Parcial

Programación Funcional, Universidad Nacional de Quilmes

2018 primer cuatrimestre

Aclaraciones:

- *Los ejercicios fueron pensados para ser resueltos en el orden presentado.*
- *No vale convertir la estructura en otra para resolver las funciones más fácilmente (ejemplo: convertir un árbol en una lista y resolver el ejercicio con funciones sobre listas).*
- *Puede utilizar funciones definidas en las prácticas, y propiedades ya demostradas, citándolas.*
- *Es sumamente aconsejable resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en clase, dado que este examen evalúa principalmente dichos aspectos.*
- *La complejidad de este examen es incremental, se recomienda resolverlo en el orden en que se presentan los ejercicios y problemas. No obstante, es aconsejable tratar hacer la mayor cantidad posible de ejercicios, por lo que también se recomienda no demorar mucho con un ejercicio puntual.*

Mapa

Representaremos un `Mapa` donde se van presentando objetos. Existen cofres en los finales de caminos, puntos que no poseen objetos, y bifurcaciones con objetos y dos caminos posibles. Además existen direcciones para recorrer el mapa: `Left` (ir hacia la izquierda), `Right` (ir hacia la derecha), `Straight` (seguir derecho). Un **camino** `[Dir]` en este mapa representa un camino desde la raíz del mapa hasta un cofre. Las definiciones de tipos son las siguientes:

```
data Dir = Left | Right | Straight
data Mapa a = Cofre [a] | Nada (Mapa a) | Bifurcacion [a] (Mapa a) (Mapa a)
```

1. Definir las siguientes funciones usando recursión explícita[1] sobre la estructura de `Mapa` (se pueden ver ejemplos en el anexo):

   a) `objects :: Mapa a -> [a]`
   Recolecta los objetos del mapa.

   b) `mapM :: (a -> b) -> Mapa a -> Mapa b`
   Dada una función, transforma los objetos del mapa

   c) `hasObjectAt :: (a -> Bool) -> Mapa a -> [Dir] -> Bool`
   Indica si un objeto al final de un camino cumple con cierta condición. Precondición: el camino existe en el mapa.

   d) `longestPath :: Mapa a -> [Dir]`
   Devuelve el camino más largo del mapa.

   e) `objectsOfLongestPath :: Mapa a -> [a]`
   Devuelve todos los objetos del camino más largo del mapa.

   f) `allPaths :: Mapa a -> [[Dir]]`
   Devuelve todos los caminos posibles en el mapa.

2. Dar tipo y definir `foldM` y `recM`, una versión de `fold` y recursión primitiva, respectivamente, para la estructura `Mapa`.

3. Definir las funciones del primer punto usando `foldM` o `recM`, según corresponda.

4. Demostrar las siguientes equivalencias usando las funciones definidas en el punto 1.

[1] Esto significa que, cuando se define una función usando recursión explícita, debe ser recursiva en sí misma, y no resolverse simplemente transformando el resultado de otra función.

a) `length . objects = countObjects`

Donde:

```
countObjects (Cofre xs) = length xs
countObjects (Nada m)  = countObjects m
countObjects (Bifurcacion xs m1 m2) = length xs + countObjects m1 + countObjects m2
```

b) `elem x . objects = hasObject (==x)`
Suponer ya demostrada la siguiente propiedad

- Para todo `x`, `as`, `bs`: `elem x (as ++ bs) = elem x as || elem x bs`

c) `length . map f . map g . objects = countObjects . mapM (f . g)`

Anexo con ejemplos

```
m1 = Bifurcacion []
       (Nada (Nada (Cofre [1])))
       (Bifurcacion []
           (Nada (Cofre []))
           (Bifurcacion [2]
              (Cofre [3])
              (Nada (Cofre [4, 5]))))

>> objects m1
[1,2,3,4,5]

>> mapM (const 0) m1
Bifurcacion []
   (Nada (Nada (Cofre [0])))
   (Bifurcacion []
       (Nada (Cofre []))
       (Bifurcacion [0]
          (Cofre [0])
          (Nada (Cofre [0, 0]))))

>> hasObjectAt (==1) m1 [Left, Straight, Straight]
True


>> longestPath m1
[Right,Right,Right,Straight]

>> objectsOfLongestPath m1
[2,4,5]

>> allPaths m1 [
  [Left,Straight,Straight],
  [Right,Left,Straight],
  [Right,Right,Left],
  [Right,Right,Right,Straight]
]
```