Lógica Computacional
Práctica 02

Profesor: Favio Ezequiel Miranda Perea
Ayudante: Javier Enríquez Mendoza
Ayudante: Fernando Abigail Galicia Mendoza
Laboratorio: Emiliano Galeana Araujo
Laboratorio: América Montserrat García Coronado

Facultad de ciencias, UNAM

Fecha de entrega: 2021

1 Fórmulas Proposicionales

El archivo `Operadores.hs` es una clase en la que definimos los operadores que vamos a utilizar con las fórmulas proposicionales, así que ese archivo no debe ser modificado. Pero es importante para la práctica, así que déjalo en la misma carpeta que `FormProp.hs`.

El archivo *importante* es `FormProp.hs` donde definimos lo siguiente:

1. Las variables proposicionales se pueden implementar como cadenas, esto es un alias para el tipo `String`.

```
type VarP = String
```

2. Un estado se representa con una lista de variables proposicionales que están definidas como verdaderas.

```
type Estado = [VarP]
```

3. Nuestro tipo de dato para la lógica proposicional, puedes ver que es una definición recursiva y el caso base es `Var` que es para las fórmulas atómicas.

```
data Prop = Var VarP
          | Neg Prop
          | Conj Prop Prop
          | Disy Prop Prop
          | Impl Prop Prop
          | Syss Prop Prop
```

1.1 Ejercicios

Deberás completar las funciones que se listan en el archivo `FormProp.hs`, no puedes utilizar bibliotecas que resuelvan el problema, pero te puedes ayudar de `Data.List`. Encontrarás ejemplos que te pueden ayudar a verificar casos para tus funciones, pero te invitamos a crear tus propios ejemplos.

1. Instancias

   (a) Deberás crear la instancia de `Show` para `Prop`. Es muy parecida a la instancia de `Show` para listas (Práctica 1). Recuerda que para `Var VarP` vamos a quitar el constructor `Var`.

   (b) Deberás crear la instancia de `Operadores` para `Prop`. Solo necesitas mapear cada operador con su respectivo construtor.

   Esto es para poder escribir las fórmulas de la siguiente manera:

   ```
   i  = Conj (Var "p") (Var "q")
   i' = (Var "p") /\ (Var "q")
   ```

   Las cuales representan la misma fórmula. Lo único distinto es con la negación que se escribe entre paréntesis (¬):

2. Funciones

   (a) Función que recibe una fórmula y devuelve el conjunto (no hay repeticiones) de variables que hay en una fórmula.

   ```
   vars :: Prop -> [VarP]
   ```

   (b) Función que evalúa una proposición dado un estado.

```
interp :: Estado -> Prop -> Bool
```

(c) Función que cuenta el número de conectivos.

```
numConectivos :: Prop -> Int
```

(d) Función que elimina las equivalencias (<->).

```
elimEquiv :: Prop -> Prop
```

(e) Función que elimina las implicaciones, puedes suponer que no hay equivalencias.

```
elimImpl :: Prop -> Prop
```

(f) Función que dada una fórmula ϕ con n-variables devuelve la lista con 2^n estados distintos para ϕ.

```
estados :: Prop -> [Estado]
```

(g) Función que nos da TODOS los modelos de una proposición.

```
modelos :: Prop -> [Estado]
```

(h) Función que nos dice si una proposición es una tautologia.

```
tautologia :: Prop -> Bool
```

(i) Función que nos dice si una proposición es satisfacible en una interpretación.

```
satisfen :: Estado -> Prop -> Bool
```

(j) Función que nos dice si una proposición es satifacible.

```
satisfacible :: Prop -> Bool
```

(k) Función que nos dice si una proposición es insatisfacible en una interpretación.

```
insatisfen :: Estado -> Prop -> Bool
```

(l) Función que nos dice si una proposición es instisfacible.

```
contrad :: Prop -> Bool
```

(m) Función que regresa una fórmula equivalente donde las negaciones solo se aplican a fórmulas atómicas.

```
meteNegacion :: Prop -> Prop
```

(n) Función que regresa una fórmula equivalente donde las disyunciones sólo se aplica a disyunciones o literales. Puedes suponer que la fórmula que recibes está en FNN.

```
interiorizaDisyuncion :: Prop -> Prop
```

(o) Función que regresa una fórmula equivalente donde las conjunciones sólo se aplica a conjunciones o literales. Puedes suponer que la fórmula que recibes está en FNN.

```
interiorizaConjuncion :: Prop -> Prop
```

Probablemente necesites de las siguientes funciones auxiliares:

(a) Función que calcula el conjunto potencia.

```
subconj :: [a] -> [[a]]
```

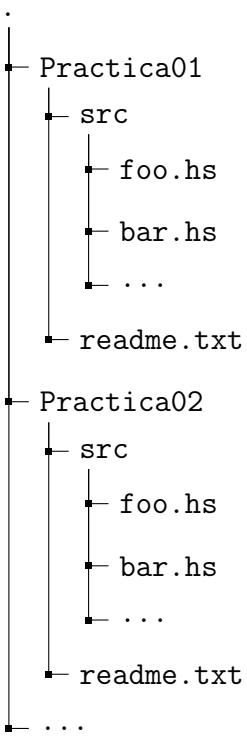
2 Lineamientos

Todas las entregas se harán mediante la plataforma Google Classroom usando gitHub.

Para las prácticas (Equipos de a lo más tres integrantes) deberán crear un repositorio (privado) en github y agregar a los ayudantes como colaboradores: `mildewyPrawn`, `amecoronado`.

Las entregas serán por medio de classroom, deberán adjuntar un enlace a su práctica, agregando en un comentario privado el nombre de los integrantes del equipo, además, los nombres deberán venir en todos los archivos que incluyan código.

Estructuren su repositorio:



Dentro de la entrega se deberá incluir un archivo readme.txt que contenga los nombres de los integrantes del equipo, una breve descripción de cómo se desarrolló la práctica y las dificultades que enfrentaron para resolverla, y una carpeta llamada src la cual contendrá todos los archivos de código fuente. Cada sección de ejercicios deberá ir en un archivo diferente y se recomienda poner un nombre descriptivo a cada archivo.