Práctico 1 de Introducción a la Computación

UdelaR/FCien/CMat

21 al 28 de agosto de 2013

1. Usando el tipo de datos **Bool** –primitivo en Haskell– se pide:

   (a) Definir en Haskell las funciones que calculen las operaciones $\wedge$, $\vee$, $\neg$, $\Rightarrow$ sobre el tipo de datos **Bool**.

   $$\textbf{data Bool} = \texttt{False} \mid \texttt{True}$$

   ```
   and  :: Bool→Bool→Bool
   or   :: Bool→Bool→Bool
   neg  :: Bool→Bool
   implies :: Bool→Bool→Bool
   ```

   (b) Explorar en qué argumentos las definiciones implementadas son estrictas y en qué argumentos no. Modificar las definiciones para que los argumentos estrictos sean no estrictos y los no estrictos sean estrictos. ¿Tiene sentido definirlas para que no sean estrictas en ningún argumento?

   (c) Definirlas para que sean estrictas en todos sus argumentos.

   (d) Definir una función de tipo

   $$\texttt{(eq)} :: \textbf{Bool} \to \textbf{Bool} \to \textbf{Bool}$$

   que calcule la igualdad en el tipo **Bool**. Calcular

   $$\bot \texttt{ eq } \bot \quad \text{y} \quad \bot \texttt{ eq False}$$

   ¿En qué argumentos esta función es estricta?

   (e) Considerar la relación de orden estricto `<` en **Bool** tal que `False < True`. Declararla como función y definirla en Haskell de modo que sea una función estricta en ambos argumentos. Demostrar con un ejemplo que la función `prec x y = or (neg x) y` no es equivalente a la función `(<)`.

2. Definir una función `sort2 ::` (**Int**, **Int**) $\to$ (**Int**, **Int**) que reciba como argumento un par de enteros de máquina y devuelva el mismo par de enteros, pero ordenado en sentido creciente.

   Definir una función `sort3 ::` (**Int**, **Int**, **Int**) $\to$ (**Int**, **Int**, **Int**) con la misma especificación, pero para ternas.

¿Cómo definiría una función `sort` que reciba como argumentos un entero n y una n-upla de enteros y la devuelva ordenada en sentido creciente?

3. Considere la declaración

   data `Triangle = Failure | Isosceles | Equilateral | Scalene`

   vista en el teórico. Derívela como instancia de la clase `Ord`. Elimine la sentencia de derivación del archivo y defina explícitamente la relación de orden que se obtuvo previamente por derivación ¿Cuántas ecuaciones ha utilizado? ¿Cuál es la mínima cantidad de ecuaciones necesarias para hacerlo?

4. Definir una función `nextlet :: Char→Char` que calculada sobre una letra del alfabeto, devuelve la siguiente. Asuma que la `a` sigue a la `z`.

5. Escribir en Haskell las distintas versiones del factorial vistas en el teórico. Compilar un programa que imprima una llamada a la función factorial con las opciones `-prof -fprof-auto -rtsopts` y correr el ejecutable con los parámetros `+RTS -p`. Comparar cuantos recursos utiliza cada versión.

6. Considere la función de dos variables m, n que calcula el número combinatorio $\binom{m}{n}$. Declarar explícitamente su tipo y escribirla en Haskell.

7. Escribir una función `fib` que calcule el n-ésimo término de la sucesión de Fibonacci F_n:

$$\begin{aligned} F_0 &:= 0 \\ F_1 &:= 1 \\ F_{n+1} &:= F_{n-2} + F_{n-1} \end{aligned}$$