Práctica 6
Programación Funcional - Básica

Algoritmos y Estructura de Datos I

Primer Cuatrimestre 2010

Aclaraciones: Los ejercicios de esta práctica piden la definición de funciones al estilo de los programas funcionales de Haskell. En general, no hay una única manera de definir una función. Sin embargo, nos interesa que las soluciones elegidas sean claras, comprensibles, compactas (breves, de ser posible), completas (que contemplen todos los casos de interés) y declarativas (que estén más cerca del qué y no del cómo).

Cuando resulte conveniente, se pueden definir funciones auxiliares para simplificar la definición de la función principal. También se pueden usar funciones definidas previamente en otros ejercicios.

Dar explícitamente el tipo de una función (el tipo de su domino y su imagen) permite detectar errores tempranamente. No hay que olvidar que el valor que devuelve una función depende sólo de sus argumentos.

A pesar de que en Haskell existen las definiciones de listas por comprensión, en esta materia no vamos a permitir su uso.

Ejercicio 1. Definir la función parcial $f :: \mathit{Integer} \to \mathit{Integer}$, definida por extensión de la siguiente manera:

$$\begin{aligned} f(1) &= 8 \\ f(4) &= 131 \\ f(16) &= 16 \end{aligned}$$

cuya especificación es la siguiente:
problema f($x : \mathbb{Z}$) = $res : \mathbb{Z}${

requiere : $x == 1 \lor x == 4 \lor x == 16$;
asegura : $(x == 1 \to res == 8) \land (x == 4 \to res == 131) \land (x == 16 \to res == 16)$;

}

Ejercicio 2. Análogamente, especificar y definir la función parcial $g :: \mathit{Integer} \to \mathit{Integer}$:

$$\begin{aligned} g(8) &= 16 \\ g(16) &= 4 \\ g(131) &= 1 \end{aligned}$$

Ejercicio 3. A partir de las funciones definidas en los ejercicios 1 y 2, definir las funciones parciales $h = f \circ g$ y $k = g \circ f$

Ejercicio 4. Especificar y definir las siguientes funciones de enteros en enteros:

1. $f(n) = \begin{cases} n & \text{si } n \leq 10 \\ n+1 & \text{si } n > 10 \end{cases}$
2. $g(n) = \begin{cases} 1 & \text{si } n \text{ es primo} \\ 0 & \text{si } n \text{ no es primo} \end{cases}$
3. $\sum_{i=0}^{n} g(i)$

Ejercicio 5. Especificar y definir la función f dada por $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{3^3}$, etc.

Ejercicio 6. Especificar la función *divisiblePorTres* :: *Integer* → *Bool*, que indica si un número es divisible por tres o no, y definirla usando el criterio de divisibilidad por tres: *un número es divisible por tres si y sólo si la suma de sus cifras también lo es.*

Ejercicio 7. Se tiene definido el tipo *Nat* con los siguientes elementos constructores:

- 0 (cero)
- *Suc*(*Nat*) (el sucesor del número pasado como parámetro)

Empleando solamente 0 y *Suc*, definir las siguientes funciones:

1. *igual* :: *Nat* → *Nat* → *Bool*
2. *suma* :: *Nat* → *Nat* → *Nat*
3. *producto* :: *Nat* → *Nat* → *Nat*
4. *potencia* :: *Nat* → *Nat* → *Nat*
5. *menor* :: *Nat* → *Nat* → *Bool*
6. *resta* :: *Nat* → *Nat* → *Nat* (es 0 si el minuendo es menor que el sustraendo)
7. *división* :: *Nat* → *Nat* → *Nat* (división entera; por ejemplo, *división* 7 3 = 2)
8. *resto* :: *Nat* → *Nat* → *Nat*
9. *mcd* :: *Nat* → *Nat* → *Nat* (máximo común divisor)
10. *factorial* :: *Nat* → *Nat*
11. *nFibonacci* :: *Nat* → *Nat* (*n*-ésimo término de la sucesión de Fibonacci)

Ejercicio 8. Se tiene el tipo enumerado *Día*, formado por las constantes *lunes*, *martes*, *miércoles*, *jueves*, *viernes*, *sábado* y *domingo*. Enriquecerlo, definiendo las siguientes operaciones:

1. *siguiente* :: *Día* → *Día*
2. *esLaborable* :: *Día* → *Bool* (lunes a viernes)
3. *esFinDeSemana* :: *Día* → *Bool* (sábado o domingo)