

PROGRAMACIÓN FUNCIONAL

Trabajo Práctico Nro. 9

Temas: Patrones genéricos de recursión. Funciones sobre árboles.

Bibliografía relacionada:

- Bird, Richard. Introduction to functional programming using Haskell. Prentice Hall, 1998 (Second Edition). Cap. 3.
1. Dar el tipo e implementar la función `foldTip` para el tipo `TipTree` de la práctica 6. Reimplementar las funciones pedidas usando `foldTip`.
 2. Dado el tipo `data BinTree a = Empty | Bin a (BinTree a) (BinTree a)`
 - a) Definir las funciones `nodesBin` y `heightBin`, que calculan la cantidad de nodos y la altura del árbol binario respectivamente.
 - b) Definir las funciones `mapBin` y `mirrorBin`.
 - c) Definir la función `foldBin :: (a -> b -> b -> b) -> b -> BinTree a -> b`
 - d) Usando `foldBin`, definir las funciones `mapBin`, `heightBin` y `mirrorBin`.
 - e) Demostrar la equivalencia de las funciones definidas en los ítems b) y d)
 - f) Demostrar que `(mirrorBin . mirrorBin)` es equivalente a la identidad de árboles binarios.
 3. Considere la siguiente definición de tipo para representar árboles generales:
`data GenTree a = Gen a [GenTree a]`
 - a) Definir la función `foldGen :: (a -> [b] -> b) -> GenTree a -> b`
 - b) \star Definir otra función de fold para árboles generales, con el siguiente tipo:
`foldGen' :: (a -> c -> b) -> ([b] -> c) -> Gen a -> b`
 4. Usando `foldGen`, defina y dé el tipo de las funciones `mapGen`, `heightGen` y `mirrorGen`.
 5. Implementar los *folds* sobre cada uno de los siguientes tipos:
 - a) `data GenExp a = Leaf a | Un (GenExp a) | BinG (GenExp a) (GenExp a)`
 - b) `data NExp = Num Int | Sum NExp NExp | Sub NExp NExp | Neg NExp`
 - c) `Either a b = Left a | Right b`
 - d) `data Nat = Zero | Succ Nat`

6. Explique las ventajas de poder definir esquemas de funciones como funciones de alto orden.
7. Demostrar que para todo árbol binario t , de tipo `BinTree a`

$$\text{nodesBin } t \leq 2^{\text{heightBin } t} - 1$$

Ejercicios complementarios

8. Definir `mapT` (map para árboles) y `++T` (concatenación de árboles) para los tipos `TipTee` y `BinTree`. Demostrar:

- a) Ejercicio 6.a de la práctica 7.
- b) Ejercicio 9.a de la práctica 7.

9. Definiendo:

```
concatT :: [BinTree a] -> BinTree a
concatT = foldr1 (++T)
```

probar la igualdad del ejercicio 6.b de la práctica 7 y ver que no se cumple la propiedad 6.e de la misma.