Ejercicios de Programación Declarativa

Curso 2021/22

Hoja 2

1. Define funciones **recursivas** en Haskell para calcular las siguientes expresiones, en donde n es un número entero no negativo, que representa el argumento de la función.

   a) La lista de los cuadrados de los números naturales entre 0 y n (o sea, $[0, 1, 4, 9, \ldots, n^2]$).

   b) La lista anterior, pero con cada número emparejado con su cuadrado y en orden inverso ($[(n, n^2), \ldots, (2, 4), (1, 1), (0, 0)]$).

   c) La suma $\sum_{i=1}^{i=n} i \cdot \mid cos(i) \mid$.

   d) La suma de los números menores que n que sean múltiplos de 3 o 5.

   e) El número de potencias de 3 que sean menores que n y acaben en 43. Usa funciones auxiliares si te son de utilidad.

   Anota el tipo de las funciones que definas.

2. Programa en Haskell las siguientes funciones sin utilizar definiciones recursivas, sino llamadas a funciones de orden superior predefinidas:

   a) `zip3 :: [a] ->[b] ->[c] ->[(a,b,c)]`, análoga a `zip`, pero "empareja" tres listas en lugar de dos. El número de elementos de la lista resultante coincidirá con el de la lista más corta.

   b) `imparesEn xs` = lista de los números impares en la lista `xs`. Por ejemplo: `imparesEn [1..6]` = `[1,3,5]`

   c) `escalar xs ys` = producto escalar de las listas de igual longitud `xs` e `ys`. Por ejemplo: `escalar [1,3,5] [2,4,6]` $= 1 * 2 + 3 * 4 + 5 * 6$

   d) `mcdList xs` = máximo común divisor de los elementos de la lista (no vacía) `xs`.

3. Determina razonadamente cuál es el tipo (cualificado si es necesario) de las funciones definidas por las siguientes ecuaciones:

   a) `f1 x y = if x < y then x else y`

   b) `f2 x y = x (y + 1)`

   c) `f3 x y = (x y) + 1`

   d) `f4 x y z = x y (y z)`

4. Simplifica las siguientes expresiones siempre que estén bien tipadas:

   a) `(\x y -> y x) 2`

   b) `(\x y -> y x) 2 (\x -> x + 1)`

   c) `(\x -> \y -> x y) (\z -> z + 1) 2`

   d) `(\x -> \y -> y/x) 2`

   e) `(\x y -> y * x) 2 (\x -> x + 1)`

   f) `(\x y z -> y x (z x)) 2 (\x y -> y * x)`

   g) `(\x y z -> y x (z x)) 2 (\x y -> y * x) (\x -> x + 1)`

   h) `let y = (\x -> x + 1) in y 2`

   i) `(\x -> x + 1) (let y = \x -> x + 1 in y 2)`

5. Indica razonadamente cuál es el tipo (cualificado si es necesario) de las siguientes λ-expresiones:

   a) `\x -> \y -> y/x`

   b) `\x y z -> (y : x) ++ z`

   c) `\x -> \y -> \z -> z (y/x)`

   d) `\x y z -> if x == y then z x else z y`