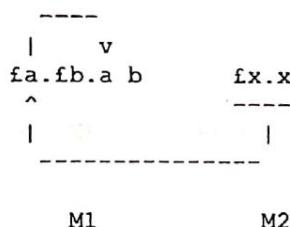
- Observen, pensó Carlos: una máquina M1 procesa a otra máquina M2 incorporando el código de M2 y combinándolo con su propio código para producir el de la máquina nueva. En esta suerte de fagocitosis M1 y M2 desaparecen, para transformarse en una nueva máquina M3.

A este proceso lo llamaremos β -reducción. Volviendo al ejemplo anterior

$((fa.fb.ab) (fx.x))$

M1 M2

podemos imaginar a M2 "aproximándose" por la derecha a M1. En M1, la letra "a" que se encuentra en la expresión "fa." representa la entrada. Cuando M2 pasa por la entrada de M1 (a), su código "fx.x" va a reemplazar a todas las apariciones de "a" en el cuerpo de M1:

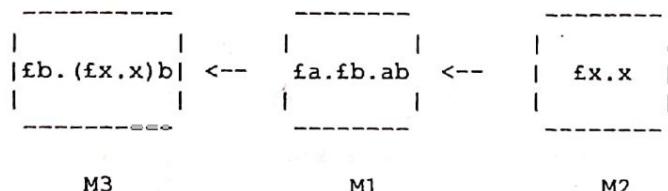


aquí hemos indicado con flechas el "recorrido" de M2 fuera y dentro de M1, el resultado de este proceso es una nueva máquina M3, que combina los códigos de M1 y M2:

$\rightarrow fb.(fx.x) b$

M3

donde la flecha indica que la máquina "fb.(fx.x) b" es la que resulta del proceso, gráficamente:



Aquí conviene hacer algunas observaciones:

a) la única utilidad de la variable "a" en la abstracción de M1, es indicar en qué lugar del cuerpo se debe insertar el código de la máquina entrante, y en realidad eso ocurre con las variables ligadas en todas las máquinas, ¡y los combinadores son sólo eso: referencias al orden en que son concatenadas otras máquinas!.

b) Como los nombres de las variables ligadas son irrelevantes, las siguientes expresiones son totalmente equivalentes:

$fx.fy.xy$ $fa.fb.ab$ $fm.bn.mn$

c) Los nombres de las variables libres no son irrelevantes, las expresiones " $fa.ab$ " y " $fa.ac$ " no son equivalentes.

d) en este ejemplo, M3 es una abstracción que en su cuerpo contiene otra aplicación, en la cual la máquina " $(fx.x)$ " está por procesar a la máquina "b". Poniendo todos los paréntesis, M3 sería:

$fb.((fx.x) b)$

e) Puede ocurrir que en el cuerpo de la abstracción de la máquina que está procesando no haya ninguna ocurrencia de la variable que representa la entrada, como en el ejemplo siguiente:

$fa.fb.bb$ $fx.x$

en este caso no se produce reemplazo alguno, aunque la reducción se hace de todos modos, la máquina resultante es:

--> $fb.bb$

volveremos a esto cuando veamos la máquina constante.

f) En M3 los paréntesis no son irrelevantes. Si se omiten, la expresión resultante se podría haber escrito:

$fb.fx.xb$

que de acuerdo con las reglas anteriores se interpreta como:

$fb.(fx.(xb))$

g) Las variables correspondientes a distintas máquinas representan cosas distintas aunque tengan el mismo nombre. Esto puede conducir a errores en los procesos de reducción, por eso es aconsejable usar siempre letras distintas para distinguir a las variables de la máquina procesadora, de las de la máquina procesada.

h) En la expresión de la aplicación, vemos la máquina procesadora M1 a la izquierda y la máquina procesada M2 a la derecha, asumiremos por convención que siempre la máquina de la derecha es la entrada de la de la izquierda, es decir, ésta última procesa a la primera.

Ejercicio 15: Efectuar la β -reducción indicada en la observación d)

Ejercicio 16: Encontrar la máquina que produce M2 procesando a M1.

ESTRUCTURAS
MIG
AC

Alfa-reducción:

Como vimos anteriormente, las expresiones " $fa.aa$ " y " $fb.bb$ " son equivalentes, esto quiere decir que en una expresión cualquiera, las variables ligadas pueden ser reemplazadas por otras, por ejemplo en:

$fa.fx.fc.axdcc$

si reemplazamos la "c" por una "m" obtenemos otra expresión equivalente a la anterior:

$fa.fx.fm.axdmm$

a este proceso lo denominaremos alfa-reducción, recordando que no pueden renombrarse las variables libres.

Cuando se efectúa una alfa-reducción, se debe tener en cuenta que a raíz de ello una variable libre no se transforme en ligada, por ejemplo la siguiente sustitución es correcta:

$fa.fy.by$
--> $fa.fz.bz$ (se renombró "y" como "z")

en cambio la siguiente sustitución no lo es:

$fa.fy.by$
--> $fb.fy.by$ (se renombró "a" como "b")

Observemos que en la primer expresión la variable "b" es libre, mientras que en la segunda es ligada, se dice que "b" fué "capturada". En general una sustitución la indicaremos como $E[x/y]$; esta expresión denota el resultado de reemplazar todas las ocurrencias de "x" en "E" por "y", cuidando que ésta última no resulte capturada.

En una β -reducción, también una variable puede resultar capturada, por ejemplo la siguiente es una reducción efectuada correctamente:

$(fx.fy.xy fz.z)$
--> $fy.(fz.z y)$

en cambio la siguiente no es correcta:

$(fx.fy.xy fz.yz)$
--> $fy.(fz.yz y)$
|
-- esta "y" fué capturada

Para evitar esta captura, antes de la β -reducción se pueden renombrar la variable "y" en la máquina procesadora:

$(fx.fy.xy fz.yz)$
--> $(fx.fa.xa fz.yz)$
--> $fa.(fz.yz a)$

en esta última expresión "y" permanece libre.

Ejercicios:

Efectuar las siguientes alfa-reducciones:

Ejercicio 17: $((fx.(x y)) (fz.z)) [x/a]$

Ejercicio 18: $((fx.((fy.(xy)) x)) (fz.w)) [x/b]$

Como se vió en el ejemplo anterior, cuando se produce una reducción pueden resultar máquinas que admitan a su vez procesos de reducción posteriores. Cuando sobre una máquina no sea posible efectuar ningún otro proceso de reducción, diremos que está en su "forma normal".

Punto fijo:

Cuando una máquina M1 procesa a otra máquina M2, puede ocurrir que la máquina producida sea idéntica a M2. Consideremos el siguiente ejemplo:

```
((fx.((fy.y) x)) (fa.ab))
<-----> <---->
      M1           M2

--> ((fy.y) (fa.ab))
--> (fa.ab)
<---->
      M2
```

Cuando esto ocurre, diremos que M2 es un "punto fijo" de M1. Es importante destacar que para cada máquina M1 existe al menos una máquina M2, tal que M2 es un punto fijo de M1.

Ejercicios:

En las siguientes aplicaciones, investigar en qué casos M2 es un punto fijo de M1

Ejercicio 19: ((fa.fb.ab) (fx.xx))

Ejercicio 20: ((fa.(fb.b) a) (fx.x))

Ejercicio 21: ((fa.aa) (fx.fy.x))

Ejercicio 22: ((fa.fb.aa) (fx.fy.y))

Una clasificación para las máquinas:

Con el objeto de estudiarlas mejor, podemos intentar una clasificación de las máquinas. En principio se nos ocurre que podríamos clasificarlas por su complejidad, es decir por la cantidad de abstracciones y aplicaciones que contienen. Así tendremos máquinas con una abstracción, por ejemplo:

fx.x
fx.xy
fx.xyz
....

máquinas con dos abstracciones:

fx.fy.x
fx.fy.xy
fx.fy.xyz
.....

con tres:

fx.fy.fz.x
fx.fy.fz.xy
fx.fy.fz.xyz
....

y así sucesivamente. Por otra parte, y recordando que una expresión como "xyz" es una forma abreviada de escribir " $((x y) z)$ ", vemos que está representando dos aplicaciones. De esta forma, en cada uno de los ejemplos anteriores representamos máquinas sin ninguna aplicación (x), con una (xy) y con dos aplicaciones (xyz). Teniendo en cuenta esto, vemos que la siguiente es la representación de una máquina con cuatro abstracciones y dos aplicaciones:

$fa.fb.fc.fd.cda$

De aquí en adelante, y para simplificar la notación, identificaremos a las máquinas con nombres cortos de unas pocas letras; esto a modo de apodo o sobrenombre.

Algunas máquinas simples: la máquina I

Entre las máquinas mas simples están las que tienen una sola abstracción y no tienen aplicaciones, de ellas solo existen dos, la máquina I y la máquina constante. Una de estas máquinas simples es la que denominaremos máquina identidad (I), este nombre es debido a que la máquina que entrega a la salida es exactamente la misma que recibe como entrada. La expresión que define a la máquina identidad es la siguiente:

$fa.a$

la misma indica que la salida es la misma que la entrada, "a" en ambos casos. Procesando a una máquina cualquiera, por ejemplo:

—
| v
 $fa.a \quad fx.fy.x$
^ -----
.

--> $fx.fy.x$

La característica particular de I es que ella despide una máquina idéntica a la que recibe cualquiera que sea ésta, es decir, todas las otras máquinas son su punto fijo.

La máquina constante:

Existe otra familia de máquinas extremadamente simples, que son aquellas cuya salida no depende de la entrada. La más sencilla de estas máquinas, con una abstracción y ninguna aplicación, es la siguiente:

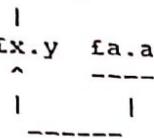
$fa.b$

la salida de esta máquina siempre será "b" indistintamente de la máquina que le ingrese. En general, tendrán esta respuesta todas aquellas máquinas en las cuales la letra que identifique a la entrada, no se encuentre en la expresión que identifica a la salida. Otros ejemplos de máquinas constantes son:

$fc.ab$	ab
$fc.fb.ab$	$fb.ab$
$fa.fb.b$	$fb.b$
etc..	

donde a la derecha se indicó cual es la salida para cualquier entrada. Veamos una máquina constante procesando a una máquina identidad:

-> ?



--> y

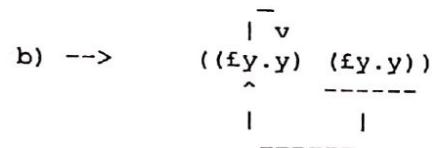
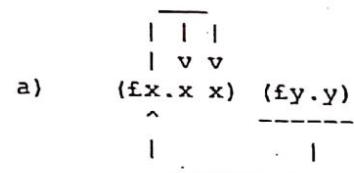
Una máquina insólita:

Esta máquina es un poco más interesante. Es el único combinador que existe con una abstracción y una aplicación, y por ser tan simple, su comportamiento es bastante curioso. Cuando recibe una máquina A, lo que hace es producir lo que hubiera producido esa máquina A si hubiera recibido como entrada a una máquina A!.

La abstracción de la máquina omega es:

$\lambda x. xx$

Veamos que produce la máquina omega cuando recibe como entrada a la máquina identidad:



la máquina identidad recibe I ...

c) $\rightarrow (\lambda y. y)$

... por lo tanto devuelve I

Se puede pensar en la máquina omega como una especie de máquina universal, que lo que requiere es una descripción de una máquina para transformarse a sí misma en esa máquina.

Nuevamente, observemos que la máquina omega desapareció en el proceso de producir su salida, o mas bien, se transformó en su respuesta. La pregunta inmediata es: ¿que produce la máquina omega cuando recibe como entrada a una máquina omega?, veamos:

$$\begin{array}{c}
 | \quad v \quad v \\
 (\text{fx}.x \ x) \quad (\text{fx}.xx) \\
 ^ \qquad \qquad \hline \\
 | \qquad | \\
 \hline
 \end{array}$$

--> ($\text{fx}.\text{xx}$) ($\text{fx}.\text{xx}$)
--> ($\text{fx}.\text{xx}$) ($\text{fx}.\text{xx}$)
--> ...

esto es, una secuencia infinita de máquinas omega procesando máquinas omega. Si la máquina omega siempre se transforma a sí misma en la máquina que le ingresa, y a su vez le ingresa la máquina omega, solo puede transformarse a sí misma en sí misma, y así sucesivamente. A una máquina que se encuentre en este estado paradógico y mágico se la llama una máquina Omega (con mayúscula):

((fa.aa) (fb.bb))

Como la máquina Omega nunca para de reducirse, se dice que no tiene una forma normal, también se dice que es irresoluble. A esta máquina se la identifica con todas las máquinas irresolubles.

Las máquinas con dos abstracciones:

Existe una pequeña familia de dos combinadores, que son un poco más complicados que las máquinas que vimos anteriormente, pero que juegan un papel muy importante en este extravagante universo de máquinas. Estas máquinas poseen dos abstracciones y ninguna aplicación, las llamaremos "K" y "KI".

La máquina K:

Esta máquina produce la máquina constante, la cual siempre devuelve la máquina que le ingresó a la máquina K. La misma está definida por:

$\text{fx}.\text{fy}.\text{x}$

como ejemplo veamos una máquina K procesando a una máquina omega:

$$\begin{array}{ccc}
 & \overline{\quad} & v \\
 & \text{fx}.\text{fy}.\text{x} & \text{fa.aa} \\
 ^ & \hline & \hline \\
 | & | & \\
 \hline
 \end{array}$$

--> $\text{fy}.\text{fa.aa}$

esta última es una máquina constante, que ante cualquier entrada devolverá la máquina omega, que es la entrada de la máquina K. Si la máquina K en lugar de una máquina omega hubiera recibido una máquina I:

(($\text{fx}.\text{fy}.\text{x}$) (fa.a))
--> $\text{fy}.\text{fa.a}$

esta máquina constante siempre devolverá la máquina identidad.

La máquina KI:

Es un miembro de la familia de máquinas constantes. Ésta en particular, para cualquier entrada, siempre produce la máquina identidad.

$fy.fx.x$

La máquina KI se produce cuando una máquina K procesa a una máquina identidad, en efecto, como vimos en el ejemplo anterior:

$$((fx.fy.x) (fx.z))$$

$$\rightarrow fy.fz.z$$

ésta es la máquina KI, que para cualquier entrada produce una máquina I.

Ejercicios:

Ejercicio 23: Investigar que produce una máquina K cuando procesa a otra máquina K.

Ejercicio 24: Investigar que produce una máquina omega cuando procesa a una máquina K.

Ejercicio 25: Investigar que produce una máquina K procesando a una Omega (con mayúsculas).

Ejercicio 26: Investigar que produce una máquina omega cuando procesa a una KI

Ejercicio 27: Investigar que produce una máquina K cuando procesa a una KI.

Simplificar las siguientes descripciones de máquinas hasta llevarlas a la forma normal si es que existe:

Ejercicio 28: $((fx.(x.y)) (fx.z))$

Ejercicio 29: $((fx.((fy.(x.y)) x)) (fx.w))$

Eta-reducción:

Además de las ya vistas reglas de alfa y β -reducción, existe otra regla que se conoce como eta-reducción y que responde a la idea de que, para cualquier expresión arbitraria E, las dos expresiones:

$$(fx.(E.x)) \text{ y } E$$

se refieren a la misma máquina cuando no existe la variable "x" libre en E. Ello se deduce fácilmente si consideramos que para cierta máquina A, la expresión

$$(fx.(E.x)) A$$

se reduce a

$$\rightarrow (E.A)$$

En general: $fx.E.x \rightarrow E$

La familia de dos abstracciones y una aplicación:

- Observen esta interesante familia de cuatro combinadores, pensó Carlos. Todas poseen dos abstracciones y una aplicación. Con estas características existen los siguientes casos posibles:

$\text{fx}.\text{fy}.xy$
 $\text{fx}.\text{fy}.yx$
 $\text{fx}.\text{fy}.xx$
 $\text{fx}.\text{fy}.yy$

- ¿Qué tienen de particular?, pensé.
- Veamos las máquinas una por una.

La máquina I*

También podemos denominarla "la máquina identidad una vez removida la identidad"

$\text{fa}.\text{fb}.ab$

Véamnos a la máquina I* procesando a una máquina omega:

$(\{\text{fa}.\text{fb}.ab) (\text{fx}.xx))$

- > $\text{fb}.(\text{fx}.xx b)$ β -reducción
--> $\text{fb}.bb$ β -reducción

es decir que I* se comporta como I, en este caso fb.b, una vez que se removió fa.a, de allí el nombre de "la máquina identidad una vez removida la identidad".

La máquina T:

También se la llama "la máquina identidad una vez removida la identidad cruzada"

$\text{fa}.\text{fb}.ba$

Supongamos una máquina T procesando a una máquina omega:

$(\text{fx}.\text{fy}.yx (\text{fz}.zz))$

- > $(\text{fy}.y (\text{fz}.zz))$
--> $\text{fz}.zz$

aquí se removió la identidad fx.x, la palabra "cruzada" es porque el orden de las variables en el cuerpo está invertido respecto de las entradas.

La máquina Kw:

Esta es otra máquina perteneciente a la familia de máquinas constantes, está definida por la expresión:

$\text{fa}.\text{fb}.bb$

esta máquina se caracteriza porque siempre produce a la salida la máquina omega, independientemente de cual sea la entrada. En efecto, para una máquina cualquiera M:

((fa.fb.bb) M)

--> fb.bb

recordemos que en la β -reducción reemplazamos todas las ocurrencias de la variable "a" por la máquina "M", y por supuesto que si no hay ninguna ocurrencia no se producen reemplazos, pero la reducción se efectúa de todos modos.

La máquina C(Kw) :

La máquina C(Kw) está definida por la expresión:

fa.fb.aa

Veamos a la máquina C(Kw) procesando a una máquina identidad.

--> (fa.fb.aa fx.x)
--> fb.(fx.x fx.x)
--> fb.fx.x

ésta es la máquina KI, que para cualquier entrada produce una máquina I. Recordemos que la máquina KI también se produce cuando una máquina K procesa a una máquina identidad.

Ejercicio 30: Investigar que produce una máquina I* cuando procesa a una I*.

Ejercicio 31: Investigar que produce una máquina C(Kw) cuando procesa a una I*.

Ejercicio 32: Investigar que produce una máquina Kw cuando procesa a una I*.

Ejercicio 33: Investigar que produce una máquina C(Kw) cuando procesa a una Kw.

Máquinas mas complejas: La máquina Kx:

Las máquinas pueden estar formadas por una cantidad arbitraria de abstracciones y aplicaciones. Un combinador interesante de tres abstracciones y ninguna aplicación es la máquina Kx, que cuando recibe una máquina M, responde con una máquina KI, es decir una máquina que responde con la máquina identidad para cualquier entrada. Su código es:

fx.fy.fz.z

Observemos qué produce una máquina Kx cuando procesa a una máquina K:

((fx.fy.fz.z) (fa.fb.a))
--> fy.fz.z

Al revés, observemos qué produce una máquina Kx cuando es procesada por una máquina K:

((fa.fb.a) (fx.fy.fz.z))
--> fb.fx.fy.fz.z

Esta es una máquina constante, que siempre devuelve una máquina constante (fx.fy.fz.z), que siempre devuelve una máquina constante (fy.fz.z), que siempre devuelve una máquina identidad (fz.z).

La máquina L

Está definida por:

$fa.fb.a(b\ b)$

el siguiente es un ejemplo de una máquina L procesando a una máquina KI:

$((fa.fb.a(bb))\ (fx.fy.y))$

--> $(fb.\ ((fx.fy.y)\ (b\ b)))$

--> $fb.fy.y$

en este caso, ésta es nuevamente una máquina KI.

La máquina O

La definimos como:

$fa.fb.b(a\ b)$

Veamos una máquina O procesando a una máquina I:

$((fa.fb.b(a\ b))\ (fx.x))$

--> $fb.b(fx.x\ b)$

--> $fb.bb$

Ejercicios

Analizar qué produce cada una de las siguientes máquinas cuando procesa a una máquina I:

Ejercicio 34: La máquina W

$fa.fb.(ab)b$

Ejercicio 35: La máquina C

$fa.fb.fc.(ac)b$

Ejercicio 36: La máquina B

$fa.fb.fc.a(bc)$

Ejercicio 37: La máquina S

$fa.fb.fc.ac(bc)$

Ejercicio 38: La máquina J

$fa.fb.fc.fd.ab(adc)$

Verdad y falsoedad:

- Aquí, debajo del tiempo, no hay otra cosa que máquinas, obviamente no hay ni números, ni operaciones, ni ninguno de los objetos matemáticos conocidos por nosotros, pensó Carlos. Particularmente, no existen las constantes lógicas como "V" y "F", ni los conectivos como la conjunción o la disyunción. Sin embargo todas esas cosas pueden ser creadas por nuestras máquinas, ¡y obviamente serán máquinas!.

- Observen esas dos máquinas, acotó Carlos, son las famosas "máquina-verdadero" y "máquina-falso".

- ¿Como?

- Como ven, en este mundo todas las cosas conocidas pueden construirse con máquinas, así que, sin mayores prolegómenos les presento las máquinas V y F:

$f_a.f_b.a$ máquina V

$f_a.f_b.b$ máquina F

con lo cual es posible por ejemplo definir la máquina AND, que no es otra cosa que el conectivo lógico correspondiente a la conjunción:

$f_x.f_y.x.y(f_a.f_b.b)$

de esta forma, un equivalente a la expresión lógica

V AND F

será una máquina AND que procesa consecutivamente a una máquina V y a una máquina F:

(AND V F)

$(f_x.f_y.x.y(f_a.f_b.b)) (f_c.f_d.c) (f_m.f_n.n)$

--> $(f_y.(f_c.f_d.c) y (f_a.f_b.b)) (f_m.f_n.n)$

--> $(f_c.f_d.c) (f_m.f_n.n) (f_a.f_b.b)$

--> $(f_d.f_m.f_n.n) (f_a.f_b.b)$

--> $f_m.f_n.n$

que como era de esperar, es la máquina F.

de la misma forma se puede definir la máquina OR:

$f_x.f_y.(x f_a.f_b.a) y$

- ¿y así podríamos construir las tablas de verdad?, pregunté.

- No los puedo privar del inmenso placer de descubrirlo por ustedes mismos, contestó nuestro genio, y para ello nos sugirió los siguientes ejercicios:

Ejercicio 39: Completar la tabla de verdad de la conjunción.

Ejercicio 40: Construir la tabla de verdad de la disyunción.

Ejercicio 41: Una expresión para la máquina NOT está dada por:

$f_x.((x (f_a.f_b.b)) (f_c.f_d.c))$

verificar que cuando una máquina NOT procesa a una máquina V da por resultado una máquina F y viceversa.

Los números naturales y la aritmética

- Uno podría pensar que en un mundo sin números la vida sería un poco aburrida, pensó Carlos, sin embargo las cosas no son así. Si bien nuestras máquinas no tienen números, ellas pueden crearlos, de la misma forma que pueden crear las operaciones aritméticas y lógicas, demostraciones de teoremas, y en general, todo lo que sea creable.

- Ahora: ¿que serán aquí los números?, pensé.
- ¡por supuesto que máquinas!, ¡y la operación de suma?, ¡también será una máquina!, como el producto, la potencia, los logaritmos, la derivada y cualquier otra cosa que se nos ocurra.
- Veamos como hacen nuestras máquinas para crear a los números.

La máquina-que-produce-la-siguiente:

Esta es una máquina muy especial, dada cualquier máquina que reciba a la entrada, genera la máquina siguiente a la misma, el significado de esta descripción quedará mas claro unos renglones más abajo. La máquina Suc se define como:

$fa.fb.fc.b(abc)$

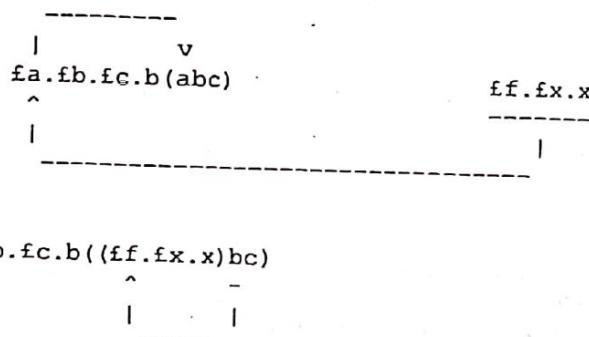
La máquina cero:

Al igual que la máquina anterior, ésta juega un papel muy importante. Su nombre es:

$ff.fx.x$

Veamos que hace la máquina Suc cuando procesa a la máquina cero:

$((fa.fb.fc.b(abc)) (ff.fx.x))$



--> $fb.fc.b((fx.x) c)$

--> $fb.fc.bc$ a esta la llamaremos máquina uno

renombrando las variables para mantener la uniformidad:

--> $ff.fx.fx$

En nuestro universo de máquinas, la expresión anterior viene a ser algo así como el código genético del número uno, de la misma forma que "ff.fx.x" es el código genético del cero. De paso observemos que la máquina cero en realidad es la misma que la máquina KI, que siempre produce la máquina identidad para cualquier máquina que le ingrese.

Cuando la máquina Suc procesa a una máquina uno:

fa.fb.fc.b(abc) v **ff.fx.fx**

```
-->   fb.fc.b((ff.fx.fx)bc)
          |     v
          ^           -
          |           |
          -----
```

--> **f**.**b**.**c**.**b**((**f****x**.**b****x**)**c**)

--> **fb.fc.b(bc)**

renombrando las variables:

ff.fx.f(fx) a esta la llamamos máquina dos

- Observemos que la máquina cero es la misma que la máquina F y que la máquina KI, mientras que la máquina V es la misma que la máquina K, indicó Carlos.

Ejercicio 42: Demostrar que cuando la máquina Suc procesa a la máquina dos, se obtiene

`ff. fx. f (f (fx))`

a esta la llamaremos máquina tres

Ejercicio 43: Seguir usando Suc, para obtener sucesivamente las máquinas:

`ff.fx.f(f(f(fx)))`

`£f.£x.f(f(f(f(fx))))` cinco

- Observemos que siguiendo este procedimiento podemos generar todos los números a partir de las máquinas cero y Suc, aclaró Carlos. En nuestro universo, ésta es la representación del sistema axiomático de Peano, que permite definir a los números naturales, el nombre Suc que le damos a "fa.fb.fc.b(abc)" es la abreviatura de "sucesor", y corresponde a la función sucesor en dicho sistema axiomático. En el ámbito del cálculo lambda a estos números se los conoce como "números de Church".

- ¿Y las operaciones aritméticas?, pregunté.

- ¡una tontería para nuestras poderosas máquinas!. Aquí tenemos la suma:

`fm,fn,ff,fx,mfnfx`

el producto

$\text{fa}.\text{fb}.\text{fz}.(\text{a}(\text{bz}))$

y la potenciación:

$\{a, b, (ba)\}$

Simplificando las expresiones:

Podemos observar que a medida que avanzamos en el análisis del comportamiento de las distintas máquinas, las expresiones se hacen cada vez más complicadas, y por lo tanto más difíciles de seguir. Este problema puede simplificarse si en las expresiones en lugar del nombre de las máquinas usamos su apodo, por ejemplo, en lugar de escribir "fx.fy.x" escribimos "V", en lugar de "ff.fx.ffffx" escribimos "3", etc. De esta forma una expresión lógica como la siguiente

(fx.fy.x y (fx.fy.y)) (fx.fy.x) (fx.fy.y)

puede escribirse simplemente como

(AND V F)

mientras que una suma, por ejemplo:

fm.fn.ff.fx.mfnfx fa.fb.aaaaab fc.fd.ccccd

podríamos escribirla de una forma mas parecida a la que estamos acostumbrados:

(+ 5 3)

entendiendo que el cambio es solamente notacional.

Cuando analizamos el comportamiento de una máquina podemos usar una notación mixta, siempre que ello simplifique las expresiones sin generar confusiones. Tomemos por ejemplo la reducción de la expresión "AND V F" hecha mas arriba:

(fx.fy.xy (fa.fb.b)) (fc.fd.c) (fm.bn.n)

--> (fy.(fc.fd.c) y (fa.fb.b) (fm.bn.n))

--> (fc.fd.c) (fm.bn.n) (fa.fb.b)

--> (fd.fm.bn.n) (fa.fb.b)

--> fm.bn.n

si en lugar de usar "fm.bn.n" usamos "F", la expresión se simplifica:

(fx.fy.xy (fa.fb.b)) (fc.fd.c) F

--> (fy.(fc.fd.c) y (fa.fb.b) F)

--> (fc.fd.c) F (fa.fb.b)

--> (fd.F) (fa.fb.b)

--> F

también podemos tener en cuenta que la expresión "fa.fb.b" podemos reemplazarla por "F", con lo cual:

(fx.fy.xy F) (fc.fd.c) F

--> fy.(fc.fd.c) y F F

--> (fc.fd.c) F F

--> (fd.F) F

--> F

donde se logra una simplificación notable sin perder detalles sobre los mecanismos de reducción aplicados, la máxima simplificación será:

(AND V F)

--> F

Hasta donde simplificar las expresiones sin perder detalles es solo una cuestión de práctica y sentido común.

Algunas otras simplificaciones de ejemplo son:
(Suc 3)

--> 4

(I K)

--> K

Ejercicio 44: Demostrar que la máquina suma, procesando a las máquinas-número tres y dos, obtiene la máquina cinco. Simplificar las expresiones.

Ejercicio 45: Demostrar que la máquina producto, procesando a las máquinas-número tres y dos, obtiene la máquina seis.

¿Máquinas programadoras?:

Ya vimos como nuestras máquinas pueden representar a las constantes lógicas "Verdadero" y "Falso", los números y las operaciones, también que podemos escribir las expresiones de la forma que estamos acostumbrados. En realidad, cuando reemplazamos los nombres por apodos, lo que estamos haciendo es introducir constantes, con lo cual, nuestra operatoria con máquinas abstractas se va transformando en un cálculo aplicado, o sea en un lenguaje de programación. Veamos por ejemplo una típica estructura de control, el "IF" de la mayoría de los lenguajes. Consideremos la máquina siguiente:

fp.fq.fr.pqr.V M N

en la cual M y N son dos máquinas cualesquiera. Aplicando sucesivos pasos de reducción:

--> fq.fr.Vqr M N

--> fr.VMr N

--> VMN

reescribiendo V como fa.fb.a:

--> fa.fb.a MN

--> fb.MN

--> M.

si en la primer expresión hubiéramos tenido F en lugar de V:

fp.fq.fr.pqr F M N

--> fq.fr.Fqr M N

--> fr.FMr N

--> FMN

reescibiendo F como fa.fb.b

--> fa.fb.b MN

--> fb.b N

--> N

como vemos la máquina "fp.fq.fr.pqr" se comporta como selectora, reduciéndose a "M" o "N" según procese a la máquina "V" o "F". A esta máquina le ponemos el apodo de "COND", con lo cual los sucesivos pasos de reducción pueden abreviarse escribiendo:

(COND V M N)

--> M

para el caso verdadero, y también:

(COND F M N)

--> N

para el caso falso, ésta es la típica función selectora de la mayoría de los dialectos Lisp.

¡Una máquina increíble!:

Consideremos la siguiente máquina de aspecto inocente, a la cuál llamaremos máquina Y:

fh. (fx.h (x x)) (fx.h (x x))

cuando esta máquina procesa a una máquina M cualquiera:

fh. (fx.h (x x)) (fx.h (x x)) M

--> (fx.M (x x)) (fx.M (x x)) expresión de YM

--> M ((fx.M (x x)) (fx.M (x x)))

pero la segunda máquina de esta aplicación ¡es nuevamente YM!. Escribiendo estas expresiones en forma simplificada:

YM

--> M(YM)

Esto se interpreta más o menos así: cuando una máquina Y procesa a una máquina cualquiera M, su salida es una máquina M, ¡procesando a ella misma mientras procesa a una máquina M!

Veamos a la máquina Y procesando a la máquina identidad:

Y fa.a

--> fa.a (Y fa.a))

--> (Y fa.a)

¡¿Recursión?!, ¡Sí señor!

Supongamos una máquina, que llamaremos "sumatoria", que cuando procesa una máquina-número N, devuelve la máquina-suma de todas las máquinas-números desde 1 hasta N. Una versión simplificada procesando a la máquina 5 sería:

(suma 5)

--> 15

iremos aproximando paulatinamente a la definición de "sumatoria", para comenzar consideremos la siguiente máquina:

fs.fn.(+ n (s (- n 1)))

donde los símbolos "+" y "-" corresponden a los apodos de la máquina "suma", -ya vista-, y a la máquina "resta", que se define en forma similar. Para reducir esta expresión, todo lo que tenemos que hacer es alimentarla con la función "sumatoria" que estamos tratando de definir. Este efecto paradójico lo logramos alimentando con esta máquina a la máquina Y definida anteriormente:

Y (fs.fn.(+ n (s (- n 1))))

que como sabemos se reduce a:

fs.fn.(+ n (s (- n 1))) (Y(fs.fn.(+ n (s (- n 1))))

->fn.(+ n ((Y (fs.fn.(+ n (s (- n 1)))) (- n 1)

en forma un poco más abreviada:

Y sumatoria

sumatoria (Y sumatoria)

expandiendo la primer definición de "sumatoria":

fs.fn.(+ n (s (- n 1))) (Y sumatoria)

--> fn.(+ n ((Y sumatoria) (- n 1)))

La ocurrencia interior de Y sumatoria, construye una copia de la máquina "sumatoria" original, colocándose a sí misma en lugar de "s" dentro de la copia. Incluyendo el caso elemental, y usando la máquina selectora "COND", la definición de nuestra máquina recursiva queda:

Y (fs.fn. COND (= n 0) 0 (+ n (s (- n 1))))

Las madres de todas las máquinas:

- Tengo una duda, -pregunté-, si en general las máquinas pueden construirse a partir de otras máquinas, ¿existirán máquinas elementales a partir de las cuales puedan construirse todas las demás?.

- Por lo menos eso seguramente ocurre con los combinadores, pensó Carlos, llamaremos "base" a un conjunto de máquinas con las cuales puedan construirse todos los combinadores. Observen esa trinidad.

Y allí estaban, como en un altar dorado, las madres de todas las máquinas:

S = ff.fg.fx.fx(gx)

K = fx.fy.x

I = fx.x

- Es posible demostrar [Curry, 1958], que con estas tres máquinas es posible construir cualquier otra, pensó Carlos.

Con estas definiciones estamos en condiciones de demostrar el siguiente

Teorema I. $\{S, K, I\}$ es una base

En efecto, dada una f -expresión cualquiera, todas las abstracciones pueden ser eliminadas por la aplicación repetida de las siguientes reglas:

$$fx.x \Rightarrow I$$

$$fx.A \Rightarrow KA \quad (\text{si } x \text{ no es libre en } A)$$

$$fx.AB \Rightarrow S(fx.A) (fx.B)$$

Podemos demostrar fácilmente que con la máquina S y la máquina I se construye la máquina omega, en efecto, veamos que $SII = fx.xx$

$$\begin{aligned} & ((S I) I) \\ & (((fx.fy.fz.((x z) (y z))) I) I) \\ \rightarrow & ((fy.fz((I z) (y z))) I) \\ \rightarrow & (fz.((I z) (I z))) \\ \rightarrow & (fz.zz) \end{aligned}$$

Ejercicio 46: Expresar a la siguiente máquina como composición de las máquinas S, K e I :

$$fx.fy.xy$$

Ejercicio 47: Demostrar que SKA se reduce a I para cualquier "A" arbitrario:

Ejercicio 48: Mostrar que $((S(KK))I)S$ es (KS)

Ejercicio 49: ¿A qué se reduce la expresión

$$(((S I) I) X)$$

para cualquier fórmula arbitraria X ?

- Aquí no terminan las sorpresas, pensó nuestro genio, en realidad es posible demostrar [Schönfinkel, 1924] que I puede ser construida con S y K . Podemos enunciar el:

Teorema II. $\{S, K\}$ es una base.

Esto puede demostrarse reemplazando todas las máquinas I por SKA , donde A es una máquina arbitraria cualquiera, por ejemplo K .

- Observen estas reducciones, pensó Carlos

$$\begin{aligned} & SKA \\ & (ff.fg.fx.fx(gx))KA \\ & (fg.fx.Kx(gx))A \\ & (fg.fx.(fa.fb.a)x(gx))A \\ & (fg.fx.fb.x(gx))A \\ & fx.(fb.x)(Ax) \\ & fx.x \end{aligned}$$

Como A es una máquina cualquiera, en general se reemplaza por K y se dice que $I = SKK$.

La máquina de los mil nombres:

- Pero aún no han visto lo más asombroso, pensó Carlos, en algún lugar de este universo infinito está el combinador mas increíble que puedan imaginar, el que genera a absolutamente todos los combinadores que existen.

- ¿Inclusive a S y K?, pregunté.

- Inclusive a S y K, respondió, él está en la cumbre, en la cima de todas la jerarquías. Y si no me equivoco creo que está a la vista.

Traté de percibir con una actitud casi religiosa. Y por un momento logré percibir al fantástico combinador X, el de los mil nombres. Aquí está uno de ellos:

$ff.fS(fx.fy.fz.x)$

- Con ustedes la máquina de los mil nombres, pensó Carlos. El Brahama de los combinadores.

- ¿Por qué de los mil nombres?, pregunté.

- Tiene muchos nombres, de los cuales algunos matemáticos y lógicos han descubierto algunos [Schönfinkel, 1924] [Barendregt, 1984] [Fokker, 1997], fíjense en estos otros:

$ff.fKSK$	(Rosser)
$ff.f(fS(KK))K$	(Barendregt)
$ff.f(fS(KKI))K$	(Böhm)
$fa.fb.fc.fd.cd(a(fx.d))$	(Meredith)

- Es posible demostrar que con todos ellos se pueden generar los combinadores K y S y, por lo tanto, los infinitos combinadores existentes. Como ejemplo podemos enunciar el:

Teorema III. Sea $X = ff.fS(fx.fy.fz.x)$

entonces $\{X\}$ es una basc.

Esto se demuestra reemplazando todas las ocurrencias de K por XX, y todas las ocurrencias de S por X(XX) [Fokker, 1992]

Un último párrafo

Hemos visto las máquinas más asombrosas que podríamos haber imaginado, tan asombrosas que parecen tener el poder de crear todo lo creable, o por lo menos de computar todo lo que es computable. Como dijimos anteriormente, podemos pensar a estas máquinas como funciones, y como tales podemos utilizarlas para desarrollar lenguajes de programación. El cálculo lambda, junto con la teoría de los combinadores, ambos estrechamente relacionados, juegan un papel importante en los fundamentos de la lógica, las matemáticas y las ciencias de la computación.

Respuestas a los ejercicios:

Ejercicio 1:

(fa.a(x y))

R: completamos los paréntesis:

(fa.(a (x y)))
 <---> aplicación
 <-----> aplicación
 <-----> abstracción

Ejercicio 2:

a (fa.a b) a

R: completamos paréntesis:

((a (fa.(a b))) a)
 <---> aplicación
 <-----> abstracción
 <-----> aplicación
 <-----> aplicación

Ejercicio 3:

((fx.(x y)) (fx.z))
 <---> aplicación
 <-----> <----> abstracciones
 <-----> aplicación

Ejercicio 4:

(fx.xxx) (fx.xxx)

R: completando paréntesis:

((fx.((x x) x)) (fx.((x x) x)))
 <---> <---> aplicaciones
 <-----> <-----> aplicaciones
 <-----> <-----> abstracciones
 <-----> aplicación

Ejercicio 5:

(fx.xx) a (fx.xx)

R: completando paréntesis

(((fx.(x x)) a) (fx.(x x)))
 <---> <---> aplicaciones
 <-----> <-----> abstracciones
 <-----> aplicación
 <-----> aplicación

Ejercicio 6:

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

R:

$((\lambda x. ((\lambda y. (xy))x)) (\lambda z.w))$	
<-->	aplicación
<-----> <---->	abstracciones
<-----o-->	aplicación
<----->	abstracción
<----->	aplicación

Ejercicio 7:

a) $(fx. (M1 \ (fy. ((M2 \ M3) \ M4))))$

Ejercicio 8: $\{x, y, z, abxyzc\}$
Libres: a, b, c

Ligadas: x, . y, z

Ejercicio 9: $f_x, f_y, (axb)$
Libres: a, b

Ligadas: x, y

Ejercicio 10: $(fx.b(xy))$
Libres: b, y

Ligada: x

Ejercicio 11: $\{x \cdot ax \mid a \in \{a, b\}\}$

Ligadas: x, a (en la abstracción interior)

Ejercicio 12: ((fx.bax) (fb.by))
Libres: a, b (en la abstracción de la izquierda), y
Ligadas: x, b (en la abstracción de la derecha)

Ejercicio 13: ((fx.x) (fa.fb.ab))
Libres: no hay, se trata de un combinador.
Ligadas: a, b, x

Ejercicio 14: ((fa.fb.fc.b(abc)) (ff.fx.x))
Libres: no hay, se trata de un combinador.
Ligadas: a, b, c, f, x

Ejercicio 15: Efectuar la β -reducción indicada en la observación c)
B:

R:

$$\begin{array}{rcl} & \text{fb. } (\text{fx. } x) \ b \\ \rightarrow & \text{fb. } b \end{array}$$

Ejercicio 16: Encontrar que máquina produce M2 procesando a M1.

Ri

$$\rightarrow ((\lambda x.x) (\lambda a.\lambda b.ab))$$

Scanned by CamScanner

Ejercicio 17: $((fx.(x\ y))\ (fz.z))$ [x/a]

R: $((fa.(a\ y))\ (fz.z))$

Ejercicio 18: $((fx.((fy.(xy))\ x))\ (fz.w))$ [x/b]

R: $((fb.((fy.(by))\ b))\ (fz.w))$

En las siguientes aplicaciones, investigar en que casos M2 es un punto fijo de M1

Ejercicio 19: $((fa.fb.ab)\ (fx.xx))$

R:

--> fb.((fx.xx) b)
--> fb.bb

Es punto fijo. Recordar que el nombre de las variables ligadas es irrelevante: "fx.xx" y "fb.bb" son la misma máquina.

Ejercicio 20: $((fa.(fb.b)\ a)\ (fx.x))$

R:

--> ((fb.b)\ (fx.x))
--> fx.x

Es punto fijo

Ejercicio 21: $((fa.aa)\ (fx.fy.x))$

--> ((fx.fy.x)\ (fa.fb.a))
--> fy.fa.fb.a

No es punto fijo

Ejercicio 22: $((fa.fb.aa)\ (fx.fy.y))$

--> fb.((fx.fy.y)\ (fm.fn.n))
--> fb.fy.y

Es punto fijo

Ejercicio 23: Investigar que produce una máquina K cuando procesa a otra máquina K.

R:

((fa.fb.a)\ (fx.fy.x))
--> fb.(fx.fy.x)

Como era de esperar, esta es una máquina constante, que ante cualquier ingreso devuelve una máquina K.

Ejercicio 24: Investigar qué produce una máquina omega cuando procesa a una máquina K .

R:

((fx.xx) (fx.fy.x))

--> ((fx.fy.x) (fx.fy.x))

aquí estamos en el caso anterior, renombramos variables:

--> ((fx.fy.x) (fa.fb.a))

--> fy.(fa.fb.a)

Ejercicio 25: Investigar que produce una máquina K procesando a una Omega (con mayúsculas).

R:

((fx.fy.x) ((fa.aa) (fb.bb)))

--> fy.((fa.aa) (fb.bb))

Esta es una máquina constante, que ante cualquier ingreso devuelve una máquina Omega.

Ejercicio 26: Investigar qué produce una máquina omega cuando procesa a una KI

R:

((fa.aa) (fx.fy.y))

--> ((fx.fy.y) (fx.fy.y))

--> ((fx.fy.y) (fa.fb.b))

renombramos variables

--> (fy.y)

Ejercicio 27: Investigar que produce una máquina K cuando procesa a una KI.

((fx.fy.x) (fa.fb.b))

--> fy.(fa.fb.b)

esta es una máquina constante que siempre devuelve una máquina constante, que siempre devuelve la máquina identidad.

Ejercicio 28: ((fx.(x y)) (fz.z))

R:

--> (fz.z y)

--> y

Ejercicio 29: $\{(fx, ((fy, (x y)) x)) (fz, w)\}$

R:

```
--> ((fy, ((fz, w) y)) (fz, w))
--> ((fz, w) (fz, w))
--> w
```

Ejercicio 30: Investigar que produce una máquina I^* cuando procesa a una I^* .

R:

```
(fa.fb.ab fx.fy.xy
--> fb.(fx.fy.xy b)
--> fb.fy.by
```

Ejercicio 31: Investigar que produce una máquina $C(Kw)$ cuando procesa a una I^* .

R:

```
(fa.fb.aa fx.fy.xy
--> fb.(fx.fy.xy fm.fn.mn)
--> fb.fy.(fm.fn.mn y)
--> fb.fy.fn.yn
```

Ejercicio 32: Investigar que produce una máquina Kw cuando procesa a una I^* .

R:

```
(fa.fb.bb fx.fy.xy
--> fb.bb
```

Ejercicio 33: Investigar qué produce una máquina $C(Kw)$ cuando procesa a una Kw .

R:

```
(fa.fb.aa fx.fy.yy
--> fb.(fx.fy.yy fm.fn.nn)
--> fx.fy.yy
```

esta es una máquina Kw .

Analizar que produce cada una de las siguientes máquinas cuando procesa a una máquina I :

Ejercicio 34: La máquina W

fa.fb.(ab)b

R:

```
(fa.fb.(ab)b fx.x)
--> fb.((fx.x b) b)
--> fb.bb
```

También podemos llamarla "la máquina omega una vez removida la identidad"

Ejercicio 35: La máquina C

fa.fb.fc.(ac)b

R:

```
((fa.fb.fc.(ac)b) (fx.x))
--> fb.fc.(fx.x c) b
--> fb.fc.cb
```

Ejercicio 36: La máquina B

$f_a.f_b.f_c.a(bc)$

R:

$((f_a.f_b.f_c.a(bc)) (fx.x))$
--> $f_b.f_c.((fx.x)(bc))$
--> $f_b.f_c.bc$

Ejercicio 37: La máquina S

$f_a.f_b.f_c.ac(bc)$

R:

$((f_a.f_b.f_c.ac(bc)) (fx.x))$
--> $f_b.f_c.((fx.x)c(bc))$
--> $f_b.f_c.c(bc)$

esta es una máquina O.

Ejercicio 38: La máquina J

$f_a.f_b.f_c.f_d.ab(adc)$

R:

$((f_a.f_b.f_c.f_d.ab(adc)) (fx.x))$
--> $f_b.f_c.f_d.((fx.x)b((fx.x)dc))$
--> $f_b.f_c.f_d.b(dc)$

Ejercicio 39: Completar la tabla de verdad de la conjunción.

R:

(AND V V)

$(fx.fy.xy (fa.fb.b)) (fc.fd.c) (fm.bn.m)$
--> $(fy.(fc.fd.c) y (fa.fb.b)) (fm.bn.m)$
--> $(fc.fd.c) (fm.bn.m) (fa.fb.b)$
--> $(fd.fm.bn.m) (fa.fb.b)$
--> $fm.bn.m$

que es la máquina V

(AND F V)

$(fx.fy.xy (fa.fb.b)) (fc.fd.d) (fm.bn.m)$
--> $(fy.(fc.fd.d) y (fa.fb.b)) (fm.bn.m)$
--> $(fc.fd.d) (fm.bn.m) (fa.fb.b)$
--> $(fd.d) (fa.fb.b)$
--> $fa.fb.b$

que es una máquina F

(AND F F)

(fx.fy.xy (fa.fb.b)) (fc.fd.d) (fm.fn.n)
--> (fy.(fc.fd.c) y (fa.fb.d)) (fm.fn.n)
--> (fc.fd.c) (fm.fn.n) (fa.fb.b)
--> (fd.fm.fn.n) (fa.fb.b)
--> fm.fn.n

una máquina F

Ejercicio 40: Construir la tabla de verdad de la disyunción.

R:

(OR F F)

((fx.fy.((x fa.fb.a) y)) (fc.fd.d)) (fm.fn.n)
--> ((fy.((fc.fd.d fa.fb.a) y)) (fm.fn.n))
--> ((fc.fd.d) (fa.fb.a)) (fm.fn.n)
--> ((fd.d) (fm.fn.n))
--> fm.fn.n

(OR V F)

((fx.fy.((x fa.fb.a) y) (fc.fd.c)) (fm.fn.n))
--> ((fy.((fc.fd.c fa.fb.a) y)) (fm.fn.n))
--> (((fc.fd.c)
 (fa.fb.a)) (fm.fn.n))
--> (fd.(fa.fb.a) (fm.fn.n))
--> fa.fb.a

(OR F V)

((fx.fy.((x fa.fb.a) y)) (fc.fd.d)) (fm.fn.m)
--> ((fy.((fc.fd.d fa.fb.a) y)) (fm.fn.m))
--> ((fc.fd.d) (fa.fb.a)) (fm.fn.m)
--> ((fd.d) (fm.fn.m))
--> fm.fn.m

(OR V V)

((fx.fy.((x fa.fb.a) y) (fc.fd.c)) (fm.fn.m))
--> ((fy.((fc.fd.c fa.fb.a) y)) (fm.fn.m))
--> (((fc.fd.c) (fa.fb.a)) (fm.fn.m))
--> (fd.(fa.fb.a) (fm.fn.m))
--> fa.fb.a

Ejercicio 41: Una expresión para la máquina NOT está dada por:

$$fx.((x (fa.fb.b)) (fc.fd.c))$$

verificar que cuando una máquina NOT procesa a una máquina V da por resultado una máquina F y viceversa.

R:

procesando a una máquina V:

$$\begin{aligned} & (fx.((x (fa.fb.b)) (fc.fd.c)) (fm.bn.m)) \\ \rightarrow & ((fm.bn.m) (fa.fb.b)) (fc.fd.c) \\ \rightarrow & (fn. (fa.fb.b)) (fc.fd.c) \\ \rightarrow & fa.fb.b \text{ que es la máquina F} \end{aligned}$$

procesando a la máquina F:

$$\begin{aligned} & fx.((x (fa.fb.b)) (fc.fd.c)) (fm.bn.n) \\ \rightarrow & ((fm.bn.n) (fa.fb.b)) (fc.fd.c) \\ \rightarrow & fn.n (fc.fd.c) \\ \rightarrow & fc.fd.c \text{ que es la máquina V} \end{aligned}$$

Ejercicio 42: Demostrar que cuando la máquina Suc procesa a la máquina dos, se obtiene

$$ff.fx.f(f(fx))$$

a esta la llamaremos máquina tres

R:

$$\begin{array}{c} \hline | & v \\ (fa.fb.fc.b(abc) & ff.fx.f(fx)) \\ ^ & \hline | \\ | & | \\ \hline \end{array}$$

$$\begin{aligned} & \begin{array}{c} \hline | & v & v \\ ff.fx.f(fx)bc & \hline | \\ | & | \\ \hline \end{array} \\ \rightarrow & fb.fc.b((ff.fx.f(fx))bc) \\ \rightarrow & fb.fc.b((fx.b(bx))c) \\ \rightarrow & fb.fc.b(b(bc)) \end{aligned}$$

renombrando las variables:

$$ff.fx.f(f(fx)) \quad \text{máquina tres}$$

Ejercicio 43: Seguir usando Suc, para obtener sucesivamente las máquinas:

ff.fx.f(f(f(fx))) cuatro

ff.fx.f(f(f(f(fx)))) cinco

R:

(fa.fb.fc.b(abc) ff.fx.f(f(fx)))

--> fb.fc.b((ff.fx.f(f(fx)))bc)

--> fb.fc.b((fx.b(b(bx)))c)

--> fb.fc.b(b(b(bc)))

o sea:

ff.fx.f(f(f(fx))) máquina cuatro

(fa.fb.fc.b(abc) ff.fx.f(f(f(fx))))

--> fb.fc.b((ff.fx.f(f(f(fx))))bc)

--> fb.fc.b((fx.b(b(b(bx))))c)

--> fb.fc.b(b(b(b(bc))))

o sea:

ff.fx.f(f(f(f(fx)))) máquina cinco

Ejercicio 44: Demostrar que la máquina suma, procesando a las máquinas-número tres y dos, obtiene la máquina cinco. Simplificar las expresiones.

3 + 2 = 5

R:

((fm.fn.ff.fx.mfnfx 3) 2)

-> (fn.ff.fx.(3 f)nf) 2
-> ff.fx.(3 f)2fx
-> ff.fx.(fm.fn.m(m(mn)) f)2fx
-> ff.fx.(fn.f(f(fn)))2fx
-> ff.fx.f(f(f 2))fx
-> ff.fx.f(f(f fx.f(fx))))x
-> ff.fx.f(f(f(f(fx))))

Ejercicio 45: Demostrar que la máquina producto, procesando a las máquinas-número tres y dos, obtiene la máquina seis.

3 * 2 = 6

R:

((fa.fb.fc.a(bc) 3) 2)
(fb.fc.3(bc) 2)
(fc.3(2c))
(fc.3 (fx.fy.x(xy))) c
(fc. (3 fy.c(cy)))
(fc. (fx.fy.x(x(xy)) fb.c(cb)))
(fc.fy.fb.c(cb)) (fb.c(cb)) (fb.c(cb)y))
(fc.fy.fb.c(cb)) (fb.c(cb)) (c(cy))
(fc.fy.fb.c(cb)) (c(c(c(cy)))
(fc.fy.c(c(c(c(c(cy))))

Ejercicio 46: Expresar a la siguiente máquina como composición de las máquinas S, K e I:

$$fx.fy.xy$$

R:

$$fx.fy.xy$$

$$fx.(S(fy.x)(fy.y))$$

$$fx.(S(Kx)(fy.y))$$

dado que $(fa.fb.a x) \rightarrow fb.x$

$$fx.(S(Kx)I)$$

dado que $I = fy.y$

SKI

dado que $(fa.fb.a A) \rightarrow fb.a$

Ejercicio 47: Demostrar que SKA se reduce a I para cualquier "A" arbitrario:

Evaluando $((S K) A)$:

$$(((fx.(fy.(fz.((xz)(yz)))))) K) A)$$

<-----> -

$$((fy.(fz.((Kz)(yz)))) A)$$

$$((fy.(fz.(((fm.bn.m) z)))) (yz)) A)$$

<-----> -

$$((fy.fz.((fn.z)(yz))) A)$$

<-----> -

$$fz.(fn.z) (A z)$$

<-----> -----

$$fz.z$$

Ejercicio 48: Mostrar que $((S(KK))I)S$ es (KS)

$$(((S(KK))I)S)$$

$$\rightarrow (((S((fx.fy.x)K))I)S)$$

<-----> -

$$\rightarrow (((Sfx.K)I)S)$$

$$\rightarrow (((fa.fb.fc.ac(bc))(fx.K))I)S$$

$$\rightarrow (((fb.fc.((fx.K)c)(b c))I)S)$$

$$\rightarrow (((fb.fc.K(b c))I)S)$$

<-----> -

$$\rightarrow ((fc.K(c))S)$$

- -

$$\rightarrow ((fc.Kc)S)$$

$$\rightarrow (K S)$$

Ejercicio 49: ¿A que se reduce la expresión

$((S\ I)\ I)\ X)$

para cualquier fórmula arbitraria X?

$((S\ I)\ I)\ X)$

$\rightarrow (((fx.fy.fz.xz(yz))\ I)\ I)\ X)$

$\rightarrow (((fy.fz.Iz(yz))\ I)\ I)\ X)$

$\rightarrow ((fz.Iz(Iz))\ X)$

$\rightarrow ((fz.zz)\ X)$

$\rightarrow (X\ X)$

Bibliografia:

- [Barendregt, 1984] Barendregt, H. P.: The lambda calculus, its syntax and semantics. North-Holland, 1984.
- [Church, 1930] Church, A.: The calculi of lambda-conversion. Princeton University Press. (1930)
- [Curry, 1958] Curry, H. B. and Feys, R.: Combinatory logic Vol I. North-Holland, 1958
- [Field, 1988] Field, A. J. and Harrison, P. G.: Functional programming. Addison-Wesley Publishing Company. (1988)
- [Fokker, 1992] Fokker, J.: The systematic construction of a one-combinator basis for Lambda-terms. Formal Aspects of Computing (1992) 4: 776-780
- [Larson] Larson, Jim: An Introduction to Lambda Calculus and Scheme.
- [Keenan] Keenan, D. C.: To Dissect a Mockingbird: A Graphical Notation for the Lambda Calculus with Animated Reduction.
<http://uq.net.au/~zzdkeena/Lambda/index.htm>
- [Sethi, 1989] Sethi, Ravi: Programming Languages. Concepts and Constructs. Addison-Wesley Publishing Company. (1989)
- [Schönfinkel, 1924] Schönfinkel, M.: Über die Bausteine der mathematischen Logik. Mathematische Annalen, 92, 307-316 (1924)
- [Smullyan, 1985] Smullyan, R. M.: To mock a mockingbird. Alfred A Knopf, New York. (1985)
- [Tasistro, 1988] Tasistro, A. y Vidart, J.: Programación lógica y funcional. Ediciones Ebai. (1988)

Direcciones útiles:

- <http://www.compapp.dcu.ie/~roconnor/modules/lcc/fisher.html>
- <http://ctp.di.fct.unl.pt/~lcaires/lp2/Lambda.html>
- <http://www.cs.unc.edu/~stotts/204/Lambda>
- <http://www.engr.uconn.edu/~jeffm/Classes/CSE298/Lambda/lambda-1.html>
- <http://www.rbjones.com/rbjpub\logic/cl/cl017.htm>
- <http://www.comp.nus.edu.sg/~cs6202/lambda.html>
- <http://www.plover.com/~mjd/perl/lambda/tpj.html>
- <http://www.cs.princeton.edu/courses/archive/fall98/cs441/Lectures/Lec1/Lec1.html>
- <http://www.csse.monash.edu.au/~lloyd/tildeFP/Lambda/Ch/01.Calc.html>
- <http://www.cis.ohio-state.edu/~gb/cis655/cis655/2>
- <http://www.cs.cornell.edu/Courses/cs212/2000SP/notes/126-lambda.htm!>

Indice

El genio del grupo
La virtualidad real
¡Un mundo increíble!
¿Máquinas?
El mundo de las máquinas
Abstracciones
Aplicaciones
Algunas observaciones
Variables libres y ligadas
 β -reducción
Alfa-reducción
Punto fijo
Una clasificación para las máquinas
Algunas máquinas simples: la máquina I
La máquina constante
Una máquina insólita
Máquinas con dos abstracciones
La máquina K
La máquina KI
Eta-reducción
La familia de dos abstracciones y una aplicación
La máquina I*
La máquina T
La máquina Kw
La máquina C(Kw)
Máquinas mas complejas: La máquina Kx
La máquina L
La máquina O
Verdad y falsedad
Los números naturales y la aritmética
La-máquina-que-produce-la-siguiente
La máquina cero
Simplificando las expresiones
¿Máquinas programadoras?
¡Una máquina increíble!
¡¿Recursión?! , ¡Sí señor!
Las madres de todas las máquinas
La máquina de los mil nombres
Un último párrafo
Respuestas a los ejercicios
Bibliografía
Direcciones útiles

;The Matrix! Peri-Scucimarrí