Introducción al Lambda Cálculo (λC)

Francisco Rafael Yépez Pino

Programación Declarativa Avanzada

Introducción al λ-Cálculo:

1. δ-reducciones y β-reducciones
   - 1.1. λ – Teorías
   - 1.2. Eta-conversión y extensionalidad
   - 1.3. Reducción generada por un programa

2. Formas Normales. Teoremas de Church-Rosser

3. Órdenes de Reducción. Teorema de Estandarización

4. λ-Definibilidad
   - 4.1. Operaciones Lógicas
   - 4.2. Computabilidad
   - 4.3. Puntos Fijos y recursión
   - 4.4. Listas en el λ-Cálculo

1. δ-reducciones y β-reducciones

- Evaluación de una expresión = serie de pasos de reducción, obtenidos cada uno por reescritura,

  e → é ; e se reescribe mediante un paso en é, usando una reducción indicada por →

- Semantica Operacional de un lenguaje funcional = a partir de una expresión inicial, mediante un computo (= sucesivas reescrituras), se obtiene una expresión final, p.e.: $ei \rightarrow_1 e2 \rightarrow_2 \ldots. \rightarrow_{n-1} en \rightarrow_n ef$

- Semántica Operacional de una expresión = computo al que da lugar tal expresión,

  p.e.: $S.O.(en) = e1 \rightarrow_1 e2 \rightarrow_2 \ldots. \rightarrow_{n-1} en$

1. δ-reducciones y β-reducciones (II)

- Redex (reducible expresión) = cada una de las subexpresiones que son factibles de ser reemplazadas al hacer una reducción en una expresión de acuerdo a ciertas reglas.

  P.e.: cond (> 3 2) (- 3 2) (- 2 3)

  tiene 3 redexes: > 3 2 - 3 2 y - 2 3 ,

  que pueden ser reemplazados por: True 1 y -1

- Cómputo finalizado = aquél en el que ya no aparecen más redexes

- Dos tipos de Reglas:
  - Reglas predefinidas = operaciones primitivas del lenguaje, en particular las reglas para las constantes
  - Reglas determinadas por un programa

δ-reducciones

- Son las reducciones que transforman constantes, descritas por $\rightarrow_\delta$ p.e.: > 3 2 $\rightarrow_\delta$ True y + 1 2 $\rightarrow_\delta$ 3

  y también, extendiéndolo:

* (+ 1 2) (- 4 1)		ó	* (+ 1 2) (- 4 1)
$\rightarrow_\delta$			$\rightarrow_\delta$
* (+ 1 2) 3			* 3 (- 4 1)
$\rightarrow_\delta$			$\rightarrow_\delta$
* 3 3			* 3 3
$\rightarrow_\delta$			$\rightarrow_\delta$
9			9

  El orden de la evaluación es muy importante aunque el resultado no dependa de este

- Se dice que la extensión de la relación inicial δ es compatible o sustutiva, ya que permite reducir una expresión a partir de reducciones de subexpresiones

- Usaremos $\twoheadrightarrow_\delta$ para indicar una reducción en varios pasos (cierre transitivo y reflexivo de $\rightarrow_\delta$), y así: * (+ 1 2) (- 4 1) $\twoheadrightarrow_\delta$ 9

δ-reducciones (II)

- Las reglas condicionales están entre las δ-reglas más interesantes

  - Cond *True* E F $\rightarrow_\delta$ E
  - Cond *False* E F $\rightarrow_\delta$ F

  - Estas condicionales se dicen estrictas en el primer argumento si éste está en forma normal o es *True o False*

β-reducciones

- Con las β-reducciones reescribimos una llamada f u a la función f si conocemos el cuerpo de la función f.

  Es decir, si para f u tenemos que $f \equiv \lambda x.E$, hay que reemplazar en el cuerpo E todas las ocurrencias de x por el argumento u:

  $$(\lambda x.E)\ u \rightarrow_\beta\ <x:=u>E$$

  P.e.: $(\lambda x.x)\ y \rightarrow_\beta y$

- La regla puede utilizarse también en el sentido opuesto:

  $$(\lambda x.E)\ u \leftarrow_\beta\ <x:=u>E$$

  y en ese caso se llama una β-expansión. Si el sentido no se precisa se habla de β- equivalencia: $(\lambda x.E)\ u =_\beta\ <x:=u>E$

- $\rightarrow_\beta$ tambien es sustitutiva: $(\lambda x.x\ y)\ (\lambda z.z) \rightarrow_\beta (\lambda z.z)\ y \rightarrow_\beta y$

- Por medio de $\twoheadrightarrow_\beta$ denotamos el cierre transitivo y reflexivo de $\rightarrow_\beta$, teniendo: $(\lambda x.x\ y)\ (\lambda z.z) \twoheadrightarrow_\beta y$

Mezcla de δ-reducciones y β-reducciones

- Indicadas por $\twoheadrightarrow_{\beta\delta}$

$$
\begin{array}{lllllll}
(i) & (\lambda x.*\ x\ x)\ 2 & \rightarrow_\beta & *\ 2\ 2 & \rightarrow_\delta & 4 & \\
 & (\lambda x.*\ x\ x)\ 2 & \twoheadrightarrow_{\beta\delta} & 4 & & & \\
 & & & & & & \\
(ii) & (\lambda x.x\ y)\ (\lambda z.z) & \rightarrow_\beta & (\lambda z.z)\ y & \rightarrow_\beta & y & \\
 & (\lambda x.x\ y)\ (\lambda z.z) & \twoheadrightarrow_\beta & y & & & \\
 & & & & & & \\
(iii) & ((\lambda x.\lambda y.*\ x\ y)\ 7)\ 8 & \rightarrow_\beta & (\lambda y.*\ 7\ y)\ 8 & \rightarrow_\beta & *\ 7\ 8 & \rightarrow_\delta\ 56 \\
 & ((\lambda x.\lambda y.*\ x\ y)\ 7)\ 8 & \twoheadrightarrow_{\beta\delta} & 56 & & & \\
 & & & & & & \\
(iv) & (\lambda f.f\ 3)\ (\lambda x.+\ x\ 1) & \rightarrow_\beta & (\lambda x.+\ x\ 1)\ 3 & \rightarrow_\beta & +\ 3\ 1 & \rightarrow_\delta\ 4 \\
 & (\lambda f.f\ 3)\ (\lambda x.+\ x\ 1) & \twoheadrightarrow_{\beta\delta} & 4 & & &
\end{array}
$$

- $\twoheadrightarrow_{\beta\delta}$ genera una relación de igualdad $=_{\beta\delta}$ que es de equivalencia (reflexiva, simétrica y transitiva) y debe de cumlplir que:

$$A \twoheadrightarrow_{\beta\delta} B \quad \Rightarrow \quad A =_{\beta\delta} B$$

Reducciones generadas por una relación

- Sea la relación $\mathcal{R} \subseteq \Lambda$ x Λ, escribiremos x$\mathcal{R}$y o (x,y)$\in\mathcal{R}$

  Entre estas relaciones resaltamos:

$$
\begin{array}{lcl}
\beta & = & \{\ (\ (\lambda x.M)\ N,\ [x := N]M\ )\ \mid\ M, N \in \Lambda\ \} \\
\eta & = & \{\ (\ \lambda x.M\ x,\ M\ )\ \mid\ M \in \Lambda\ \} \\
\beta\delta & = & \beta \cup \delta \\
\beta\eta & = & \beta \cup \eta \\
\beta\eta\delta & = & \beta \cup \eta \cup \delta
\end{array}
$$

  Que inducen distintas nociones de reducción, que resultan ser compatibles.

- Definición : Se dice que $\mathcal{R}$ es una relación compatible si $\forall$ A, B, M, N $\in \Lambda$
  (A,B)$\in\mathcal{R}$ $\Rightarrow$ (M A,M B)$\in\mathcal{R}$, (A N,B N)$\in\mathcal{R}$ *y* (λx.A, λx.B)$\in\mathcal{R}$

Una reducción es una relación compatible, reflexiva y transitiva (no necesariamente simétrica).
Una igualdad (sustitutiva) es una relación de equivalencia compatible.

Reducciones generadas por una relación (II)

- Definición: (Reducciones generadas por una relación)

Definición **(Cierre compatible de una relación)** $\rightarrow_{\mathcal{R}}$ *es el* **cierre compatible** *de* $\mathcal{R}$ *o* **reducción en un paso**, *es decir, la mínima relación que verifica los axiomas*

$$\frac{(A,B) \in \mathcal{R}}{A \rightarrow_{\mathcal{R}} B} \qquad \frac{A \rightarrow_{\mathcal{R}} B}{M\ A \rightarrow_{\mathcal{R}} M\ B} \qquad \frac{A \rightarrow_{\mathcal{R}} B}{A\ N \rightarrow_{\mathcal{R}} B\ N} \qquad \frac{A \rightarrow_{\mathcal{R}} B}{\lambda x.A \rightarrow_{\mathcal{R}} \lambda x.B}$$

Definición **(Cierre reflexivo y transitivo de una relación)** $\twoheadrightarrow_{\mathcal{R}}$ *es el* **cierre reflexivo y transitivo** *de* $\rightarrow_{\mathcal{R}}$, *es decir, la mínima relación que verifica*

$$\frac{A \rightarrow_{\mathcal{R}} B}{A \twoheadrightarrow_{\mathcal{R}} B} \qquad \frac{}{A \twoheadrightarrow_{\mathcal{R}} A} \qquad \frac{A \twoheadrightarrow_{\mathcal{R}} B \qquad B \twoheadrightarrow_{\mathcal{R}} C}{A \twoheadrightarrow_{\mathcal{R}} C}$$

Definición **(Equivalencia generada por una relación)** $=_{\mathcal{R}}$ *es la (relación de)* **equivalencia** *generada por* $\twoheadrightarrow_{\mathcal{R}}$, *es decir, la mínima relación verificando*

$$\frac{A \twoheadrightarrow_{\mathcal{R}} B}{A =_{\mathcal{R}} B} \qquad \frac{A =_{\mathcal{R}} B}{B =_{\mathcal{R}} A} \qquad \frac{A =_{\mathcal{R}} B \qquad B =_{\mathcal{R}} C}{A =_{\mathcal{R}} C}$$

Reducciones generadas por una relación (III)

- Lema: Las relaciones $\to_{\mathcal{R}}$, $\twoheadrightarrow_{\mathcal{R}}$, $=_{\mathcal{R}}$ son todas compatibles, con lo cual $\twoheadrightarrow_{\mathcal{R}}$ es una reducción y $=_{\mathcal{R}}$ es una igualdad.

  Por tanto, las relaciones $\twoheadrightarrow_{\beta}$, $\twoheadrightarrow_{\delta}$ y $\twoheadrightarrow_{\beta\delta}$ son todas reducciones.

1.1 λ – Teorías

- La λ–teoría tiene como fórmulas M = N, donde M y N son λ-expresiones y como reglas de derivación:

  - Axiomas de equivalencia:

$$\frac{}{M = M} \qquad \frac{M = N}{N = M} \qquad \frac{M = N \qquad N = P}{M = P}$$

  - Axiomas de compatibilidad:

$$\frac{M = N}{Y\ M = Y\ N} \qquad \frac{M = N}{M\ X = N\ X} \qquad (\xi\textbf{–regla})\frac{M = N}{\lambda x.M = \lambda x.N}$$

  - β**–regla**:

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

- Si la igualdad M = N es demostrable en tal teoría escribiremos simplemente: λC ⊢ M = N. Por tanto, la λ – teoría es una teoría con igualdad compatible donde es válida la β-regla.

λ – Teorías (II)

- En el λC tenemos dos igualdades:
  - La igualdad $=_\beta$ generada por $\twoheadrightarrow_\beta$
  - La igualdad $=$ descrita por la λ–teoría

- Puede demostrarse que ambas representan el mismo concepto, es decir: $M =_\beta N \Leftrightarrow \lambda C \vdash M = N$

- La principal ventaja del uso del λC como teoría es que podemos razonar sobre las igualdades de forma más simple.

λ – Teorías (III)

- En una teoría $\mathcal{T}$ con igualdad donde son aplicables los siguientes axiomas (siendo A y B igualdades)

$$\begin{array}{lll} \vdash & A \Rightarrow (B \Rightarrow A) & \\ \vdash & (\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B) & \\ A\ ,\ (A \Rightarrow B) \vdash & B & (modus\ ponens) \end{array}$$

- Se cumple que los dos enunciados siguientes con equivalentes:
  - $\mathcal{T}$ es consistente (p.e. existe una igualdad M = N no demostrable)
  - No existen M, N tales que $\mathcal{T} \vdash$ (M = N) y $\mathcal{T} \vdash \neg$(M = N)

- En nuestra λ – Teoría incluiremos los axiomas anteriores. Luego son equivalentes:
  - $\forall$ M,N $\in \Lambda$. λC $\vdash$ (M = N)
  - El λC es inconsistente

- Esto lo usaremos para encontrar inconsistencias.

Convenio de Variables y definicion de sustitución

- <x:=u> e → "sustituir x por u en todas las apariciones libres de x en e"

- La sustitución plantea problemas si en e aparece x ligada:

  $F \equiv \lambda xy.y\ x \equiv \lambda x.(\lambda\ y.y\ x)$, entonces $F\ M\ N =_\beta N\ M$,
  si $M \equiv y$ & $N \equiv x$, entonces $F\ y\ x =_\beta x\ y$, pero tambien:

  $F\ y\ x \equiv ((\lambda x.(\lambda\ y.y\ x))\ y)\ x \rightarrow_\beta (\lambda y.y\ y)\ x \rightarrow_\beta x\ x$

  Luego debe darse $x\ y =_\beta x\ x$, que es absurdo e introduce inconsistencias: $x\ y = x\ x \Rightarrow \forall\ M,N \in \Lambda\ .\ \lambda C \vdash (M = N)$

- El problema está en la definición de sustitución, pues en estos casos es obligatorio realizar una α-conversion ($\equiv_\alpha$) o renombramiento de variables:

  $F\ y\ x \equiv ((\lambda x.(\lambda\ y.y\ x))\ y)\ x \equiv_\alpha ((\lambda z.(\lambda\ r.r\ z))\ y)\ x \rightarrow_\beta (\lambda r.r\ y)\ x \rightarrow_\beta x\ y$

Convenio de Variables y definicion de sustitución (II)

- Convenio (Convenio sobre Variables (CV)):
  - *Si una serie de términos aparecen juntos en cierto contexto, entonces todas las variables instanciables de tales términos se considerarán distintas de las libres.*
- Definición: Teniendo en cuanta de ahora en adelante el convenio de variables se define:

$$\begin{array}{lcll} [x := N]x & \equiv & N & \\ [x := N]y & \equiv & y & si\ x \not\equiv y \\ [x := N](P\ Q) & \equiv & [x := N]P\ [x := N]Q & \\ [x := N](\lambda y.P) & \equiv & \lambda y.[x := N] & \end{array}$$

- Para cualquier relación $\mathcal{R}$ se cumple:

$$A \twoheadrightarrow_{\mathcal{R}} B \qquad \Rightarrow \qquad [x := A]M \twoheadrightarrow_{\mathcal{R}} [x := B]M$$

- Pero no necesariamente se tiene:

$$M \twoheadrightarrow_{\mathcal{R}} N \qquad \not\Rightarrow \qquad [x := A]M \twoheadrightarrow_{\mathcal{R}} [x := A]N$$

- Una relación $\mathcal{R}$ es sustitutiva si:

$$\forall M, N, A \in \Lambda \ : \qquad (M, N) \in \mathcal{R} \Rightarrow (\ [x := A]M\ ,\ [x := A]N\ ) \in \mathcal{R}$$

- Si $\mathcal{R}$ sustitutiva, también: $\rightarrow_{\mathcal{R}}$, $\twoheadrightarrow_{\mathcal{R}}$, $=_{\mathcal{R}}$; si β sustitutiva, tambien $\rightarrow_\beta$ $\twoheadrightarrow_\beta$ $=_\beta$ y si δ sustitutiva, tambien $\rightarrow_\delta$ $\twoheadrightarrow_{\beta\delta}$ $=_{\beta\,\delta}$

1.2. Eta-conversión y extensionalidad

- En el λC siempre se tiene: $(\lambda x.M)\ x = M$

- Sin embago, la regla (η): $\dfrac{}{\lambda x.M\ \ x = M}$ (el cuerpo de la función es M x) no es necesariamente cierta.

- P.e.:si consideramos la función suc ≡ λx. + 1 x, se verifica suc y = + 1 y, de forma que suc tiene el mismo comportamiento que la función parcial + 1, pero no pueden considerarse iguales salvo que se considere la η-regla: (λx. + 1 x) = + 1

1.2. Eta-conversión y extensionalidad (II)

- Curry demostró en 1941 que λC ∪ (ext) es equivalente a λη, donde la siguiente regla (ext) captura el **principio de extensionalidad:**

$$(\mathbf{ext})\frac{M\ x = N\ x}{M = N} \qquad \text{si } x \notin \phi[M\ N]$$

- La (ext) se usa para obtener funciones vía parcialización. Como aplicación, veamos que en el ληδ-cálculo se tiene:

  COND TRUE 3 = λx.3, comprobémoslo de dos formas:

  - A través de la η–regla:

$$\begin{aligned} & COND\ TRUE\ 3 \\ =\ & !\ \eta\text{–regla} \\ & \lambda x.COND\ TRUE\ 3\ x \\ =\ & !\ \delta\text{–regla para } COND \\ & \lambda x.3 \end{aligned}$$

  - A través de la regla (**ext**):

$$\begin{aligned} & TRUE \\ =\ & !\ \text{Reflexividad de} = \\ & 3 = 3 \\ =\ & !\ \beta\text{–regla} \\ & 3 = (\lambda x.3)\ y \\ =\ & !\ \delta\text{–regla para } COND \\ & COND\ TRUE\ 3\ y = (\lambda x.3)\ y \\ \Rightarrow\ & !\ \text{regla } (\mathbf{ext}) \\ & COND\ TRUE\ 3 = \lambda x.3 \end{aligned}$$

1.3. Reducción generada por un programa

- Consideremos el siguiente programa Haskell junto sus evaluaciones en el intérprete:

```
pi          :: Float          --1
cero, cien  :: Int            --2
pi          = 3.1415927       --3
cero        = 0               --4
cien        = 100             --5

sucesor     :: Int → Int      --6
sucesor n   = n + 1           --7

difCua      :: Int → Int      --8
difCua n    = (n − 2) * (n + 2)   --9

rec         :: Int → Int      --10
rec x       = rec x           --11

nula        :: Int → Int      --12
nula x      = 0               --13
```

```
MAIN> cero + cien
100
MAIN> sucesor 4
5
MAIN> difCua 4
12
```

Que equivale formalmente a:

$$cero + cien \Longrightarrow 100$$
$$sucesor\ 4 \Longrightarrow 5$$
$$difCua\ 4 \Longrightarrow 12$$

¿Podemos formalizar la reducción $\Rightarrow$?

1.3. Reducción generada por un programa (II)

$$
\begin{array}{lll}
\textbf{pi} & \overset{1}{\equiv} & 3.1415927 \\
\textbf{cero} & \overset{4}{\equiv} & 0 \\
\textbf{cien} & \overset{5}{\equiv} & 100 \\
\textbf{sucesor} & \overset{6}{\equiv} & \lambda\, n.\ +\ n\ 1 \\
\textbf{difCua} & \overset{9}{\equiv} & \lambda\, n.\ *\ (-\ n\ 2)\ (+\ n\ 2) \\
\textbf{rec} & \overset{10}{\equiv} & \ldots \\
\textbf{nula} & \overset{13}{\equiv} & \lambda\, x.\, 0
\end{array}
$$

Diremos que:

e $\Rightarrow$é si e* $\twoheadrightarrow_{\beta\delta}$ é*

donde E* es la traslación al λC de E

$$(cero\ +\ cien)^* \equiv +\ \mathbf{cero\ cien} \equiv +\ 0\ 100 \rightarrow_\delta\ 100$$

de donde $cero\ +\ cien \Longrightarrow 100.$

$$(sucesor\ 4)^* \equiv (\lambda\, n.+\ n\ 1)\ 4 \twoheadrightarrow_{\beta\delta}\ 5$$

de donde $sucesor\ 4 \Longrightarrow 5$

$$(nula\ (rec\ 2))^* \equiv (\lambda\, x.\, 0)(\mathbf{rec}\ 2) \rightarrow_\beta\ 0$$

de donde $difCua\ 4 \Longrightarrow 12$

2. Formas Normales. Teoremas de Church-Rosser

- Un término M se dice que
  - es un $\mathcal{R}$–redex si para algún término N se tiene que $M \rightarrow_{\mathcal{R}} N$
  - está en $\mathcal{R}$–forma normal ($\mathcal{R}$–FN) si no contiene ningún $\mathcal{R}$–redex
- Por tanto definimos el conjunto $\mathcal{FN}$ de forma inductiva como:

  (1) $x \in \mathcal{FN}$

  (2) si $M1,\ldots,Mm \in \mathcal{FN}$ entonces $x\ M1\ \ldots\ Mm \in \mathcal{FN}$

  (3) si $M \in \mathcal{FN}$ entonces $\lambda x.M \in \mathcal{FN}$

  Las FN se corresponden con la idea de "fin de cómputo", así un evaluador deberá seguir "mientras existan redexes, reducir uno de ellos".

- Lema (caracterización de formas normales): Si M es una $\mathcal{R}$–FN, entonces $M \twoheadrightarrow_{\mathcal{R}} N \Rightarrow M \equiv N$.

No todas las expresiones tienen FN, p.e. si $\Omega \equiv(\lambda y.y\ y)\ (\lambda y.y\ y)$, Ω es un redex y no es FN, pero además:

$\Omega \equiv(\lambda y.y\ y)\ (\lambda y.y\ y) \rightarrow_{\beta} <z:= \lambda y.y\ y>(z\ z) \equiv (\lambda y.y\ y)\ (\lambda y.y\ y) \equiv \Omega$

$\Omega \rightarrow_{\beta} \Omega$ y el cómputo de la expresión Ω no termina.

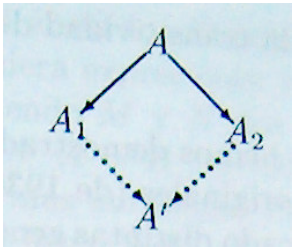
2. Formas Normales. Teoremas de Church-Rosser(II)

- Definición (Confluencia):

(a) Se dice que una relación $\rightarrow$ verifica la propiedad del diamante,donotada por $\rightarrow\vDash^{\diamond}$, si se tiene, $\forall$ A,A_1,A_2,

$$A \rightarrow A_1 \wedge A \rightarrow A_2 \quad \Rightarrow \quad \exists Á.\ A_1 \rightarrow Á \wedge A_2 \rightarrow Á$$

Tambien se indica diciendo que el siguiente diagrama es comutativo:



O incluso también en la forma: $\leftarrow \cdot \rightarrow \subseteq \rightarrow \cdot \leftarrow$, donde $\cdot$ denota la composición de funciones

(b) Una relación $\mathcal{R}$ se dice confluente o también que verifica la propiedad de Church-Rosser, que donotamos con $\mathcal{R} \in$ CR, si la relación inducida $\twoheadrightarrow_{\mathcal{R}}$ verifica la propiedad del diamante; es decir: $\mathcal{R} \in$ CR $\Leftrightarrow$ $\twoheadrightarrow_{\mathcal{R}} \vDash^{\diamond}$

2. Formas Normales. Teoremas de Church-Rosser(III)

- Teorema (Teoremas de Church-Rosser, 1936)

(a) $\beta \in CR,\ \eta \in CR,\ \beta\eta \in CR$

Sea ahora $\mathcal{R} \in CR$. *Entonces:*

(b) $M =_{\mathcal{R}} N \ \Rightarrow\ \exists Z \,.\, M \twoheadrightarrow_{\mathcal{R}} Z \wedge N \twoheadrightarrow_{\mathcal{R}} Z.$

(c) *Si* N *es una* $\mathcal{R}$*–FN de* M, *entonces* $M \twoheadrightarrow_{\mathcal{R}} N$.

(d) *Si un término tiene forma normal, dicha forma normal es única (salvo* α*–equivalencias).*

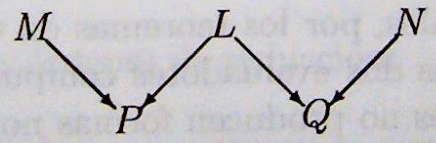
2. Formas Normales. Teoremas de Church-Rosser(IV)

Demostración.– Probemos (b) por inducción sobre la estructura de la derivación $M =_{\mathcal{R}} N$ (Definición 16.1):

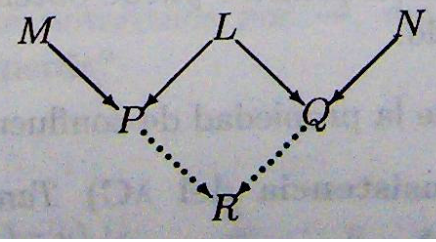
— Si $M =_{\mathcal{R}} N$ es consecuencia de $\dfrac{M \twoheadrightarrow_{\mathcal{R}} N}{M =_{\mathcal{R}} N}$ tomamos $Z \equiv N$.

— Si $M =_{\mathcal{R}} N$ es consecuencia de $\dfrac{N =_{\mathcal{R}} M}{M =_{\mathcal{R}} N}$ aplicamos hipótesis de inducción: $\exists Z \,.\, N \twoheadrightarrow_{\mathcal{R}} Z \wedge M \twoheadrightarrow_{\mathcal{R}} Z$.

— Si $M =_{\mathcal{R}} N$ es consecuencia de $\dfrac{M =_{\mathcal{R}} L \qquad L =_{\mathcal{R}} N}{M =_{\mathcal{R}} N}$, por hipótesis de inducción existen términos P y Q verificando



y por la propiedad de confluencia tenemos que existe R verificando:



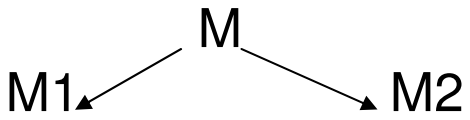
Y ahora basta aplicar la transitividad de $\twoheadrightarrow_{\mathcal{R}}$

y $M =_{\mathcal{R}} N$

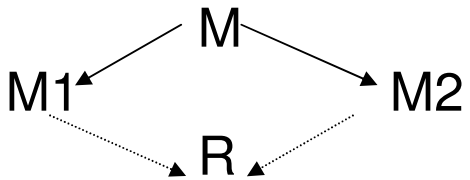
2. Formas Normales. Teoremas de Church-Rosser(V)

- Ejemplo: $M \rightarrow_1 M1$ $M \rightarrow_2 M2$

o lo que es igual:

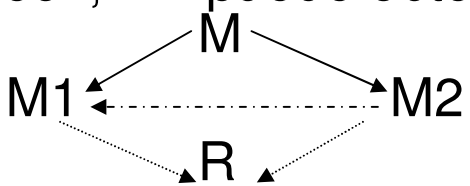


Como $\rightarrow_1 \subset \twoheadrightarrow_\beta$ y $\rightarrow_2 \subset \twoheadrightarrow_\beta$ y por ser $\twoheadrightarrow_\beta$ confluente, el diagrama anterior se puede completar con:



Es decir, los dos evaluadores 1 y 2, computan el mismo valor,

pero si el 1 produce una forma normal y el otro no, entonces sabemos que $M2 \twoheadrightarrow_\beta M1$; es decir, M1 puede obtenerse de M2 computándolo algo más por 2.



2. Formas Normales. Teoremas de Church-Rosser(VI)

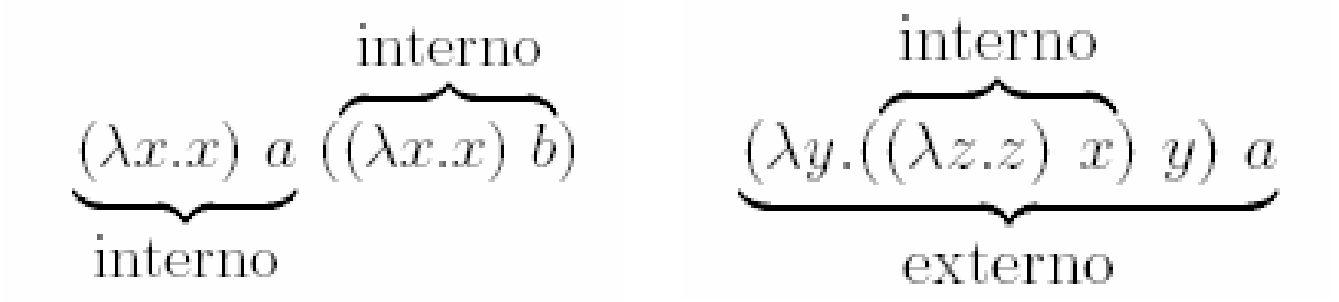
- Corolario (Consistencia del λC):
  - Tanto el λ-Cálculo como el λη-Cálculo, son consistentes.

- Demostración (por reducción al absurdo):
  - Supongamos que λC sea inconsistente
  - Entonces para cualquiera de las expresiones M y M se tiene que M=N
  - En particular, siendo M y N dos formas normales tendremos que $\lambda C \vdash M=N$
  - Pero como $\lambda C \vdash A=B \Rightarrow A =_\beta B$, entonces $M =_\beta N$
  - Por el teorema de Church-Rosser tendremos que $M \equiv N$.
  - Si M = x & N = y , que son dos formas normales no α-equivalentes:
  - Tendríamos que $x \equiv y$, lo que es absurdo, por tanto, la suposición inicial de que el λC sea inconsistente es falsa
    - **el λC es consistente.**

3. Ordenes de reducción.Teorema de estandarización

- Importancia de la elección del siguiente redex para la finalización del computo:
  - Ω ≡ (λx.x x) (λx.x x)
  - La expresión (λx.y) Ω tiene dos redexes: uno interior (Ω) y otro exterior (toda la expresión)
  - Y según por el que reduzcamos podemos tener **y** (reduciendo por el exterior) o la misma expresión (por el interior).

- Un redex es externo si no está contenido en otro redex, en cuyo caso será interno, por ejemplo:



3. Ordenes de reducción.Teorema de estandarización (II)

- Se distinguen dos órdenes de reducción:
  - Orden normal o estándar, que notaremos por $\twoheadrightarrow^s$, donde se reduce el redex externo de más a la izquierda
    - $(\lambda x.y)\ \Omega \twoheadrightarrow^s y$ (ejemplo que vimos antes $\Omega \equiv (\lambda x.x\ x)\ (\lambda x.x\ x)$)
  - Orden aplicativo, que notaremos por $\twoheadrightarrow^a$ donde se reduce el redex interno de más a la izquierda
    - $(\lambda x.y)\ \Omega \twoheadrightarrow^a (\lambda x.y)\ \Omega \twoheadrightarrow^a \cdots$ y no terminaría el cómputo.

    Pero según uno de los teoremas de Church-Rosser, si una expresión tiene FN, ésta es única; en este caso y.

Otro ejemplo:

$$\lambda\, a.(\lambda\, b.\underline{(\lambda\, c.c)\ b}\ b)\ d$$
$$\xrightarrow{a}$$
$$\lambda\, a.\underline{(\lambda\, b.b\ b)\ d}$$
$$\xrightarrow{a}$$
$$\lambda\, a.d\ d$$

con lo cual, $\lambda a.(\lambda b.(\lambda c.c)\ b\ b)\ d \twoheadrightarrow_\beta \lambda a.d\ d$ pero también

$$\lambda a.(\lambda b.(\lambda c.c)\ b\ b)\ d \xrightarrow{s} \lambda a.d\ d$$

Formas Normales por la cabeza.

- Cada término:
  - Es una forma normal por la cabeza (FNC):
    - $M \equiv \lambda x_1 x_2 \ldots x_n.x \; M1 \ldots Mm \;\; (n \geq 0) \wedge (m \geq 0)$
  - O es β-reducible por la cabeza:
    - $M \equiv \lambda x_1 x_2 \ldots x_n.(\lambda x.M0)\; M1 \ldots Mm \;\; (n \geq 0) \wedge (m \geq 0)$
      donde al redex $(\lambda x.M0)$ se le llama redex de cabecera.
- M admite FNC si M = N con N en FNC.

- Teorema de Estandarización; Curry-Feys, 1958:
  - Si $M \twoheadrightarrow_\beta N$, entonces también $M \twoheadrightarrow^s N$.
  - M tiene FNC si y sólo si su reducción por la cabeza termina, donde ésta consiste en la reducción reiterada del redex de cabecera.

- La reducción por la cabeza determina de forma única una secuencia de términos (que termina o no).

- En la reducción estandar está incluida la reducción de cabecera, teniendo:

$$M \underbrace{\overset{c}{\twoheadrightarrow} \; Z \; \overset{i}{\twoheadrightarrow}}_{\text{red. estándar}} N$$

Donde :$\twoheadrightarrow^c$ es la reducción por cabecera y $\twoheadrightarrow^i$ indica reducciones internas.

Formas Normales por la cabeza. Ejemplo: (siendo I = λz.z):

- La λE

$$\lambda x.x\ (\mathbf{I}\ b)$$

está en FNC (pues no existe redex de cabecera); sin embargo no es una FN y la reducción normal lo reduciría a $\lambda x.x\ b$

- La λE

$$(\lambda x.x\ x)\ (\lambda x.x\ x)$$

no admite FNC ni tampoco FN (ya que toda FN es una FNC)

- La λE

$$\lambda x.\mathbf{I}\ x\ (\mathbf{I}\ \mathbf{I})$$

admite como FNC tanto $\lambda x.x\ (\mathbf{I}\ \mathbf{I})$ como $\lambda x.x\ \mathbf{I}$

- La λE

$$(\lambda x.x\ x)\ \mathbf{I}\ x\ (\mathbf{I}\ a)$$

terminaría en $x\ (\mathbf{I}\ a)$ si utilizamos reducción por la cabeza, pero la reducción estándar continuaría hasta obtener la expresión $x\ a$:

$$(\lambda x.x\ x)\ \mathbf{I}\ x\ (\mathbf{I}\ a) \overset{c}{\twoheadrightarrow} \underbrace{x\ (\mathbf{I}\ a)}_{\text{fin cómputo si FNC}} \overset{s}{\twoheadrightarrow} x\ a$$

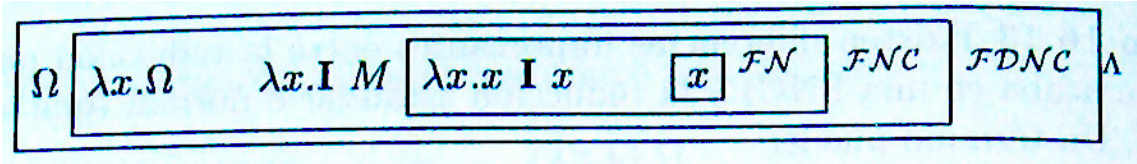
Además, obsérvese que la λE original también admite como FNC la expresión $x\ a$, pues

$$\underline{(\lambda x.x\ x)\ \mathbf{I}}\ x\ \underline{(\mathbf{I}\ a)} \twoheadrightarrow \underline{\mathbf{I}\ \mathbf{I}}\ x\ a \rightarrow \underline{\mathbf{I}\ x}\ a \rightarrow x\ a$$

pero la FNC canónica (i.e., la obtenida por reducción por la cabeza) es la expresión $x\ (\mathbf{I}\ a)$

Forma Dévil-Normal por la cabeza

- Se define el conjunto FDNC como:
  - Toda constante es una FDNC
  - λx.E es una FDNC
  - Si f es una constante o una función de aridad k > n ≥ 0, entonces f E1 E2 … En es una FNDC
- Se diferencian de las FNC en que no se evalúan las λ-abstracciones internas.



- Clasificación de cinco λ-expresiones. λx.**I** M no está en FNC pero si en FDNC, por no tener redexes externos.

- Una expresión que consista en una sola variable es, según la definición, una FDNC; sin embargo, el resultado de un programa funcional no debe contener variables libres (p.e., + *x* 1) y podemos limitarnos a FDNC cerradas.

- Una ventaja de la reducción a una FDNC es que se evita la β-reducción en presencia de variables libres.

Forma Dévil-Normal por la cabeza (II)

- Por Ejemplo:

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

está en FDNC, no reducir redex interno, pero

$$(\lambda x.(\lambda y.\lambda x.+\ x\ y)\ x)\ 4$$

no está en FDNC y hay que reducirla por orden normal:

$$(\lambda x.(\lambda y.\lambda x.+\ x\ y)\ x)\ 4 \quad \rightarrow \quad (\lambda y.\lambda x.+\ x\ y)\ 4 \quad \rightarrow \quad \lambda x.+\ x\ 4$$

4. λ-Definibilidad

- La noción de λ-Definibilidad es conceptualmente la base de la programación funcional

- Church introduce la noción de λ-Definibilidad de funciones f: $N^k \rightarrow N$ con objeto de capturar la noción de computabilidad.

- Establece que las funciones intuitivamente computables deben ser λ-definibles (tesis de Church), lo que motiva el desarrollo de lo que se conoce como teoría de las computabilidad o toría básica de las funciones recursivas.

- Además de la Aritmética y la Lógica, las estructuras de datos (lineales y arbóreas) también son representables (λ-definibles) en el λC.

- En la sintaxis que hemos visto del λC consideramos las funciones predefinidas (δ-reglas) como +, 0, -1, cond, etc.

- Ahora veremos como pueden definirse en un λC puro.

4.1. Operaciones Lógicas

Una λ-expesión sin variables libres es independiente del contexto. A tales expresiones las llamaremos **Combinadores** y las usaremos para obtener las funciones de orden superior.

Si definimos los combinadores

$$\begin{aligned} \mathbf{C} &= \lambda xy.x \\ \mathbf{F} &= \lambda xy.y \end{aligned}$$

se tiene:

$$\begin{aligned} \mathbf{C}\ P\ Q &= P \\ \mathbf{F}\ P\ Q &= Q \end{aligned}$$

combinador condicional **cond**

$$\mathbf{cond} = \lambda cpq.c\ p\ q$$

ya que cumple las reglas de reducción conocidas del si-entonces-sino; en efecto:

$$\begin{aligned} & \mathbf{cond}\ \mathbf{C}\ M\ N \\ \equiv & \\ & (\lambda cpq.c\ p\ q)\ \mathbf{C}\ M\ N \\ = & \quad !\ \beta\text{–regla tres veces} \\ & \mathbf{C}\ M\ N \\ =_\beta & \\ & M \end{aligned}$$

$$\begin{aligned} & \mathbf{cond}\ \mathbf{F}\ M\ N \\ \equiv & \\ & (\lambda cpq.c\ p\ q)\ \mathbf{F}\ M\ N \\ = & \quad !\ \beta\text{–regla tres veces} \\ & \mathbf{F}\ M\ N \\ =_\beta & \\ & N \end{aligned}$$

4.1. Operaciones Lógicas (II)

Los operadores **no**, **y** y **o** se definen en la forma:

$$\begin{aligned} \mathbf{no} &= \lambda x.x\ \mathbf{F}\ \mathbf{C} \\ \mathbf{y} &= \lambda xy.x\ y\ \mathbf{F} \\ \mathbf{o} &= \lambda xy.x\ \mathbf{C}\ y \end{aligned}$$

Así,

$$\mathbf{y}\ \mathbf{C}\ \mathbf{F} \equiv (\lambda xy.x\ y\ \mathbf{F})\ \mathbf{C}\ \mathbf{F} = \mathbf{C}\ \mathbf{F}\ \mathbf{F} = \mathbf{F}$$

$$\mathbf{no}\ \mathbf{C} \equiv (\lambda x.x\ \mathbf{F}\ \mathbf{C})\ \mathbf{C} = \mathbf{C}\ \mathbf{F}\ \mathbf{C} = \mathbf{F}$$

4.2 Computabilidad

Definición (**Sistema de numerales**) *Un sistema* $\mathcal{N}$ *de numerales es una terna* $\mathcal{N} \equiv \langle \mathbf{N}, \mathbf{Cero}, \mathbf{Suc} \rangle$ *donde* $\mathbf{N}$ *es una sucesión de combinadores*

$$\mathbf{N} \quad \equiv \quad N_0, N_1, \ldots$$

y **Cero** *y* **Suc** *son dos términos (funciones) tales que*

$$\mathbf{Cero}\ N \quad = \quad \begin{cases} \mathbf{C} & , \textit{si } N = N_0 \\ \mathbf{F} & , \textit{si } N = N_1, N_2, \ldots \end{cases}$$

$$\mathbf{Suc}\ N_k \quad = \quad N_{k+1}$$

Definición (**Sistema adecuado de numerales**) *Un sistema de numerales es adecuado si existe una función* **Pred** *(predecesor) tal que:*

$$\mathbf{Pred}\ N_{k+1} = N_k$$

4.2 Computabilidad (II)

- Sistema adecuado de numerales de Church:
  - Co ≡ λfx.x
  - Cn+1 ≡ λfx.f^{n+1}(x)

  de forma que Suc ≡ λabc.b (a b c) es una función sucesor:

  Suc Cn $\twoheadrightarrow_\beta$ Cn+1

- A partir de éstas es fácil definir funciones test-cero [λn.(n(λx.F))C], predecesor y sucesor de forma que resulte un sistema adecuado de numerales. Tambien podemos representar otras operaciones algebraicas (donde m º n = λx.m(nx).

$$
\begin{array}{lcll}
+ & \equiv & \lambda mnfx.m\ f\ (n\ f\ x) & (suma) \\
\times & \equiv & \lambda mnf.m\ (n\ f) \equiv \lambda mn.m \circ n & (producto) \\
\uparrow & \equiv & \lambda xy.y\ x & (exponencial)
\end{array}
$$

4.2 Computabilidad (III)

- Consideremos ahora [n] $\in \Lambda$ a la representación en el λC del natural n $\in \mathbb{N}$, utilizando el sistema de Church.

Definición **(Función λ–definible)** *Una función numérica* $\varphi : \mathbb{N}^p \to \mathbb{N}$ *se dice* λ*–definible si existe un término* F *tal que*

$$\lceil \varphi(n_1, \ldots, n_p) \rceil \twoheadrightarrow F \ \lceil n_1 \rceil \ \ldots \ \lceil n_p \rceil$$

4.2 Computabilidad (IV)

NOTACIÓN *Si denotamos con*

$$\mathbf{n} = n_1, \ldots, n_p \qquad\qquad \lceil \mathbf{n} \rceil = \lceil n_1 \rceil \ \ldots \ \lceil n_p \rceil$$

la definición anterior se escribe:

$$\lceil \varphi(\mathbf{n}) \rceil \twoheadrightarrow F \ \lceil \mathbf{n} \rceil$$

Definición 4.5 (Funciones recursivas)

1. *Las* **funciones iniciales** *son las funciones* U_i^p, S^+, Z, *donde*

$$\begin{array}{lcl} U_i^p(n_0, \ldots, n_p) & = & n_i \\ S^+(n) & = & n+1 \\ Z(n) & = & 0 \end{array}$$

2. *Una clase de funciones numéricas* $\mathcal{C}$ *es* **cerrada para la composición** *si contiene la composición de funciones de la clase, es decir, las funciones de la forma:*

$$\chi(\phi_1(\mathbf{n}), \ldots, \phi_m(\mathbf{n})) \in \mathcal{C} \qquad\qquad si\ \chi, \phi_1, \ldots, \phi_m \in \mathcal{C}$$

3. *Una clase de funciones numéricas* $\mathcal{C}$ *es* **cerrada para la recursión primitiva** *si contiene a las funciones de la forma:*

$$\begin{array}{lcll} \phi(0, \mathbf{n}) & = & \chi(\mathbf{n}) & \\ \phi(k+1, \mathbf{n}) & = & \psi(\phi(k, \mathbf{n}), k, \mathbf{n}) & , \ para\ \chi, \psi \in \mathcal{C} \end{array}$$

4. *Una clase* $\mathcal{C}$ *es* **cerrada para la minimización** *si contiene las funciones de la forma:*

$$\phi(\mathbf{n}) = min \ \{ \ m \ \mid \ \varphi(\mathbf{n}, m) = 0 \ \} \qquad , \ con\ \varphi \in \mathcal{C}$$

5. *La clase* $\mathcal{R}$ *de las* **funciones recursivas** *es la menor clase cerrada para la composición, recursión primitiva y minimización que contiene a las funciones iniciales.*

Teorema de Kleene:

- La clase de las funciones recursivas es la misma que la clase de funciones λ-definibles.

- Por tanto, las funciones computables pueden representarse en el λC

El teorema de Kleene es la base de la evaluación de los programas funcionales; así, dada una función computable $f : \mathbb{N} \to \mathbb{N}$ y su λ–término asociado F:

$$\lceil f(\mathbf{n}) \rceil \twoheadrightarrow F \ \lceil \mathbf{n} \rceil$$

por el teorema de Chuch-Rosser, para obtener su valor ($\lceil f(\mathbf{n}) \rceil$) basta encontrar (si existe) la forma normal de $F \ \lceil \mathbf{n} \rceil$.

4.3. Puntos Fijos y recursión

Consideremos la siguiente definición de factorial:

$$FAC = \lambda n.CONDITIONAL\ (=\ n\ 0)\ 1\ (*\ n\ (FAC\ (-\ n\ 1))) \tag{3.0}$$

y puesto que $CONDITIONAL \equiv \lambda xyz.x\ y\ x$:

$$FAC = \lambda n.(=\ n\ 0)\ 1\ (*\ n\ (FAC\ (-\ n\ 1))) \tag{3.1}$$

Tal definición es incorrecta en el λC ya que las funciones son anónimas y no pueden referirse a sí mismas; para resolver tal problema podemos escribir el miembro de la derecha como resultado de una β–reducción sobre una función φ definida en la forma:

$$\varphi = \lambda f.((\lambda n.(\ =\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1)))$$

de forma que la ecuación 3.1 se escribe en la forma:

$$FACT \quad = \quad \varphi\ FACT \tag{3.2}$$

es decir, $FACT$ es un punto fijo de la función φ; por otro lado vemos que la solución a la ecuación 3.2 dependerá solamente de φ. Veamos una forma de obtener la solución de tales ecuaciones:

4.3. Puntos Fijos y recursión (II)

Teorema **(del punto fijo)** *Para cada* F *existe una expresión* X *tal que*

$$F\ X = X$$

Demostración: Sean $W \equiv \lambda x.F\ (x\ x),\ X \equiv W\ W$; tenemos

$$\begin{aligned} & X \\ = & \\ & W\ W \\ = & \\ & (\lambda x.F\ (x\ x))\ W \\ = & \\ & F\ (W\ W) \\ = & \\ & F\ X \end{aligned}$$

El teorema afirma que existe solución de la ecuación 3.2. Veamos ahora cómo buscarla de forma general.

4.3. Puntos Fijos y recursión (III)

Definición **(Combinador para puntos fijos)** *Un término M es un* combinador para puntos fijos *(*cppf*) si*

$$\forall F \ . \ F \in \Lambda \ . \ M \ F = F \ (M \ F)$$

Se observa que, si M es un cppf, dada una F podemos obtener siempre un punto fijo de F en la forma $M \ F$. El problema es, ¿existen tales combinadores?

Corolario *El combinador* $\mathbf{Y} \equiv \lambda f.(\lambda x.f \ (x \ x)) \ (\lambda x.f \ (x \ x))$ *es un cppf.*

Demostración: Sea $W \equiv \lambda x.F \ (x \ x)$ el término del teorema; entonces:

$$\begin{aligned} & \mathbf{Y} \ F \\ \equiv & \\ & W \ W \\ = & \\ & F \ (W \ W) \\ = & \\ & F \ (\mathbf{Y} \ F) \end{aligned}$$

4.3. Puntos Fijos y recursión (IV)

El combinador $\mathbf{Y}$ resuelve el problema de la recursión antes planteado, ya que la función factorial puede definirse en la forma:

$$FACT \quad = \quad \mathbf{Y}\ F$$

donde F es

$$F = \lambda f.((\lambda n.(=\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1)))$$

que es una definición no recursiva del factorial. Así tenemos una técnica para eliminar la recursión transformándola en iteración, pero tal iteración puede no terminar (si las iteraciones no tienen forma normal).

Tal solución es acertada desde un punto de vista matemático; sin embargo, debido a la ineficiencia de la implementación del combinador $\mathbf{Y}$ usando su λE, la mayoría de las implementaciones proporcionan $\mathbf{Y}$ como un constructor con la regla de reducción

$$\mathbf{Y}\ F \rightarrow F\ (\mathbf{Y}\ F)$$

En efecto, tenemos la siguiente reducción:

$$\begin{aligned}
& FACT\ 1 \\
\equiv\ & \\
& \mathbf{Y}\ F\ 1 \\
\rightarrow\ & \\
& F\ (\mathbf{Y}\ F)\ 1 \\
\equiv\ & \\
& (\lambda f.((\lambda n.(=\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1))))\ (\mathbf{Y}\ F)\ 1 \\
\rightarrow\ & \\
& ((\lambda n.(=\ n\ 0)\ 1\ (*\ n\ (\mathbf{Y}\ F\ (-\ n\ 1))))\ 1 \\
\rightarrow\ & \\
& *\ 1\ (\mathbf{Y}\ F(-\ 1\ 1)))) \\
\rightarrow\ & \\
& *\ 1\ (\mathbf{Y}\ F\ 0) \\
\rightarrow\ & \\
& *\ 1\ (F\ (\mathbf{Y}\ F)\ 0) \\
\equiv\ & \\
& *\ 1\ (\lambda f.((\lambda n.(=\ n\ 0)\ 1\ (*\ n\ (f\ (-\ n\ 1))))\ (\mathbf{Y}\ F)\ 0 \\
\rightarrow\ & \\
& *\ 1\ ((\lambda n.(=\ n\ 0)\ 1\ (*\ n\ (\mathbf{Y}\ F\ (-\ n\ 1))))\ 0 \\
\rightarrow\ & \\
& *\ 1\ 1 \\
\rightarrow\ & \\
& 1
\end{aligned}$$

4.4 Listas en el λC

Sean las funciones:

$$\begin{array}{lcl} \textbf{cons} & \equiv & \lambda abf.f\ a\ b \\ \textbf{vacia} & \equiv & \lambda xyz.y \\ \textbf{vacia?} & \equiv & \lambda x.x\ (\lambda abcd.d) \end{array}$$

Entonces es fácil ver que:

$$\textbf{vacia?}\ \textbf{vacia} = \textbf{C} \qquad\qquad \textbf{vacia?}\ (\textbf{cons}\ a\ b) = \textbf{F}$$

Por ejemplo:

$$\begin{array}{ll} & \textbf{vacia?}\ (\textbf{cons}\ a\ b) \\ \equiv & \\ & (\lambda x.x\ (\lambda abcd.d))\ (\textbf{cons}\ a\ b) \\ =_\beta & \\ & \textbf{cons}\ a\ b\ (\lambda abcd.d) \\ \equiv & \text{! definición de } \textbf{cons} \\ & (\lambda abf.f\ a\ b)\ a\ b\ (\lambda abcd.d) \\ = & \text{! } \beta\text{–regla tres veces} \\ & (\lambda abcd.d)\ a\ b \\ = & \text{! } \beta\text{–regla dos veces} \\ & \lambda cd.d \\ \equiv & \\ & \textbf{F} \end{array}$$

4.4 Listas en el λC (II)

Podemos identificar **cons** con el constructor de listas y consideraremos las siguientes equivalencias sintácticas:

$$\begin{array}{lcl} \textbf{vacia} & \equiv & [\,] \\ \textbf{cons}\ a\ b & \equiv & a : b \\ [x_1, \ldots, x_n] & \equiv & x_1 : [x_2, \ldots, x_n] \end{array}$$

entonces es fácil ver que:

$$\textbf{cab} \equiv \lambda x.x\ \mathbf{C} \qquad\qquad \textbf{col} \equiv \lambda x.x\ \mathbf{F}$$

son funciones que extraen la cabeza y la cola de una lista:

$$\textbf{cab}\ (\textbf{cons}\ p\ q) = p \qquad\qquad \textbf{col}\ (\textbf{cons}\ p\ q) = q$$

Por ejemplo:

$$\begin{array}{ll} & \textbf{cab}\ (\textbf{cons}\ p\ q) \\ \equiv & \text{! definición de } \textbf{cab} \\ & (\lambda x.x\ \mathbf{C})\ (\textbf{cons}\ p\ q) \\ =_\beta & \\ & \textbf{cons}\ p\ q\ \mathbf{C} \\ \equiv & \text{! definición de } \textbf{cons} \\ & (\lambda abf.f\ a\ b)\ p\ q\ \mathbf{C} \\ = & \text{! } \beta\text{–regla tres veces} \\ & \mathbf{C}\ p\ q \\ \equiv & \text{! definición de } \mathbf{C} \\ & (\lambda xy.x)\ p\ q \\ = & \text{! } \beta\text{–regla dos veces} \\ & p \end{array}$$

4.4 Listas en el λC (III)

Ahora veamos cómo definir la función de concatenación. En principio, nuestra función deberá verificar la ecuación:

$$CONCA\ x\ y = \textbf{vacia?}\ x\ y\ (\textbf{cons}\ (\textbf{cab}\ x)\ (CONCA\ (\textbf{col}\ x)\ y))$$

y si procedemos a eliminar la recursión de la misma forma que hicimos con el factorial, tendremos:

$$CONCA = \mathbf{Y}\ (\lambda f x y.\textbf{vacia?}\ x\ y\ (\textbf{cons}\ (\textbf{cab}\ x)\ (f\ (\textbf{col}\ x)\ y)))$$

Bibliografía:

- Razonando con Haskell: Una Introducción a la Programación Funcional;
  Blas C. Ruiz Jimenez, Fco. Gutierrez López, Pablo Guerrero García, José E. Gallardo Ruiz (2000).

- El λ-Cálculo (con tipos y sin tipos);
  Blas C. Ruiz Jimenez, Pablo Guerrero García (2002).

Muchas Gracias