Chapter 1: All You Need is Lambda

Contexto

> *Even the greatest mathematicians, the ones that we would put into our mythology of great mathematicians, had to do a gread deal of leg work in order to get to the solution in the end.*
>
> *Daniel Tammett*

- El cálculo lambda es la base de la programación funcional. Su desarrollo fue gracia a Alonzo Church en los 30´s, y es un modelo de computación equivalente a las máquinas de Turing. (**Nota**: Es, además, capaz de expresar toda la lógica de primer orden y más. A esta área se le llama lógica combinatoria.)
- La motivación para estudiar programación funcional en un lenguaje *puro* como Haskell, es la búsqueda de transparencia referencial. Esta base permite un mayor grado de abstracción y composibilidad.

Calculo lambda

- Existen 3 términos o "componentes" en el cálculo lambda: expresiones, variables y abstracciones.
  1. Una *expresión* puede referirse a cualquiera de los demás términos, o a una combinación de ellos.
  2. Una *variable* es un nombre asociado a un potencial input de una función.
  3. Una *abstracción* o *lambda* es una función. Este término tiene la siguiente estructura:

     $$\lambda x.E$$

     Donde:

     - Llamamos *head* a todo lo que precede al punto, es decir, al símbolo λ junto a una variable.
     - Llamamos *body* a todo lo que sucede al punto, es decir, la expresión E.

     La variable de la cabeza liga todas las ocurrencias de dicha variable en el cuerpo; por lo que, al aplicar la función a un argumento, cada x en el cuerpo de la función tendrá el valor de tal argumento.
- Definimos el término *alpha equivalence* para referirnos al hecho de qué, dada una abstracción del cálculo lambda, las variables de la cabeza no tienen importancia semántica; por lo que salvo variables libres (noción que se verá más adelante), existe una equivalencia entre términos lambda al reemplazar el nombre de la variable usada en la cabeza, y todas sus ocurrencias ligadas en el cuerpo. $\lambda x.x \overset{\alpha}{\equiv} \lambda z.z$.
- Llamamos al proceso de aplicar una función a un argumento *beta reduction*, proceso que consiste en reemplazar todas las instancias del cuerpo ligadas a la variable de la cabeza, por el argumento; para luego remover la cabeza de la abstracción. Ejemplo: $(\lambda x.x)2 \Rightarrow 2$ o

  $$(\lambda x.x)(\lambda y.y)$$
  $$[x := (\lambda y.y)]$$
  $$\lambda y.y$$

  Nota: Las aplicaciones en el cálculo lambda son *asociativas a izquierda.*
- Introducimos la noción de *variable libre* como aquellas variables en el cuerpo de una abstracción, que no están en la cabeza.

Note que esto introduce un caso sobre el cuál tener cuidado al aplicar alpha equivalencia ya que esta no aplica a variables libres, es decir, $\lambda x.xz \overset{\alpha}{\not\equiv} \lambda x.xy$ porque x y y pueden ser expresiones diferentes. Sin embargo $\lambda x.xz \overset{\alpha}{\equiv} \lambda a.az$

- Decimos que una computación consiste en una expresión lambda inicial (función + input) y una secuencia finita de términos lambda donde cada una representan la beta reducción de la expresión anterior. Esta comptuación termina cuando no hay más heads para evaluar, o más argumentos a los cuáles aplicarlas.
- Cada lambda puede tener un solo parámetro y recibir un único argumento. Esto nos permite representar funciones multivariadas como funciones de múltiples heads anidadas.

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

  Al proceso de convertir una función multivariada en múltiples funciones univariadas anidadas, se le llama *Currying*. Además, la evaluación sobre este tipo de funciones se hace aplicando primero los argumentos a la cabeza (reducible) más a la izquierda, y continuando a partir de ahí.
- Haremos alución a la *forma normal* de una expresión para referirnos a la *beta normal form*, es decir, a la expresión resultante cuando no se puede aplicar ninguna beta reducción a los términos. Esto corresponde en computación a una ejecución completa, o a una expresión completamente evaluada.
- Definimos los *combinadores* como lambdas sin términos libres. Este nombre es natural ya qué, dado un lambda sin variables libres, lo más que puede hacer con las variables de su cabeza es combinarlas.
- Es interesante resaltar el hecho de que no todas las expresiones son reducibles a una forma beta normal, ya que *divergen*. Un ejemplo de expresión con este comportamiento es

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

Glosario de términos

Lambda abstraction Es una función anónima o término lambda.

Aplicación Procedimiento con el que se evalúan o reducen lambdas. Note que son las funciones las que se aplican a los argumentos y no al revés, ya qué, una vez se reemplaza el argumento en la función, la función se *"consume"*.

Normal order Es la estrategia de evaluación común en el cálculo lambda. Este orden consiste en evaluar primero los lambdas "más afuera y a la izquierda" que se pueda, evaluando los términos anidados una vez que se queda sin argumentos para aplicar. Aunque esta es la estrategia estándar en el cálculo lambda, **no** es la usada por Haskell.