CRA. Tema 4: $\lambda-$cálculo

José Enrique Morais San Miguel

Universidad de Alcalá

Listas

Codificar, en $\lambda-$cálculo, la lista $L = [1]$. Comprobar que no es vacía, que su cola es la lista vacía y que su cabeza es el número 1 usando los $\lambda-$términos definidos en los apuntes (página 60).

Empezamos la codificación de la lista dada que denotaremos por **l**.

$$\mathbf{l} \equiv \mathbf{cons}\ \underline{1}\ \mathbf{nil} \equiv \mathbf{pair\ false}\ (\mathbf{pair}\ \underline{1}\ \mathbf{nil}) \equiv$$

$$\equiv (\lambda xyf.fxy)\ \mathbf{false}\ (\mathbf{pair}\ \underline{1}\ \mathbf{nil}) \twoheadrightarrow$$

$$\twoheadrightarrow \lambda f.f\ \mathbf{false}\ (\mathbf{pair}\ \underline{1}\ \mathbf{nil}) \equiv \lambda f.f\ \mathbf{false}\ ((\lambda xyg.gxy)\ \underline{1}\ \mathbf{nil}) \twoheadrightarrow$$

$$\twoheadrightarrow \lambda f.f\ \mathbf{false}\ (\lambda g.g\ \underline{1}\ \mathbf{nil})$$

Listas

Una vez codificada la lista ($\mathbf{l} \equiv \lambda f.f\ \mathbf{false}\ (\lambda g.g\ \underline{1}\ \mathbf{nil})$), comprobamos que no es vacía:

$$\mathbf{null}\ \mathbf{l} \equiv \mathbf{fst}\ \mathbf{l} \equiv (\lambda p.p\ \mathbf{true})\ \mathbf{l} \rightarrow\ \mathbf{l}\ \mathbf{true} \equiv$$

$$\equiv (\lambda f.f\ \mathbf{false}\ (\lambda g.g\ \underline{1}\ \mathbf{nil}))\ \mathbf{true} \rightarrow$$

$$\rightarrow \mathbf{true}\ \mathbf{false}\ (\lambda g.g\ \underline{1}\ \mathbf{nil}) \twoheadrightarrow \mathbf{false}$$

Listas

Calculemos la cola de **l**

$$\textbf{tl}\ \textbf{l} \equiv (\lambda z.\textbf{snd}(\textbf{snd}\ z))\ \textbf{l} \to \textbf{snd}(\textbf{snd}\ \textbf{l}) \equiv$$

$$\equiv \textbf{snd}((\lambda p.p\ \textbf{false})\ \textbf{l}) \to \textbf{snd}(\textbf{l}\ \textbf{false}) \equiv$$

$$\equiv \textbf{snd}((\lambda f.f\ \textbf{false}\ (\lambda g.g\ \underline{1}\ \textbf{nil}))\ \textbf{false}) \to$$

$$\to \textbf{snd}(\textbf{false}\ \textbf{false}\ (\lambda g.g\ \underline{1}\ \textbf{nil})) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{snd}(\lambda g.g\ \underline{1}\ \textbf{nil}) \equiv (\lambda p.p\ \textbf{false})(\lambda g.g\ \underline{1}\ \textbf{nil}) \to$$

$$\to (\lambda g.g\ \underline{1}\ \textbf{nil})\ \textbf{false} \to$$

$$\to \textbf{false}\ \underline{1}\ \textbf{nil} \twoheadrightarrow \textbf{nil}$$

Finalmente, computamos la cabeza de **l**:

$$\mathbf{hd}\ \mathbf{l} \equiv (\lambda z.\mathbf{fst}(\mathbf{snd}\ z))\ \mathbf{l} \to \mathbf{fst}(\mathbf{snd}\ \mathbf{l}) \twoheadrightarrow {}^{a}$$

$$\twoheadrightarrow \mathbf{fst}(\lambda g.g\ \underline{1}\ \mathbf{nil}) \equiv$$

$$\equiv (\lambda p.p\ \mathbf{true})(\lambda g.g\ \underline{1}\ \mathbf{nil}) \to (\lambda g.g\ \underline{1}\ \mathbf{nil})\ \mathbf{true} \to$$

$$\to \mathbf{true}\ \underline{1}\ \mathbf{nil} \twoheadrightarrow \underline{1}$$

[a] La reducción **snd l** $\twoheadrightarrow \lambda g.g\ \underline{1}$ **nil** se ha hecho antes.

Combinador de punto fijo

Un combinador de punto fijo es un $\lambda-$término $\mathbf{Y}$ tal que $\mathbf{Y}\ F = F(\mathbf{Y}\ F)$ para todo término F. Esta terminología proviene del concepto de punto fijo para funciones: X es punto fijo de F si $F(X) = X$ (en este caso, $X = \mathbf{Y}\ F$). Por otro lado, un combinador es un término sin variables libres (también llamado término cerrado). Para codificar la recursión, F representa el cuerpo de la definición recursiva. La ley $\mathbf{Y}\ F = F(\mathbf{Y}\ F)$ permite desplegar F tantas veces como sea necesario.

El combinador de punto fijo paradójico

El combinador **Y** fue descubierto por Haskell B. Curry. Se define como

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

Es obvio que es un combinador. Veamos que posee la propiedad de punto fijo:

$$\begin{array}{rcl} \mathbf{Y}\ F & \rightarrow & (\lambda x.F(xx))(\lambda x.F(xx)) \\ & \rightarrow & F((\lambda x.F(xx))(\lambda x.F(xx))) \\ & = & F(\mathbf{Y}\ F) \end{array}$$

Hemos aplicado dos $\beta-$reducciones seguidas de una $\beta-$expansión. No hay reducción posible **Y** $F \twoheadrightarrow F(\mathbf{Y}\ F)$.

Ejemplo 1. El factorial

Definición recursiva:

$$\textbf{fact}\ N \quad = \quad \textbf{if}\ (\textbf{iszero}\ N)\ \underline{1}\ (\textbf{mult}\ N\ (\textbf{fact}(\textbf{pre}\ N)))$$

Para definir un $\lambda-$término que calcule el factorial, la llamada recursiva se reemplaza por una nueva variable g y se tiene, obviando el **if**:

$$\mathbf{F} \equiv \lambda gn.(\textbf{iszero}\ n)\ \underline{1}\ (\textbf{mult}\ n\ (g\ (\textbf{pre}\ n)))$$

Finalmente, se define

$$\textbf{fact} \equiv \mathrm{Y}\ \mathbf{F}$$

Cálculo de 2!

$$\textbf{fact}\ \underline{2} \equiv \textbf{YF}\ \underline{2} =^{a} \textbf{F}(\textbf{YF})\ \underline{2} \equiv \textbf{F}\ \textbf{fact}\ \underline{2} \twoheadrightarrow$$

$$\twoheadrightarrow (\textbf{iszero}\ \underline{2})\ \underline{1}\ (\textbf{mult}\ \underline{2}\ (\textbf{fact}\ (\textbf{pre}\ \underline{2}))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{false}\ \underline{1}\ (\textbf{mult}\ \underline{2}\ (\textbf{fact}\ (\textbf{pre}\ \underline{2}))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{fact}\ (\textbf{pre}\ \underline{2})) \twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{fact}\ \ \underline{1}) \equiv \textbf{mult}\ \underline{2}\ (\textbf{YF}\ \underline{1}) =$$

$$= \textbf{mult}\ \underline{2}\ (\textbf{F}(\textbf{YF})\ \underline{1}) \equiv \textbf{mult}\ \underline{2}\ (\textbf{F}\ \textbf{fact}\ \underline{1}) \twoheadrightarrow$$

[a] Aquí no cabe reducción alguna pues la igualdad es resultado de reducciones y expansiones. Si se utilizase el combinador de punto de fijo Θ de Turing, por ejemplo, podríamos sustituir la igualdad por $\twoheadrightarrow$.

Cálculo de 2!

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ ((\textbf{iszero}\ \underline{1})\ \underline{1}\ (\textbf{mult}\ \underline{1}\ (\textbf{fact pre}\ \underline{1}))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{false}\ \underline{1}\ (\textbf{mult}\ \underline{1}\ (\textbf{fact pre}\ \underline{1}))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ (\textbf{fact pre}\ \underline{1})) \twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ (\textbf{fact}\ \underline{0})) \equiv$$

$$\equiv \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ (\textbf{YF}\ \underline{0})) =$$

$$= \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ (\textbf{F}(\textbf{YF})\ \underline{0})) \equiv \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ (\textbf{F fact}\ \underline{0})) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ ((\textbf{iszero}\ \underline{0})\ \underline{1}\ (\textbf{mult}\ \underline{0}\ (\textbf{fact}\ (\textbf{pre}\ \underline{0}))))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{mult}\ \underline{2}\ (\textbf{mult}\ \underline{1}\ \underline{1}) \twoheadrightarrow \textbf{mult}\ \underline{2}\ \underline{1} \twoheadrightarrow \underline{2}$$

Ejemplo 2. Concatenación de listas

Definición recursiva:

$$\textbf{append}\ Z\ W \quad = \quad \textbf{if}\ (\textbf{null}\ Z)\ W\ (\textbf{cons}\ (\textbf{hd}\ Z)\ (\textbf{append}(\textbf{tl}\ Z)\ W))$$

Para definir un $\lambda-$término que calcule la concatenación de listas, la llamada recursiva se reemplaza por una nueva variable g y se tiene, obviando el **if**:

$$\textbf{F} \equiv \lambda gzw.(\textbf{null}\ z)\ w\ (\textbf{cons}\ (\textbf{hd}\ z)\ (g\ (\textbf{tl}\ z)w))$$

Finalmente, se define

$$\textbf{append} \equiv \mathrm{Y}\ \textbf{F}$$

Ejemplo 3. Longitud de listas

Definir un $\lambda-$término **long** *que aplicado a una lista codificada en $\lambda-$cálculo, calcule la longitud de la misma. Utilizarlo para calcular la longitud de la lista $L = [1]$.* (Este ejercicio está extraído del examen de 21 de mayo de 2018).

Ejemplo 3. Longitud de listas: solución

Definición recursiva de la longitud:

$$\textbf{long}\ L \quad = \quad \textbf{if}\ (\textbf{null}\ L)\ \underline{0}\ (\textbf{suc}\ (\textbf{long}\ (\textbf{tl}\ L)))$$

Para definir un $\lambda-$término que calcule la longitud de una lista, la llamada recursiva se reemplaza por una nueva variable g y se tiene, obviando el **if**,:

$$\mathbf{G} \equiv \lambda gl.(\textbf{null}\ l)\ \underline{0}\ (\textbf{suc}(g\ (\textbf{tl}\ l)))$$

Finalmente, se define

$$\textbf{long} \equiv \mathrm{Y}\ \mathbf{G}$$

Ejemplo 3. Longitud de listas: solución (y 2)

La lista [1] codificada en $\lambda-$cálculo es, como hemos visto antes,

$$\mathbf{l} \equiv \lambda f.f\ \mathbf{false}\ (\lambda g.g\ \underline{1}\ \mathbf{nil})$$

Calculemos su longitud:

$$\mathbf{long\ l} \equiv \mathbf{YG\ l} = \mathbf{G(YG)\ l} \equiv \mathbf{G\ long\ l} \twoheadrightarrow$$

$$\twoheadrightarrow (\mathbf{null\ l})\ \underline{0}\ (\mathbf{suc}\ (\mathbf{long(tl\ l)})) \twoheadrightarrow$$

$$\twoheadrightarrow \mathbf{false}\ \underline{0}\ (\mathbf{suc}\ (\mathbf{long(tl\ l)})) \twoheadrightarrow \mathbf{suc}\ (\mathbf{long}\ (\mathbf{tl\ l})) \twoheadrightarrow$$

$$\twoheadrightarrow \mathbf{suc}\ (\mathbf{long\ nil}) \equiv$$

Ejemplo 3. Longitud de listas: solución (y 3)

$$\equiv \textbf{suc}\ (\textbf{YG nil}) = \textbf{suc}\ (\textbf{G}(\textbf{YG})\ \textbf{nil}) \equiv \textbf{suc}\ (\textbf{G long nil}) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{suc}\ ((\textbf{null nil})\ \underline{0}\ (\textbf{suc}\ (\textbf{long}(\textbf{tl nil})))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{suc}\ (\textbf{true}\ \underline{0}\ (\textbf{suc}\ (\textbf{long}(\textbf{tl nil})))) \twoheadrightarrow$$

$$\twoheadrightarrow \textbf{suc}\ \underline{0} \twoheadrightarrow \underline{1}$$

El combinador de Turing

El combinador Θ descubierto por Turing está dado por:

$$\begin{array}{ccc} A & \equiv & \lambda xy.y(xxy) \\ \Theta & \equiv & AA \end{array}$$

En este caso, al contario de lo que ocurría con el combinador paradójico, sí que se tiene la reducción $\Theta\, F \twoheadrightarrow F(\Theta\, F)$:

$$\Theta\, F \equiv AAF \twoheadrightarrow F(AAF) \equiv F(\Theta\, F)$$

Ejemplo 4. Resto de la división euclídea

Este problema está sacado del examen extraordinario de junio de 2012

Probar que el $\lambda-$término Θ definido por:

$$A \equiv \lambda xy.y(xxy)$$

$$\Theta \equiv AA$$

es un combinador de punto fijo. Usando Θ, definir un $\lambda-$término que calcule el resto de la división de dos números naturales positivos codificados a la Church. (**Nota**: No hace falta redefinir todos los $\lambda-$términos involucrados que ya hayan sido definidos en clase.)

Ejemplo 4. Resto de la división euclídea: solución

Como hemos hecho en casos anteriores, empezamos con la definición recursiva del resto:

$$\textbf{resto}\ m\ n = (\textbf{menor}\ m\ n)\ m\ (\textbf{resto}\ (\textbf{sub}\ m\ n)\ n)$$

La resta está definida en los apuntes, pero no hay una definición de un $\lambda-$término que decida si un número es estrictamente menor que otro. Si suponemos que disponemos del $\lambda-$término **menor**, podemos definir

$$F \equiv \lambda gmn.(\textbf{menor}\ m\ n)\ m\ (g\ (\textbf{sub}\ m\ n)\ n)$$

y, por lo tanto,

$$\textbf{resto} \equiv \Theta\ F$$

Ejemplo 4. Resto de la división euclídea: solución (y2)

Definición de menor

$$\textbf{menor} \equiv \lambda mn.\textbf{and}\ (\textbf{iszero}(\textbf{sub}\ m\ n))\ (\textbf{not}(\textbf{iszero}\ (\textbf{sub}\ n\ m)))$$

Ejemplo 5. Máximo común divisor

Este problema está sacado del examen extraordinario de junio de 2015

Probar que el $\lambda-$término Θ definido por:

$$A \equiv \lambda xy.y(xxy)$$

$$\Theta \equiv AA$$

es un combinador de punto fijo. Usando Θ, definir un $\lambda-$término que calcule el máximo común divisor de dos números naturales codificados a la Church. (**Nota**: No hace falta redefinir todos los $\lambda-$términos involucrados que ya hayan sido definidos en clase.)

Ejemplo 5. Máximo común divisor: solución

Para definir el $\lambda-$término que se pide no hay más que pensar en el Algoritmo de Euclides:

$$\mathbf{mcd}\; m\; n\; = (\mathbf{iszero}\; n)\; m\; (\mathbf{mcd}\; n\; (\mathbf{resto}\; m\; n))$$

Ahora, si definimos **G** como

$$\mathbf{G} \equiv \lambda gmn.(\mathbf{iszero}\; n)\; m\; (g\; n\; (\mathbf{resto}\; m\; n))$$

el $\lambda-$término

$$\mathbf{mcd} \equiv \Theta\; \mathbf{G}$$

cumple los requerimientos pedidos. (**Nota**: En el examen había que definir el $\lambda-$término **resto** porque no está en los apuntes. No lo hemos hecho aquí porque lo acabamos de definir anteriormente).

Ejemplo 6

Este problema está sacado de la PEI 2 del 17 de mayo de 2019

Definir, usando **Y**, un $\lambda-$término **sub** que, aplicado a dos listas de números naturales a la Church codificadas en $\lambda-$cálculo, decida si la primera es subconjunto de la segunda, vistas ambas listas como conjuntos. No es necesario redefinir los $\lambda-$términos utilizados ya definidos en clase

Ejemplo 6. Solución

La definición recursiva de la función **sub** es:

$$\textbf{sub}\ z\ w\ =\ (\textbf{null}\ z)\ \textbf{true}\ ((\textbf{member}\ (\textbf{hd}\ z)\ w)\ (\textbf{sub}\ (\textbf{tl}\ z)\ w)\ \textbf{false})$$

Como debemos usar el combinador de punto fijo **Y**, notando

$$G \equiv \lambda gzw.\ (\textbf{null}\ z)\ \textbf{true}\ ((\textbf{member}\ (\textbf{hd}\ z)\ w)\ (g\ (\textbf{tl}\ z)\ w)\ \textbf{false}),$$

podemos definir $\textbf{sub} \equiv \textbf{YG}$.
Queda por definir un $\lambda-$término **member**.

Ejemplo 6. Solución (y 2)

La definición recursiva de la función **member** es:

$$\textbf{member}\; n\, l \;=\; (\textbf{null}\; l)\; \textbf{false}\; ((\textbf{igual}\; n\, (\textbf{hd}\; l))\; \textbf{true}\; (\textbf{member}\; n\, (\textbf{tl}\; l)))$$

Como debemos usar el combinador de punto fijo **Y**, notando

$$\mathbf{G} \equiv \lambda gnl.\, (\textbf{null}\; l)\; \textbf{false}\; ((\textbf{igual}\; n\, (\textbf{hd}\; l))\; \textbf{true}\; (g\; n\, (\textbf{tl}\; l)))$$

podemos definir **member** $\equiv$ **YG**.
Aún queda por definir el $\lambda-$término **igual**, que determina si dos numerales de Church son iguales, pero esto no requiere recursión:

$$\textbf{igual} \equiv \lambda nm.\textbf{and}\; (\textbf{iszero}\; (\textbf{sub}\; n\; m))\; (\textbf{iszero}\; (\textbf{sub}\; m\; n))$$