Ejercicios de Programación Declarativa

Curso 2021/22

Soluciones Hoja 3

1. Utiliza listas intensionales para representar las siguientes listas:

   a) $[[1, 2, 3, 4, \ldots, 20], [1, 4, 9, 16, \ldots, 400], [1, 8, 27, \ldots, 8000], ..., [1, 2^{10}, 3^{10}, \ldots, 20^{10}]]$

   ```
   [[i ^ k | i <- [1..20]] | k <- [1..10]]
   ```

   b) $[[1, 1, 1, \ldots, 1], [2, 4, 8, 16, \ldots, 2^{10}], [3, 9, 27, \ldots, 3^{10}], \ldots, [20, 20^2, 20^3, \ldots, 20^{10}]]$

   ```
   [[i ^ k | k <- [1..10]] | i <- [1..20]]
   ```

2. Elimina, reemplazándolas por funciones auxiliares no locales, las definiciones locales y la λ-abstracción de la definición siguiente:

   ```
   f x y = map (\u -> (g u,g (u+1))) y
           where z = x * last y
                 g u = (x+z)*u

   f1 :: Num b => b -> [b] -> b -> (b, b)
   f1 x y = \u -> (g x y u,g x y (u+1))

   g :: Num a => a -> [a] -> a -> a
   g x y u = (x + z x y) * u

   z :: Num a => a -> [a] -> a
   z x y = x * last y

   newf :: Num b => b -> [b] -> [(b, b)]
   newf x y = map (f1 x y) y
   ```

   Más sencillo aun:

   ```
   f1 :: Num b => b -> [b] -> b -> (b, b)
   f1 x y = \u -> (g x y u,g x y (u+1))

   g :: Num a => a -> [a] -> a -> a
   g x y u = (x + x * last y) * u

   newf :: Num b => b -> [b] -> [(b, b)]
   newf x y = map (f1 x y) y
   ```

3. Elimina las listas intensionales de las siguientes definiciones, usando *map, filter* y *concat*:

```
f n     = [x*x | x <- [1..n], mod x 2 == 0]
f :: Integral a => a -> [a]
f n = map (^2) (filter (\x -> mod x 2 == 0) [1..n])
--f n = map (^2) (filter even [1..n])

g p n m   = [x+y | x <- [1..n], y <- [x..m], p y]
g :: (Num a, Enum a) => (a -> Bool) -> a -> a -> [a]
g p n m = concat (map f [1..n]) where f x = map (\y -> x + y) (filter p [x..m])

h p q n m = [x+y | x <- [1..n], p (n-x), y <- [x..m], q y]
h :: (Num a, Enum a) => (a -> Bool) -> (a -> Bool) -> a -> a -> [a]
h p q n m = concat (map f (filter (\z -> p (n-z)) [1..n]))
             where f x = map (\y -> x + y) (filter q [x..m])
```

4. Programa usando listas intensionales las siguientes expresiones:

   a) La lista con los números entre 19 y 50 emparejados cada uno con la lista de sus divisores (excluido el propio número), es decir, la lista:
   $[(19,[1]),(20,[1,2,4,5,10]),(21,[1,3,7]),\ldots,(50,[1,2,5,10,25])]$

   ```
   [(x,[y| y <- [1..(div x 2)], mod x y == 0])| x <- [19..50]]
   ```

   b) La lista de los números perfectos menores que 1000. Un número es perfecto si es igual a la suma de sus divisores (excluido él mismo). Por ejemplo, 6 es perfecto, pues 6=1+2+3.

   ```
   [x | x<-[1..(1000-1)], x == sum [y|y <- [1..(div x 2)], mod x y == 0]]
   ```

   c) Generaliza los dos apartados anteriores definiendo funciones, para que no dependan de números naturales concretos sino de los argumentos de la función que definas en cada caso.

   ```
   paresNumDiv :: Integral a => a ->a ->[(a, [a])]
   paresNumDiv n m = [(x,[y|y<- [1..(div x 2)], mod x y ==0])|x <-[n..m]]

   perfectos :: Integral a => a ->[a]
   perfectos n = [x | (x,d) <- paresNumDiv 1 (n-1), x == sum d]
   ```

5. Sea `minimoDesde p n` una función que devuelve el menor natural mayor o igual que `n` que satisface la propiedad `p`. Programa esta función usando listas intensionales. Utilízala para encontrar el primer primo a partir de 702.

   ```
   minimoDesde p n = head [x | x <- [n..], p x]

   minimoDesde primo 702
   ```

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

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

   ```
   [(2^i - 1, 2 * (2^i -1)) | i <- [0..]] :: Integral a => [(a, a)]
   ```

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

   ```
   [(- 1)^(k+1)*k | k <- [1..]] :: Integral a => [a]
   ```

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

   ```
   -- [x | x <- [0..n], even x]

   paresHasta :: Integral a => a -> [a]
   paresHasta n = [k * 2 | k <- [0..(div n 2)]]
   ```

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

   ```
   listpares :: Integral a => a -> [a]
   listpares n = [k * 2 | k <- [0..n-1]]
   ```

   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`.

   ```
   mezclaParImpar :: (Integral a, Integral b) => [a] -> [b] -> [(a, b)]
   mezclaParImpar xs ys = [(x,y) | x <- xs, even x, y <- xs, odd y]
   ```

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

   ```
   prefijos :: [a] -> [[a]]
   prefijos xs = [take n xs | n <- [0..length xs]]
   ```