Universidad Nacional Autónoma de México
Facultad de Ciencias

Estructuras Discretas

Práctica 3

Javier Enríquez Mendoza **Mauricio E. Hernández Olvera**

21 de Septiembre de 2018
Fecha de entrega: 5 de Octubre de 2018

Objetivos

- Familiarizar al alumno con el lenguaje de programación Haskell.
- Definir tipos de datos en Haskell.
- Reforzar los conocimientos adquiridos en el laboratorio.

Instrucciones generales

La práctica debe resolverse en los archivos `Binario.hs` y `Practica3.hs` y las firmas de las funciones deben ser idénticas a las que se muestran en cada ejercicio. Cada función y definición debe estar debidamente comentada con la especificación de ésta.

Se tomará en cuenta la legibilidad y el estilo del código.

Ejercicios

1 Tipos de Datos

Esta sección se resuelve en el archivo `Binario.hs` .

Ejercicio 1.1.1 (1 pt.) Definir el tipo de dato `Binario` para representar los números mayores o iguales a 1 en binario con los siguientes constructores:

`BaseUno`	1
`Uno`	Agrega un 1 como digito menos significativo
`Cero`	Agrega un 0 como digito menos significativo

Por ejemplo:

Int	Binario	show
1	BaseUno	1
2	Cero BaseUno	10
3	Uno BaseUno	11
4	Cero (Cero BaseUno)	100
5	Uno (Cero BaseUno)	101

Ejercicio 1.1.2 (1 pt.) Definir la función `natToBin` que recibe un entero y regresa su representación binaria.

```
natToBin :: Int -> Binario
```

```
> natToBin 15
1111
> natToBin 567
1000110111
```

Ejercicio 1.1.3 (1 pt.) Definir la función `binToNat` que recibe un Binario y regresa su representación entera

```
binToNat :: Binario -> Int
```

```
> binToNat (Uno (Uno (Uno BaseUno)))
15
> binToNat (Uno (Uno (Uno (Cero (Cero BaseUno)))))
39
```

Ejercicio 1.1.4 (1 pt.) Definir la función `sucesor` que recibe un Binario y regresa el sucesor de éste. **No** pueden usarse las funciones `natToBin` **ni** `binToNat`

```
sucesor :: Binario -> Binario
```

```
> sucesor (Cero (Uno (Uno (Cero BaseUno))))
10111
> sucesor (Uno (Cero (Cero BaseUno)))
1010
```

Ejercicio 1.1.5 (1 pt.) Definir la función `bitsEncendidos` que recibe un Binario y el número de bits encendidos (la cantidad de unos).

```
bitsEncendidos :: Binario -> Int
```

```
> bitsEncendidos (Uno (Cero (Uno (Uno (Cero BaseUno)))))
4
> bitsEncendidos (Uno (Cero (Uno BaseUno)))
3
```

2 Funciones de Orden Superior

Para esta sección es indispensable utilizar las funciones `map` o `filter` en cada ejercicio. Y se resuelve en el archivo `Practica3.hs` .

Ejercicio 2.1 (1 pt.) Definir la finción `binarios` que recibe una lista de enteros y regresa una lista de Binarios con la representación en binario de cada uno de ellos.

```
binarios :: [Int] -> [Binario]
```

```
> binarios (take 10 [1..])
[1,10,11,100,101,110,111,1000,1001,1010]
> binarios [20..25]
[10100,10101,10110,10111,11000,11001]
```

Ejercicio 2.2 (1 pt.) Definir la función `pares` que recibe una lista de números en su representación binaria y regrean una lista unicamente con aquellos que sean pares.

```
pares :: [Binario] -> [Binario]
```

```
> pares (binarios [20..25])
[10100,10110,11000]
> pares (binarios [2,3,5,7,11])
[10]
```

Ejercicio 2.3 (1 pt.) Definir la función `tooLong` que recibe una lista de cadenas y regresa la lista con las cadenas de longitud menor o igual a 7.

```
tooLong :: [String] -> [String]
```

```
> tooLong ["Hola", "HolaH", "HolaHo", "HolaHol", ‘‘HolaHola’’]
["Hola","HolaH", "HolaHo", "HolaHol"]
> tooLong (take 100 (repeat "TOOLONG!"))
[]
```

Ejercicio 2.4 (1 pt.) Definir la función sFibonacci que recibe un entero n y regresa una lista con los primeros n + 1 elementos de la sucesión de fibonacci.

Hint: Importar el módulo de la Práctia 2 para poder usar la función `fibonacci`.

```
sFibonacci :: Int -> [Int]
```

```
> sFibonacci 12
[0,1,1,2,3,5,8,13,21,34,55,89,144]
> sFibonacci 15
[0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610]
```

Ejercicio 2.5 (1 pt.) Definir la función `quitaElemento` que recibe una lista de elementos comparables y un elemento de la lista y regresa una nueva lista sin nunguna aparición de este elemento.

```
quitaElemento :: (Eq a) => [a] -> a -> [a]
```

```
> quitaElemento [1,2,3,4,5,1,2,3,4,5,1,1,1] 1
[2,3,4,5,2,3,4,5]
> quitaElemento [’a’,’a’,’a’,’a’,’a’,’a’] ’a’
[]
```

3 Extras

Ejercicio 3.1 (0.5 pts. extra) Define un nuevo tipo de dato `Binario2` que incluya al 0 sin generar ambigüedad en las representaciones del resto de los números.

Entrega

- La entrega se realiza mediante correo electrónico a la dirección del ayudante de laboratorio (javierem_94@ciencias.unam.mx).
- La practica deberá ser entregada en equipos de máximo 3 personas.
- Se debe entregar un directorio `numeroCuenta_P03`, dónde `numeroCuenta` es el número de cuenta de un integrante del equipo. Dentro del directorio se debe incluir:
  - Un archivo readme.txt con los nombres y números de cuenta de los alumnos, comentarios, opiniones, críticas o ideas sobre la práctica.
  - Los archivos requeridos en la práctica. Debe enviarse código lo más limpio posible.
- Los archivos requeridos para esta práctica son: `Practica3.hs y Binario.hs`.
- El directorio se deberá comprimir en un archivo con nombre `numeroCuenta_P03.tar.gz`,dónde `numeroCuenta` es el número de cuenta de un integrante del equipo.
- Únicamente **un integrante** del equipo deberá enviar el correo con la práctica.
- El asunto del correo debe ser [ED-20191-P03].
- Se recibirá la práctica hasta las 23:59:59 horas del día fijado como fecha de entrega, cualquier práctica recibida después no será tomada en cuenta.
- **Cualquier práctica total o parcialmente plagiada, será calificada automáticamente con cero y no se aceptarán más prácticas durante el semestre.**