Apunte de Introducción a los Algoritmos

Pedro Sánchez Terraf*

26 de marzo de 2017

1. Guía de clases

1.1. Definiciones de funciones

La construcción básica en un programa funcional es la definición de funciones. Para definir una función, necesito decir qué "come" y qué "devuelve". Lo que "come" o consume una función se denominan **argumentos** y lo que "devuelve" o su resultado se denomina su **valor** para dichos argumentos. Como un ejemplo muy zonzo, consideremos la función *suma* que suma dos enteros.

$$\begin{aligned} &suma : Int \to Int \to Int \\ &suma.x.y \doteq x + y \end{aligned} \tag{1}$$

Notemos que una definición de función como la (1) introduce dos cosas:

1. Un nuevo símbolo que nombra la función (en este caso, f) y qué tipo tiene.
2. Una nueva igualdad, que es verdadera para todos los valores de las variables (i.e., la igualdad $suma.x.y = x + y$).

Decimos que la igualdad $suma.x.y = x + y$ introducida por la definición (1) es **válida**: es verdadera para todos los valores de sus variables. En tal sentido, funciona como un axioma o un teorema (por ejemplo, como la conmutatividad de la suma, $a + b = b + a$).

Como las definiciones introducen igualdades (y la igualdad es **simétrica**, i.e. $a = b$ implica $b = a$), podemos pasar de la expresión $suma.x.y$ a $x + y$ y viceversa. Sin embargo le pondremos dos nombres distintos a cada una de los procesos. Por ejemplo, si calculamos $suma.9.16$,

$$\begin{aligned} & suma.9.16 \\ = \{ \;& \text{Definición de } suma \; \} \\ & 9 + 16 \\ = \{ \;& \text{Aritmética} \; \} \\ & 25, \end{aligned}$$

estamos usando la igualdad introducida por (1) de izquierda a derecha. En tal caso, decimos que estamos **desplegando** la definición. Por otro lado, si la usamos de derecha a izquierda, "haciendo aparecer" la función *suma* como a continuación:

$$\begin{aligned} & 256 \\ = \{ \;& \text{Aritmética} \; \} \\ & 128 + \underline{64 + 64}, \\ = \{ \;& \text{Definición de } suma \; \} \\ & 128 + suma.64.64 \end{aligned}$$

decimos que estamos **plegando** la definición.

*CIEM-FAMAF.

1.2. Patrones

Una herramienta muy poderosa en programación funcional es la utilización de *patrones*.

Un **patrón** es la forma que tiene el argumento de una función. La utilidad de los patrones radica en que podemos estipular qué forma tienen los argumentos. Por ejemplo, para la función

$$\begin{aligned} &f : (Int, Int) \to Int \\ &f.(x, y) \doteq x * y \end{aligned} \tag{2}$$

que toma un par de enteros y devuelve la suma de sus componentes, el patrón es $\boldsymbol{(x, y)}$.

Comentario. La función f **no es la misma** que la función *multiplicar* de la Guía 1. Aparentemente hace lo mismo, pero su tipo es distinto. *multiplicar* tiene dos argumentos, y f tiene uno que es un par (tupla de dos componentes).

Cada vez que le demos algo de comer a f, deberemos "emparejarlo" (*match*, en inglés) con el patrón antes de aplicar la función. Por ejemplo, si queremos calcular $f.(4 * 2, 3)$, debemos entender cómo se corresponde el argumento $(4 * 2, 3)$ con el patrón $\boldsymbol{(x, y)}$:

$$\begin{array}{ccccc} (& 4*2 & , & 3 &) \\ & \downarrow & & \downarrow & \\ (& \boldsymbol{x} & , & \boldsymbol{y} &). \end{array}$$

Entonces, a la variable x le corresponde el valor $4 * 2$ y a y le corresponde 3. Entonces podemos desplegar la definición de f para obtener el resultado:

$$\begin{array}{ll} & \underline{f.(4 * 2, 3)} \\ = \{ & \text{Definición de } f \text{ (de acuerdo al patrón, } x = 4 * 2 \text{ e } y = 3) \ \} \\ & 4 * 2 * 3 \\ = \{ & \text{Aritmética } \} \\ & 24. \end{array}$$

Veamos un ejemplo ligeramente distinto. Queremos calcular $f.(4 + 1, 3)$. Si lo hacemos de la siguiente manera, estará **mal**:

$$\begin{array}{ll} & \underline{f.(4 + 1, 3)} \\ = \{ & \text{Definición de } f \ \} \\ & 4 + 1 * 3 \\ = \{ & \text{Aritmética } \} \\ & 7. \end{array}$$

El resultado debería haber sido 15. Lo que sucedió aquí es que siempre que emparejamos dos expresiones, conviene poner paréntesis para no introducir efectos secundarios indeseados. En este caso, la expresión que se empareja con x es $4 + 1$, pero para estar seguros que se considerará como un paquete cerrado, debemos ponerlo entre paréntesis:

$$\begin{array}{ll} & \underline{f.(4 + 1, 3)} \\ = \{ & \text{Definición de } f \ \} \\ & (4 + 1) * 3 \\ = \{ & \text{Aritmética } \} \\ & 15. \end{array}$$

Dos patrones que utilizaremos mucho son los que describen números naturales (que denominamos *Nat* y contienen al 0 como $\mathbb{N}_0$) y listas. Para los primeros, los patrones son 0 y n. Lo veamos con un ejemplo de función más.

$$
\begin{aligned}
& g : Nat \to Int \\
& g.0 \doteq 1 && (3) \\
& g.(n+1) \doteq n+2. && (4)
\end{aligned}
$$

De acuerdo a su definición, g sólo puede consumir 0 o algo que se corresponda con el patrón $(n+1)$.

Ejercicio 1.1. Convencerse que un número natural es o bien 0 o se puede emparejar con el patrón $n+1$.

Supongamos que queremos calcular $g.6$. No podemos aplicar la definición de g a esta expresión, porque no le estamos dando de comer ni 0 ni una expresión de la forma $(n+1)$. Pero hacer un poco de aritmética antes y luego podremos aplicar el caso (4) de la definición de g:

$$
\begin{aligned}
& g.\underline{6} \\
= \{ & \text{ Aritmética } \} \\
& g.(5+1) \\
= \{ & \text{ Definición de } g \ \} \\
& 5+2 \\
= \{ & \text{ Aritmética } \} \\
& 6.
\end{aligned}
$$

En el medio de la prueba, al emparejar la expresión $(5+1)$ con el patrón $(n+1)$ deducimos que $n = 5$. Luego, al desplegar la definición de g pasamos al término derecho de la ecuación (4) obteniendo $5+2$.

Ejercicio 1.2. Deducir qué hace la función g.

El caso de las listas es muy similar. Toda lista es o bien vacía (es decir, igual a $[\,]$) o tiene un primer elemento, así que puede escribirse de la forma $x \triangleright xs$. Un ejemplo es la función *head*:

$$
\begin{aligned}
& head : [A] \to [A] \\
& head.(x \triangleright xs) \doteq x && (5)
\end{aligned}
$$

Ejercicio 1.3. Calcular $head.[1,2]$, $head.$[`"hola"`, `"chau"`] y $head.[[\,], [1,2]]$.

Técnica de definición de funciones: composición (*modularización*). Descomponer un problema en partes.

Trabajo en clase

1. Definir la función *esBisiesto* : $Num \to Bool$, que indica si un año es bisiesto. Un año es bisiesto si es divisible por 400 o es divisible por 4 pero no es divisible por 100. (usar mód).

2. Definir la función *max3* : $Num \to Num \to Num \to Num$, que dados tres números devuelve el mayor de los tres (usar máx).

Tarea

Entregar por escrito:

1. $mayor3 : (Int, Int, Int) \to (Bool, Bool, Bool)$, que dada una una terna de enteros devuelve una terna de valores booleanos que indica si cada uno de los enteros es mayor que 3.

   Por ejemplo: $mayor3.(1, 4, 3) = (False, True, False)$ y $mayor3.(5, 1984, 6) = (True, True, True)$.

2. Definir la función $dispersion : Num \to Num \to Num \to Num$, que toma los tres valores y devuelve la diferencia entre el más alto y el más bajo. ($Ayuda$: usar $max3$ y $min3$. De esa forma se puede definir $dispersion$ sin hace análisis por casos.)