Ejercicio Semanal 1

Emiliano Galeana Araujo

Facultad de Ciencias, UNAM

Fecha de entrega: 17 de Agosto de 2018

1. Descripción del programa

En este semanal vemos dos tipos de datos, las listas Snoc, y números naturales con su representación en binario.

1.1. Listas Snoc

```
data ListS a = NilS | Snoc (ListS a) a deriving Show
```

Un ejemplo de una lista `Snoc` es la lista [1,2,3,4,5] se ve la siguiente manera.

```
Snoc(Snoc(Snoc(Snoc(Snoc NilS 1)2)3)4)5
```

Las siguientes son funciones recursivas para el tipo de dato ListS(`Listas Snoc`).

```
-- | headS. Funcion que obtiene el primer elemento de la lista.
headS :: ListS a → a

-- | tailS. Funcion que obtiene la lista sin el primer elemento.
tailS :: ListS a → ListS a

-- | initS. Funcion que obtiene la lista sin el primer elemento.
initS :: ListS a → ListS a

-- | lastS. Funcion que obtiene el ultimo elemento de la lista.
lastS :: ListS a → a
```

```
-- | nthElementS. Funcion que regresa el n-esimo elemento de la lista.
nthElementS :: Int → ListS a → a

-- | deleteNthElementS. Funcion que elimina el n-esimo elemento de la lista.
deleteNthElementS :: Int → ListS a → ListS a

-- | addFirstS. Funcion que obtiene la lista donde el primer elemento es el
-- | elemento dado.
addFirstS :: a → ListS a → ListS a

-- | addLastS. Funcion que obtiene la lista donde el ultimo elemento es el
-- | elemento dado.
addLastS :: a → ListS a → ListS a

-- | reverseS. Funcion que obtiene la reversa de la lista.
reverseS :: ListS a → ListS a

-- | appendS. Funcion que obtiene la concatenaci n de dos listas.
appendS :: ListS a → ListS a → ListS a

-- | takeS. Funcion que obtiene la lista con los primeros n elementos.
takeS :: Int → ListS a → ListS a
```

1.2. Números naturales

```
data Nat = Zero | D (Nat) | O (Nat) deriving Show
```

Recordemos que Zero es la represantacion del cero (0), D x representa al doble de x, con x un número natural (2x), y O x representa al sucesor del doble de x, con x un número natural (2x + 1). Un ejemplo de la represantacion de un número natural en binario es 10011 que lo vemos de esta forma:

```
(O ( O ( D ( D ( O Zero)))))
```

Las siguientes son funciones recursivas para el tipo de dato `Nat`(`Numeros naturales`).

```
-- | toNat. Funcion que obtiene la representacion en numeros Nat de un numero
-- | entero.
toNat :: Int → Nat
```

```
-- | succ. Funcion que obtiene el sucesor de un numero Nat.
succN :: Nat → Nat

-- | pred. Funcion que obtiene el predecesor de un numero Nat.
predN :: Nat → Nat

-- | add. Funcion que obtiene la suma de dos numeros Nat.
addN :: Nat → Nat → Nat

-- | prod. Funcion que obtiene el producto de dos numeros Nat.
prod :: Nat → Nat → Nat
```

2. Entrada y ejecución

El programa es interpretado por GCHI de la siguiente forma

```
~:ghci EjerSem01.hs
```

2.1. Listas Snoc

Ya en el programa, los siguientes son ejemplos de ejecución de las listas Snoc.

```
*EjerSem01> headS (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
1

*EjerSem01> tailS (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
Snoc(Snoc(Snoc(Snoc NilS 2)3)4)5

*EjerSem01> initS (Snoc NilS 1)
NilS

*EjerSem01> lastS (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
5

*EjerSem01> nthElementS 2 (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
3

*EjerSem01> deleteNthElementS 5 NilS
NilS
```

```
*EjerSem01> addFirstS 0 (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
Snoc(Snoc(Snoc(Snoc(Snoc(Snoc NilS 0)1)2)3)4)5

*EjerSem01> addLastS 6 (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
Snoc(Snoc(Snoc(Snoc(Snoc(Snoc NilS 1)2)3)4)5)6

*EjerSem01> reverseS (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
Snoc(Snoc(Snoc(Snoc(Snoc NilS 5)4)3)2)1

*EjerSem01> appendS (Snoc(Snoc(Snoc NilS 1)2)3) (Snoc(Snoc(Snoc NilS 6)7)8)
Snoc(Snoc(Snoc(Snoc(Snoc(Snoc NilS 1)2)3)6)7)8

*EjerSem01> takeS 0 (Snoc (Snoc (Snoc (Snoc (Snoc NilS 1) 2) 3) 4) 5)
NilS
```

2.2. Nat

Ya en el programa, los siguientes son ejemplos de ejecución de `Nat`.

```
*EjerSem01> toNat 616
D (D (D (O (D (O (O (D (D (O Zero)))))))))

*EjerSem01> succN (D(O(D Zero)))
O (O Zero)

*EjerSem01> predN (D(O Zero))
O Zero

*EjerSem01> addN (D(O(D Zero))) (D(O Zero))
D (D (O Zero))

*EjerSem01> prod (D(O(D Zero))) (D(O Zero))
D (D (O Zero))
```

3. Conclusiones

Hubo varios problemas relacionados a ambos tipos de datos, pero en general fue una ejercicio sencillo, los tipos de problemas eran cuestión de entender

bien qué es lo que tenía que hacer no fue muy complicado de entender.

3.1. Listas

El mayor problema con las listas `Snoc` fue en las funciones de `nthElementS`, y `deleteNthElementS` porque en un inicio pensé que se tenía que recorrer desde digamos del elemento junto a `NilS`, fue complicado pensar como ir toda la lista hacia dentro y luego ir avanzando, luego pensé que se tenía que recorrer del último elemento, y así estaba sencillo de implementar, pero al final siempre si era desde el elemento junto a `NilS`, lo gracioso fue que la segunda vez ya no fue complicado de implementar, y eran problemas similares, una vez que logré recorrer la lista, fue fácil regresar el elemento o borrarlo, `takeS` fue similar, pero al momento de implementarlo ya había resuelto el problema de recorrer la lista.

3.2. Nat

Implemetar este tipo de dato fue relativamente sencillo, pues ya había hecho algo similar, el mayor problema fue que hay muchas maneras de escribir el mismo número, pero todas se pueden simplificar, se puede escribir `(...(D Zero))` que es igual a quitar la `D` junto al cero, pues el doble de `Zero` es el mismo, y al momento de hacer pruebas, a veces las hacía sin esa `D`, y todo salía según lo esperado, pero cuando se la ponía en la función `prod` no funcionaba, al final terminé casi por arreglarlo, pues solo puede borrar una de las n posibles `Ds`. Otro problema fue con la suma, pero no entiendo bien qué pasó, a veces jalaba y a veces no, fue interesante poder simplificarla, pues en un inicio pensé en hacer todas las combinaciones posibles para su implementación.

Referencias

[1] Archivero, curso de Estructuras Discretas 2017-1