Práctica 2
Lógica proposicional

Estructuras discretas
Facultad de Ciencias, UNAM

Se desarrollarán en Haskell funciones para trabajar con proposiciones de la lógica de predicados. Para esto se sugiere trabajar con los datos definidos por:

```
data LProp = PTrue | PFalse | Var Nombre | Neg LProp
           | Conj LProp LProp | Disy LProp LProp
           | Impl LProp LProp | Syss LProp LProp
```

Nótese que las fórmulas se definen en notación prefijal, pero la visualización se hará en notación infija. Asimismo, se usará una función de asignación:

```
type Nombre = String
type Asignación = [(String, Bool)]
```

Definir las siguientes funciones:

1. `vars`: Toma como argumento una fórmula y regresa las variables en la fórmula. Por ejemplo: `vars (p∧q)` regresa [”p”, ”q”]

2. `deMorgan`: Toma una fórmula proposicional y regresa su valor aplicando las leyes de De Morgan. Por ejemplo: `deMorgan` $\neg(p \wedge q)$ regresa $(\neg p \vee \neg q)$. Si no se aplica la ley, regresa la fórmula original.

3. `equiv_op`: Dada una fórmula con implicación $\rightarrow$ regresa su equivalente con conectores básicos. Por ejemplo, para $p \rightarrow q$ regresará el valor $\neg p \vee q$.

4. `dobleNeg`: Dada una fórmula con doble negación elimina ésta. Por ejemplo, para $\neg(\neg p)$ regresar p.

5. `num_conectivos`: Función recursiva para contar el número de conectivos lógicos de una fórmula.

6. `num_variables`: Función recursiva para contar el número de variables de una fórmula.

7. `profundidad`: Función recursiva que regresa la profundidad de una fórmula lógica.

8. `interpretación`: Regresa los valores de verdad, True o False, según una asignación. Toma como argumentos una fórmula y una asignación de las variables. Por ejemplo: `interpretación` $(p \wedge q)$ [(”p”, True), (”q”, False)] regresará True.