# Práctica 3
# Lógica computacional 2020-2
# SAT y el algoritmo DPLL.
# Parte 1

Pedro Juan Salvador Sánchez Pérez

Fecha de entrega: **domingo 15 de marzo de 2020**

Haciendo uso del algoritmo DPLL de la nota 4, el objetivo es poder resolver el problema de satisfacibilidad para la lógica proposicional. Hay que recordar que las fórmulas proposicionales deben de estar en forma normal conjuntiva y por eso es necesario terminar el archivo LProp.hs

En esta primera parte se centrarán en implementar las reglas del algoritmo DPLL y en fortalecer su conocimiento sobre las listas por comprensión, ya que será vital su uso para que puedan implementar las reglas fácil y entendible.

Para representar de manera más cómoda las fórmulas, cláusulas y modelos se crearon los siguientes tipos de datos:

```
type Literal = Prop
type Clausula = [Literal]
type Formula = [Clausula]
type Modelo = [Literal]
type Solucion = (Modelo, Formula)
```

## 1. Ejercicios

1. Implementa las funciones `elimEquiv, elimImp, meteNeg y dist`, para que pueda funcionar bien la función `cnf`.

   **Ejemplos**

   - `Practica3*> cnf (Imp (V "p") (Conj (V "r") (V "q")))`
     `(( ∼ ("p") ∨ "r") ∧ ( ∼ ("p") ∨ "q"))`

2. Implementa la función `unit` que dada una `Solucion` regresa otra `Solucion` siguiendo la regla de la cláusula unitaria, si es posible, en otro caso regresa la `Solucion` de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   **Ejemplos**

   - `DPLL> unit ([], [[V "p", V "q"],[V "p"], [Neg (V "r")], [V "r", V "q"]])`
     `(["p"],[["p","q"],[∼("r")],["r","q"]])`
   - `DPLL> unit ([], [[V "p", V "q"], [Neg (V "r")], [V "r", V "q"]])`
     `([∼("r")],[["p","q"],["r","q"]])`
   - `DPLL> unit ([], [[V "p", V "q"],[V "p", V "r"], [V "r", V "q"]])`
     `([],[["p","q"],["p","r"],["r","q"]])`

3. Implementa la función `elim` que dada una `Solucion` regresa otra `Solucion` siguiendo la regla de la eliminación, si es posible, en otro caso regresa la `Solucion` de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   **Ejemplos**

- `DPLL> elim ([V "p"], [[V "p", V "q"], [V "r"], [V "r", Neg (V "p")], [V "r", V "s", V "p"]])`
  `(["p"],[["r"],["r",∼("p")]])`
- `DPLL> elim ([Neg (V "p")], [[V "p", V "q"], [V "r"], [V "r", Neg (V "p")], [V "r", V "s", V "p"]])`
  `([∼("p")],[["p","q"],["r"],["r","s","p"]])`
- `DPLL> elim ([Neg (V "r")], [[V "p", V "q"], [V "r"], [V "r", Neg (V "p")], [V "r", V "s", V "p"]])`
  `([∼("r")],[["p","q"],["r"],["r",∼("p")],["r","s","p"]])`

4. Implementa la función `red` que dada una `Solucion` regresa otra `Solucion` siguiendo la regla de la reducción, si es posible, en otro caso regresa la `Solucion` de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   **Ejemplos**

   - `DPLL> red ([V "p"],[[V "p", V "q"], [Neg (V "p"), V "r"], [V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `(["p"],[["p","q"],["r"],["q",∼("r"),"s"]])`
   - `DPLL> red ([Neg (V "p")],[[V "p", V "q"], [Neg (V "p"), V "r"], [V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `([∼("p")],[["q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]])`
   - `DPLL> red ([V "q"],[[V "p", V "q"], [Neg (V "p"), V "r"], [V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `(["q"],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]])`

5. Implementa la función `split` que dada una `Solucion` regresa una lista de tipo `Solucion` siguiendo la regla de la separación, si es posible, en otro caso regresa la `Solucion` de entrada dentro de una lista. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   **Ejemplos**

   - `DPLL> split ([],[[V "p", V "q"], [Neg (V "p"), V "r"], [V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `[(["p"],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]]),`
     `([∼("p")],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]])]`
   - `DPLL> split ([V "p"],[[V "p", V "q"], [Neg (V "p"), V "r"], [V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `[([∼("r"),"p"],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]]),`
     `(["r","p"],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]])]`
   - `DPLL> split ([V "p", V "r", V "q", V "s"],[[V "p", V "q"], [Neg (V "p"), V "r"],`
     `[V "q", Neg (V "r"), Neg (V "p"), V "s"]])`
     `[(["p","r","q","s"],[["p","q"],[∼("p"),"r"],["q",∼("r"),∼("p"),"s"]])]`

6. Implementa la función `conlfict` siguiendo la regla de conflicto, que para una `Solucion` dada regresa True si la cláusula vacía se encuentra en la fórmula, regresa False en otro caso.

   **Ejemplos**

   - `DPLL> conflict ([V "p"], [[]])`
     `True`
   - `DPLL> conflict ([V "p"], [[V "q"]])`
     `False`

7. Implementa la función `success` siguiendo la regla de éxito, que para una `Solucion` dada regresa True si la fórmula no contiene cláusulas, regresa False en otro caso.

   **Ejemplos**

   - `DPLL> success ([V "p"], [])`
     `True`
   - `DPLL> false ([V "p"], [[V "q"]])`
     `False`

# NOTA

Recuerden que deben seguir los lineamientos establecidos para la entrega de su práctica, los cuales se encuentran en la página del curso.