Estructuras Discretas
Proyecto 1

Odín Miguel Escorza Soria Daniela Calderón Pérez

Facultad de Ciencias UNAM
Fecha de entrega: 30 Septiembre 2017

Realiza los siguientes ejercicios

Si requieren de funciones auxiliares, las tienen que hacer ustedes, si creen necesitar alguna función de Haskell que **no sean + - * / o div**, consultenme antes de usarlas.

1. Naturales

El siguiente tipo es una manera de representar naturales en Haskell

data Natural = Cero | Suce Natural deriving Show

Considerando este tipo, define las siguientes funciones.

1. Define la función **sumaN1**, la cual realiza la suma de dos naturales.
   Firma de la función:
   sumaN1 :: Natural → Natural → Natural
   Ejemplos:
   *Main> sumaN1 **(**Suce (Suce (Suce Cero))**)** **(**Suce (Suce (Suce (Suce Cero)))**)**
   Suce (Suce (Suce (Suce (Suce (Suce (Suce Cero))))))

2. Define la función **multN1** que multiplique dos Naturales
   Firma de la función:
   multN1 :: Natural → Natural → Natural
   Ejemplo:
   *Main> multN1 (Suce (Suce Cero)) (Suce (Suce (Suce Cero)))
   Suce (Suce (Suce (Suce (Suce (Suce Cero)))))

Para las funciones 1 y 2 NO pueden usar ninguna función del preludio de Haskell

3. Define una función **natToInt** que convierta un Natural a un Int de Haskell
   Firma de la función:
   natToInt :: Natural → Int
   Ejemplos
   *Main> natToInt $ Suce (Suce (Suce (Suce (Suce (Suce Cero)))))
   6

4. Define una función **intToNat** que convierta un Int a un Natural
   Firma de la función:
   intToNat :: Int → Natural
   Ejemplos
   *Main> intToNat 5
   Suce (Suce (Suce (Suce (Suce Cero))))

5. Define la función **restaSeg** la cual resta dos Naturales, cuidando que al primero se le pueda restar el segundo.
   Firma de la función:
   restaSeg :: Natural → Natural → Natural
   Ejemplo:
   *Main> restaSeg (Suce (Suce (Suce (Suce (Suce (Suce Cero)))))) (Suce (Suce (Suce (Suce (Suce Cero)))))
   Suce Cero
   *Main> restaSeg (Suce (Suce (Suce Cero))) (Suce (Suce (Suce (Suce Cero))))
   ** Exception: No definido

2. Árboles

El siguiente tipo es una manera de representar árboles en Haskell

data Arbol = Hoja Int | Nodo Arbol Int Arbol deriving Show

Considerando este tipo, define las siguientes funciones.

1. Define la función **aparece** que nos dice si un elemento está en un árbol.
   aparece :: Int → Arbol → Bool
   Ejemplo:

*Main> aparece 3 (Nodo (Nodo (Hoja (-1)) 0 (Hoja 1)) 2 (Nodo (Hoja 3) 4 (Hoja 5)))
True

2. Define la función **aplanaPO** la cual convierte un árbol en una lista haciendo el recorrido en **preorden**.
   Firma de la función:
   aplanaPO :: Arbol → [Int]
   Ejemplo:
   *Main> aplanaPO (Nodo (Nodo (Hoja (-1)) 0 (Hoja 1)) 2 (Nodo (Hoja 3) 4 (Hoja 5)))
   $[2, 0, -1, 1, 4, 3, 5]$

3. Define la función **aplanaIO** la cual convierte un árbol en una lista haciendo el recorrido en **inorden**.
   Firma de la función:
   aplanaIO :: Arbol → [Int]
   Ejemplo:
   *Main> aplanaIO (Nodo (Nodo (Hoja (-1)) 0 (Hoja 1)) 2 (Nodo (Hoja 3) 4 (Hoja 5)))
   $[-1, 0, 1, 2, 3, 4, 5]$

4. Define la función **aplanaPsO** la cual convierte un árbol en una lista haciendo el recorrido en **postorden**.
   Firma de la función:
   aplanaPsO :: Arbol → [Int]
   Ejemplo:
   *Main> aplanaPsO (Nodo (Nodo (Hoja (-1)) 0 (Hoja 1)) 2 (Nodo (Hoja 3) 4 (Hoja 5)))
   $[-1, 1, 0, 3, 5, 4, 2]$

3. Listas

El siguiente tipo es una manera de representar nuestras listas en Haskell

data Lista = Vacia | Conc Int Lista deriving Show

Considerando este tipo, define las siguientes funciones.

1. Define la función longitud sobre Lista
   Firma de la función:

long :: Lista → Int
Ejemplo:
*Main> long $ Conc 1 (Conc 2 (Conc 3 (Conc 4 Vacia)))
4

2. Define la función **takeL** sobre Lista
   Firma de la función:
   takeL :: Int → Lista → Lista
   Ejemplo:
   *Main> takeL 2 $ Conc 1 (Conc 2 (Conc 3 (Conc 4 Vacia)))
   Conc 2 (Conc 1 Vacia)

3. Define la función **dropL** sobre Lista
   Firma de la función:
   dropL :: Int → Lista → Lista
   Ejemplo:
   *Main> dropL 2 $ Conc 1 (Conc 2 (Conc 3 (Conc 4 Vacia)))
   Conc 3 (Conc 4 Vacia)

4. Define la funcion **concatena** sobre Lista
   Firma de la función:
   concatena :: Lista → Lista → Lista
   Ejemplo:
   *Main> concatena (Conc 1 (Conc 2 (Conc 3 Vacia))) (Conc 4 (Conc 5 Vacia))
   Conc 1 (Conc 2 (Conc 3 (Conc 4 (Conc 5 Vacia))))

5. Define la funcion **reversa** sobre Lista
   Firma de la función:
   reversa :: Lista → Lista
   Ejemplo:
   *Main> reversa $ Conc 1 (Conc 2 (Conc 3 (Conc 4 (Conc 5 Vacia))))
   Conc 5 (Conc 4 (Conc 3 (Conc 2 (Conc 1 Vacia))))

4. Relaciones

Los siguientes tipos son una forma de representar relaciones binarias en Haskell.

type ProductoBI = [(Int,Int)]

type RelacionBI = ProductoBI

Considerando estos tipos, realiza las siguientes funciones.

1. Realiza una función que regrese la unión de relaciones binarias
   Firma de la función:
   union :: RelacionBI → RelacionBI → RelacionBI
   Ejemplo:
   *Main> union [(1,2),(3,4),(5,6)] [(5,6),(7,8),(8,9)]
   $[(1,2),(3,4),(5,6),(7,8),(8,9)]$

2. Realiza una función que regrese la intersección de relaciones binarias
   Firma de la función:
   interseccion :: RelacionBI → RelacionBI → RelacionBI
   Ejemplo:
   *Main> interseccion [(1,2),(3,4),(5,6)] [(5,6),(7,8),(8,9)]
   $[(5,6)]$

3. Realiza una función que regrese la diferencia de relaciones binarias
   Firma de la función:
   diferencia :: RelacionBI → RelacionBI → RelacionBI
   Ejemplo:
   *Main>diferencia [(1,2),(3,4),(5,6)] [(5,6),(7,8),(8,9)]
   $[(1,2),(3,4)]$

4. Realiza una función que regresa la inversa de una relación
   Firma de la función:
   inversa :: RelacionBI → RelacionBI
   Ejemplo:
   *Main>inversa [(1,2),(3,4),(5,6)]
   $[(2,1),(4,3),(6,5)]$

5. Realiza una función que regresa la composición de una relación
   Firma de la función:
   composición :: RelacionBI → RelacionBI → RelacionBI
   Ejemplo:
   *Main> composicion [(1,8),(3,7),(5,6)] [(5,6),(7,8),(8,9)]
   $[(1,9),(3,8)]$

6. Realiza una función que regrese True o False, en caso de que se cumplan las siguientes propiedades.

a) Reflexividad
Firma de la función:
reflexividad :: RelacionBI → Bool
Ejemplos:
*Main> reflexividad [(1,2),(2,2),(3,4),(1,1),(4,4)]
False
*Main> reflexividad [(3,3),(1,2),(2,2),(3,4),(1,1),(4,4)]
True

b) Antirreflexividad
Firma de la función:
antirreflexiva :: RelacionBI → Bool
Ejemplos:
*Main> antirreflexividad [(1,2),(2,2),(3,4),(1,1),(4,4)]
False
*Main> antirreflexividad [(1,2),(3,4),(5,6)]
True

c) Simetría
Firma de la función:
simetria :: RelacionBI → Bool
Ejemplos:
*Main> simetria [(1,2),(3,4),(5,6)]
False
*Main> simetria [(6,5),(1,2),(2,1),(3,4),(5,6),(4,3)]
True

d) Antisimetría
Firma de la función:
antisimetria :: RelacionBI → Bool
Ejemplos:
*Main>antisimetria [(6,5),(1,2),(2,1),(3,4),(5,6),(4,3)]
False
*Main> antisimetria [(1,1),(2,2)]
True

e) Asimetría
Firma de la función:
asimetria :: RelacionBI → Bool
Ejemplos:
*Main> asimetria [(1,1),(2,2)]
False
*Main> asimetria [(1,2),(3,4),(5,6)]

True

f) Transitividad
Firma de la función:
transitiviad :: RelacionBI → Bool
Ejemplo:
*Main>transitividad [(2,2),(1,2),(2,1),(1,1)]
True
*Main> transitividad [(1,2),(3,4),(5,6)]
False

Observaciones

- Les recomiendo empezar su proyecto lo más pronto posible.
- Los proyectos son **individuales**.
- Si requieren de cualquier función auxiliar para realizar su práctica, deben implementarlas ustedes.
- El asunto de la práctica es [**ED2018-1 Proyecto1**]
- Se enviará un correo automatico si el proyecto se envió con el asunto correcto
- Consulten los lineamientos de entrega antes de enviar
- Cualquier duda pueden mandarme correo

¡Suerte!