Ejercicios de Programación Declarativa

Curso 2019/20

Hoja 2

1. 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)`

2. Supongamos una representación de los números racionales por medio de pares de enteros. Es decir, $\frac{n}{m}$ se representa como (n, m).

   a) Escribe una función `simplifica` que dada una fracción en forma de par, devuelva otra equivalente lo más simplificada posibles. Por ejemplo:
   `simplifica (15,9) = (5,3)`.

   b) Escribe una función para calcular el máximo común divisor de dos enteros positivos.

   c) Escribe una función para calcular el mínimo común múltiplo de dos enteros positivos.

   d) Utilizando las funciones anteriores define una función para cada una de las operaciones de abajo. Todas ellas deben devolver la expresión simplificada de la fracción resultado.

   - Suma de dos números racionales.
   - Resta de dos números racionales.
   - Multiplicación de dos números racionales.
   - División de dos números racionales.
   - Elevar un número racional a una potencia entera (incluye negativos).

   No olvides anotar las funciones con su tipo.

3. 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)`

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

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

   b) `\z -> y/x - z`

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

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

   a) Escribe una función `zip3 :: [a] ->[b] ->[c] ->[(a,b,c)]`, análoga a `zip`, pero que "empareje" 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 `xs`.

6. Utilizando listas intensionales escribe definiciones de las siguientes expresiones y funciones:

   a) `[(0,0),(1,2),(3,6),(7,14),(15,30),...]`

   b) `[1,-2,3,-4,5,-6,...]`

   c) `paresHasta n` = lista de los números naturales pares menores o iguales que `n`.

   d) `listpares n` = lista de los `n` primeros números naturales pares.

   e) `mezclaParImpar xs ys` = lista de todos los los pares posibles `(x,y)` tales que `x` es par y está en la lista `xs`, `y` es impar y está en la lista `ys`.

   f) `prefijos xs` = lista de las listas que son prefijo de `xs`. Por ejemplo:
   `prefijos [1,2,3] = [[],[1],[1,2],[1,2,3]]`.