# The $\lambda$-calculus

Foundations for Programming Languages
MASTER IN SOFTWARE AND SYSTEMS
Universidad Politécnica de Madrid/IMDEA Software Institute

November 15, 2017

**To be turned in by December 1, 2017. Send them to me by mail to `jmarino@fi.upm.es`.**

## Fundamentos

**Exercise 1.** Clasifica los siguientes términos según $\alpha$-equivalencia: $\lambda x.x\ y$, $\lambda x.x\ z$, $\lambda y.y\ z$, $\lambda z.z\ z$, $\lambda z.z\ y$, $\lambda f.f\ y$, $\lambda f.f\ f$, $\lambda y.\lambda x.x\ y$, $\lambda z.\lambda y.y\ z$ .

**Exercise 2.** Proporciona un término $\alpha$-equivalente donde cada abstracción sea sobre un nombre de variable distinto: $\lambda x.((x\ (\lambda y.x\ y))(\lambda x.x))(\lambda y.y\ x)$ .

**Exercise 3.** Reduce a forma normal el término $(\lambda x.(\lambda y.x\ y))\ y$ .

**Exercise 4.** Dados los siguientes *combinadores*: $\mathsf{S} = \lambda x.\lambda y.\lambda z.(x\ z)(y\ z)$ , $\mathsf{K} = \lambda x.\lambda y.x$ e $\mathsf{I} = \lambda x.x$, demuestra la equivalencia $\mathsf{S}\ \mathsf{K}\ \mathsf{K} = \mathsf{I}$.

## Matemática constructiva en el cálculo $\lambda$

**Exercise 5.** Usando la codificación de booleanos en el cálculo $\lambda$ puro de Church, define $\lambda$-términos en forma normal CONJ, DISY y NEG para representar conjunción, disyunción y negación, respectivamente. (Pista: define un $\lambda$-término COND que represente un *if-then-else* y aplica $\beta$-reducción.)

**Exercise 6.** Usando la codificación de Church de los números naturales define suma, multiplicación y exponentiation.

**Exercise 7.** Piensa una codificación de pares en el cálculo $\lambda$. Proporciona $\lambda$-términos PAIR (para construir un par) y las proyecciones FST y SND.

**Exercise 8.** Define las funciones predecesor y sustracción de naturales. (Pista: apóyate en los pares del ejercicio anterior.)

**Exercise 9.** Supongamos que vamos a representar la lista $[a_1, a_2 \ldots a_n]$ en el cálculo lambda mediante el término

$$\lambda f.\lambda x.f\ a_1\ (f\ a_2(\ldots f\ a_n\ x))$$

Define:

a. NIL, el constructor de la lista vacía,

b. APP, una función para concatenar dos listas,

c. HD, que proporciona el primer elemento de una lista no vacía, y

d. una función ISEMPTY para comprobar si una lista es vacía.

## El cálculo $\lambda$ simplemente tipado

**Exercise 10.** Da una demostración en $\mathrm{TA}_\lambda$ del siguiente tipado para la composición de funciones:

$$\vdash \lambda f.\lambda g.\lambda x.f\,(g\ x) : (b \to c) \to (a \to b) \to (a \to c)$$

**Exercise 11.** Proporciona un tipado para el combinador $\mathsf{S} = \lambda x.\lambda y.\lambda z.(x\ z)(y\ z)$.

**Exercise 12.** Proporciona un tipado para el término $\mathsf{I} = (\lambda x.x)(\lambda x.x)$.

**Exercise 13.** Proporciona tipos para los términos del ejercicio 5, acompañados de su derivación correspondiente.

**Exercise 14.** Demuestra que si $\Gamma \vdash M : \sigma$ entonces $\Gamma[\rho \rightarrowtail \tau] \vdash M : \sigma[\rho \rightarrowtail \tau]$ .

**Exercise 15.** Demuestra que si $\Gamma, x : \sigma \vdash M : \tau$ y $\Gamma \vdash N : \sigma$ entonces $\Gamma \vdash M[x \rightarrowtail N] : \tau$ .

**Exercise 16.** Demuestra que si $M \rightsquigarrow_\beta^* M'$ y $\Gamma \vdash M : \sigma$ entonces $\Gamma \vdash M' : \sigma$ .

## El cálculo $\lambda$ en Haskell

**Exercise 17.** En Haskell, define funciones `boolChurch` y `boolUnchurch` para traducir booleanos Haskell a booleanos de Church y viceversa. Úsalas para comprobar la corrección de tus soluciones al ejercicio 5.

**Exercise 18.** Análogamente al ejercicio anterior, define en Haskell funciones para convertir entre las representaciones Haskell y Church de los números naturales y comprueba la corrección de tus soluciones al ejercicio 6.

**Exercise 19.** Define un tipo de datos Haskell para representar la sintaxis abstracta del cálculo $\lambda$.

**Exercise 20.** Apoyándote en el ejercicio anterior, define una función Haskell para obtener las variables libres de un término lambda.

**Exercise 21.** Apoyándote en los ejercicios anteriores, define una función Haskell que realice la *sustitución sin captura*.

**Exercise 22.** Apoyándote en los ejercicios anteriores, define una función Haskell que realice la $\beta$-reducción.

**Exercise 23.** Apoyándote en los ejercicios anteriores, define una función Haskell que reduzca un término lambda a forma $\beta$-normal. (Su terminación solo estará garantizada en el caso de términos tipables en $\mathrm{TA}_\lambda$.)