75.31 Teoría de Lenguaje

75.24 Teoría de la Programación

Ejercicios adicionales Haskell

1. Dada una matriz representada mediante listas, devolver la matriz resultante de tomar las filas impares y columnas pares. El tipo de la función tiene que ser lo más general posible.

Ejemplo :

[[1, 5, 7, 9], [2, 4, 8, 4], [3, 6, 7, 8], [6, 8, 7, 3]]-> [[1, 5, 7, 9], [3, 6, 7, 8], [5, 4, 6, 8], [9, 4, 8, 3]]

2. Realizar una función posini que, dadas dos listas simples, devuelva la posición inicial en la cual la primera lista se encuentra contenida en la segunda. Si no se encuentra contenida devolver cero.

Ejemplo:

posini [5, 7] [1, 3, 5, 7, 9] -> 3

posini [12, 5] [1, 3, 5, 7, 9] -> 0

3. Definir una función que calcule la profundidad de una lista.

Ejemplo :

[[2, 3], [3, [[7]]], 4] -> 4

4. Usando programación de alto orden:

(a) definir la función paridad :: [String] -> [Int] que toma una lista de strings y devuelve una lista de enteros 0 / 1 tal que la i-ésima posición del resultado es 0 si la i-ésima cadena tiene longitud par y 1 si la i-ésima cadena tiene longitud impar.

Ejemplo :

paridad[“uno”, “do”, “tre”] -> [1, 0 ,1]

(b) definir una función sumacuad :: Int -> Int que dado un entero n calcula la suma de los cuadrados de 1 a n.

Ejemplo :

sumacuad 5 -> 1**2 + 2**2 + 3**2 + 4**2 + 5**2

(c) definir una función subsecuencias :: [a] -> [[a]] que calcule todas las subsecuencias (no necesariamente contiguas) de una lista dada.

Ejemplo :

subsecuencias [1, 2, 3] -> [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]

(d) definir una función frenteApar que recibe un entero y una lista de listas de enteros, y devuelve una lista que contiene sólo las sublistas cuyos elementos son todos pares, a las que se le agregan adelante el entero recibido por parámetro.

frenteApar 25 [[2, 3, 4], [1, 3, 2], [2, 6]. [1, 8], [8, 10]] -> [[25, 2, 6], [25, 8, 10]]

5. Definir el tipo algebraico Polinomio sobre enteros.

(a) Un polinomio es o bien una variable (representada por un valor de tipo String), una constante (de tipo Integer) o la suma o multiplicación de dos polinomios. Definir un tipo algebraico con cuatro constructores para declarar el tipo Polinomio. Por ejemplo, la siguiente es una expresión válida:

```
Add (Mul (Var "x") (Const 3)) (Mul (Var "y") (Var "y"))
```

(b) Escribir la función show para que el tipo Polinomio sea una instancia de Show (o sea

show :: Show a => a -> String) tal que:

una variable se muestre por su nombre, una constante por su valor y las sumas y productos por sus expresiones habituales (con los operandos entre paréntesis).

show Add (Mul (Var "x") (Const 3)) (Mul (Var "y") (Var "y")) -> “((x*3)+(y*y))”