Programación Funcional

Trabajo Práctico Nro. 10

Temas: Lazy evaluation. Estructuras infinitas. Elementos parciales. Principios de dualidad.

Bibliografía relacionada:

- Simon Thompson. The craft of Functional Programming. Addison Wesley, 1996. Cap. 13.
- Bird, Richard. Introduction to funtional programming using Haskell. Prentice Hall, 1998 (Second Edition). Cap. 4, 7 y 9.

1. *a*) Dar el tipo de las funciones que se definen a continuación:

   1) `head (x:xs) = x`
   2) `tail (x:xs) = xs`
   3) `pred n | n>0 = n-1`

   b) ¿Cuál es el valor de las siguientes expresiones, suponiendo evaluación lazy?

   1) `pred 0`
   2) `tail [pred 0]`
   3) `head (tail [pred 0])`
   4) `take 10 (filter p [1..])`

2. Dada la expresión:

```
let f n = n^2 : f (n+1)
    at 1 (x:xs)       = x
    at n (x:xs) | n>1 = at (n-1) xs
in at 3 (f 3)
```

   ¿Es posible, bajo algún orden de reducción, llegar a una forma normal? En caso afirmativo, indique cuál es el orden de reducción y obtenga la forma normal correspondiente. En caso negativo, explique por qué.

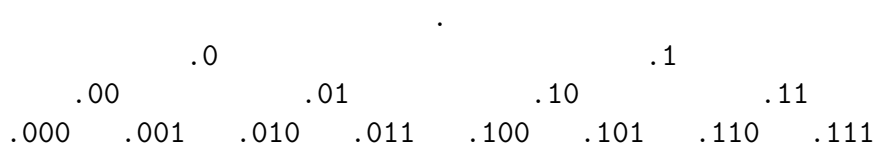
3. Sea `g :: Int -> Int -> Int` dada por `g x y = x * 10 + y`

   - Se quiere hacer una función `lstToInt` que convierta una lista de números, en el entero que representa. ¿Cuál de las dos funciones que están a continuación es correcta?
     `f1 = foldl g 0`
     `f2 = foldr g 0`
   - Determinar cuáles teoremas de dualidad `foldr-foldl` se verifican usando `g` como argumento. Justificar las respuestas.

- Determinar si las siguientes ecuaciones son verdaderas o falsas. Justificar.
  `foldr (-) x xs = x - sum xs`
  `foldl (-) x xs = x - sum xs`

4. ¿Por qué son importantes los principios de dualidad?

5. *a*) ⊛ Sea el tipo:
   `type FTree a b = a -> (b, a, a)`
   representando mediante funciones a los árboles cuyos nodos se identifican por elementos de tipo `a`, y que en cada uno tienen un dato de tipo `b`.
   Dada una función total de tipo `FTree a b` y un elemento de tipo `a`, puede generarse un árbol infinito evaluando sucesivamente la función sobre el elemento inicial.
   Por ejemplo, la función
   `paths = \s -> (s, s++"0", s++"1")`
   representaría, comenzando desde el valor `"."`, al árbol



   Implemente una función `levels :: FTree a b -> a -> [[b]]` que devuelva la lista infinita de niveles del árbol dado por la función y el elemento inicial recibidos.
   Por ejemplo,

```
levels (\x->(show x, 2*x, 2*x+1)) 1 =
    ["1"]:["2","3"]:["4","5","6","7"]:...
```

   b) Probar en Hugs

Ejercicios complementarios

6. Demuestre el ejercicio 11 de la práctica 5 (`rev = reverse`), utilizando los principios de dualidad.

7. Sea el tipo:
   `data Nat = Zero | Succ Nat`
   Considere que existe un elemento `inf :: Nat`, definido por `inf = Succ inf`, que puede verse como el 'infinito' sobre ese tipo.

Implemente una función `(<<) :: Nat -> Nat -> Bool` que devuelva `True` si el primero es menor que el segundo, teniendo en cuenta que está indefinido sólo en el caso donde ambos son `inf`.