Programación Declarativa 2020-2
Facultad de Ciencias UNAM
Tarea 1: Haaave you met Haskell?

Favio E. Miranda Perea Javier Enríquez Mendoza

Fecha de entrega: 7 febrero de 2020

Strings

En Haskell el tipo de dato `String` se define como una lista de `Char`, así que se pueden manipular con todas las funciones definidas para listas. Incluso pueden manipularse a través de listas por comprensión. Por ejemplo, usamos listas por comprensión para combinar cada adjetivo con un posible sustantivo.

```
[noun ++ " " ++ adj | noun <- ["perro", "lenguaje"]
                    , adj <- ["perezoso", "sucio"]]
= ["perro perezoso","lenguaje perezoso","perro sucio","lenguaje sucio"]
```

Realiza los siguientes ejercicios utilizando listas por comprensión:

1. Define la función `quitaMayusculas` que elimina todas las mayúsculas de una cadena.

   ```
   quitaMayusculas "I <3 Haskell" = "<3 askell"
   ```

2. Define la función `soloLetras` que elimina todos los caracteres que no sean letras de una cadena, las letras son unicamente a..z ó A...Z.

   ```
   soloLetras "Oppan Lambda Style!!" = "OppanLambdaStyle"
   ```
   [1]

3. Define la función `prefijo xs ys` que regresa un `Bool`, `True` si `xs` es prefijo de `ys` y `False` en otro caso.

   ```
   prefijo "Haskell" "Que Chido es Haskell" = False
   prefijo "Que Chido" "Que Chido es Haskell" = True
   ```

[1] https://youtu.be/Ci48kqp11F8

Merge Sort

Merge Sort es un clásico ejemplo de algoritmos que siguen la bien conocida filosofía *Divide y Vencerás*. Divide la lista de entrada en dos partes iguales y recursivamente las ordena, una vez ordenadas mezcla ambas listas. Para esta sección implementaremos nuestra propia versión de Merge Sort para listas de `Int`, en Haskell.

1. Define las funciones necesarias para que la función `mergeSort` definida a continuación funcione correctamente.

```
mergeSort :: (Ord a) => [a] -> [a]
mergeSort [] = []
mergeSort [x] = [x]
mergeSort xs = mezcla (mergeSort f) (mergeSort s)
 where (f,s) = parte xs
```

   Hay que notar que **al menos** se deben definir las funciones `parte` y `mezcla` respetando las firmas de tipos:

```
parte :: [a] -> ([a], [a])

mezcla :: (Ord a) => [a] -> [a] -> [a]
```

2. En Haskell el tipo de dato `Ordering` se define como sigue:

```
data Ordering = LT | EQ | GT
```

   Que representan menor, igual y mayor respectivamente. Además esta implementado de tal forma que se cumplen las siguientes propiedades:

```
compare x y == LT <=> x < y
compare x y == EQ <=> x == y
compare x y == GT <=> x > y
```

   Define las funciones `mezclaCon` y `mergeSortCon` que toman una comparación explicita como argumento y la utilizan para ordenar los elementos. De tal forma que se cumpla:

```
mezclaCon compare xs ys == mezcla xs ys
mergeSortCon compare xs == mergeSort xs
```

Coloración

Definamos el tipo de dato `Color` con los siguientes constructores:

```
data Color =  Rojo | Amarillo | Verde | Azul  deriving (Eq, Show)
```

Definimos también el tipo de dato `Balcanes` que define a los países que pertenecen a la Península Balcánica

```
data Balcanes = Albania
              | Bulgaria
              | BosniayHerzegovina
              | Kosovo
              | Macedonia
              | Montenegro
             deriving (Eq, Show)
```

Dos países son adyacentes cuando comparten frontera. Definimos esa relación sobre los países pertenecientes a la Península Balcánica con la siguiente lista:

```
type Ady = [(Balcanes, Balcanes)]

adyacencias :: Ady
adyacencias =
   [ (Albania, Montenegro), (Albania, Kosovo), (Albania, Macedonia)
    , (Bulgaria, Macedonia), (BosniayHerzegovina, Montenegro)
    , (Kosovo, Macedonia), (Kosovo, Montenegro)
    ]
```

`x` es adyacente a `y` si `(x,y) elem adyacencias` o `(y,x) elem adyacencias`
Se define una coloración con el sinónimo:

```
type Coloracion = [(Color, Balcanes)]
```

Que relaciona un país con un color. Una buena coloración respecto a una matriz de adyacencia es una coloración en la que si dos países son adyacentes entonces tienen colores diferentes. Una coloración es completa respecto a una matriz de adyacencia si todos los países de la matriz están coloreados, las coloraciones buenas podrían ser incompletas.

1. Define la función `esBuena` con la firma

   ```
   esBuena :: Ady -> Coloracion -> Bool
   ```

   Que regresa `True` si la coloración recibida es buena respecto a la matriz de adyacencias y `False` en otro caso.

2. Define la función `coloraciones` con la firma

   ```
   coloraciones :: Ady -> [Coloracion]
   ```

   Que calcula todas las coloraciones buenas y completas respecto a la matriz de adyacencias recibida.