Lógica Computacional 2017-1
Práctica 1

Noé Salomón Hernández Sánchez
Albert M. Orozco Camacho
C. Moisés Vázquez Reyes
Diego Murillo Albarrán
José Roberto Piche Limeta

Se deja el: 1 de septiembre
Se entrega el: **15 de septiembre**
Facultad de Ciencias UNAM

En esta práctica van a implementar dos mecanismos para verificar si una fórmula escrita en lógica proposicional es correcta. Utilizaremos el siguiente tipo para representar fórmulas dentro de Haskell:

```
data Form = Conj Form Form | Disy Form Form | Imp Form Form | DImp Form Form | Neg Form | Var String | T
| F
```

1. Interpretaciones

Un *estado* es una asignación que toma una variable proposicional y le asigna un valor. Por ejemplo, podemos pensar en el estado $\{(p,0),(q,1),(r,1)\}$. Una manera más compacta de representar esto es utilizar una lista de variables proposicionales. Si una variable está en la lista, se le asigna 1; en otro caso, se le asiga 0. De esta manera, podemos representar el estado $\{(p,0),(q,1),(r,1)\}$ como la lista $[q,r]$.

Una *interpretación* es una función que toma una fórmula proposicional junto con un estado y realiza la evaluación de dicha fórmula. Por ejemplo, si tenemos el estado $\sigma = [p,q]$, entonces:

$$\mathcal{I}_\sigma(p \to (q \vee r)) = 1 \to (1 \vee 0) = 1 \to 1 = 1$$

$$\mathcal{I}_\sigma((s \wedge p) \vee r) = (0 \wedge 1) \vee 0 = 0 \vee 0 = 0$$

La representación de estados en Haskell será con el tipo **type** `Estado = [String]`.

- `interp::Estado->Form->Bool`

  Función que toma un estado y evalúa una fórmula proposicional.

  - `>interp [‘‘p’’] $ Conj (Var ‘‘q’’) (Disy (Var ‘‘r’’) (Var ‘‘p’’))`
    `False`
  - `>interp [‘‘p’’,‘‘q’’] $ Conj (Var ‘‘q’’) (Disy (Var ‘‘r’’) (Var ‘‘p’’))`
    `True`

- `vars::Form->[String]`

  Devuelve todas las variables de una fórmula proposicional sin repeticiones.

  - `>vars $ Conj (Var ‘‘q’’) (Disy (Var ‘‘r’’) (Var ‘‘p’’))`
    `[‘‘q’’,‘‘r’’,‘‘p’’]`

- `>vars $ Conj (Var ‘‘q’’) (Disy (Var ‘‘r’’) (Var ‘‘q’’))`
  [‘‘q’’,‘‘r’’]

- `potencia::[a]->[[a]]`

  Calcula todas las sublistas de una lista.

  - `>potencia [1,2]`
    $[[], [2], [1], [1, 2]]$
  - `>potencia []`
    [[]]

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

  Dada una fórmula proposicional, la función devuelve todos los estados con los que podemos evaluar la fórmula.

  - `>estados $ Conj (Var ‘‘p’’) T`
    [[],[‘‘p’’]]
  - `>estados $ Disy (Var ‘‘q’’) (Conj (Var ‘‘r’’) (Var ‘‘q’’))`
    [[],[‘‘q’’],[‘‘r’’],[‘‘q’’,‘‘r’’]]

- `tautologia::Form->Bool`

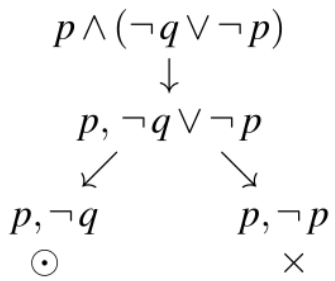
  Nos dice si una fórmula es tautología.

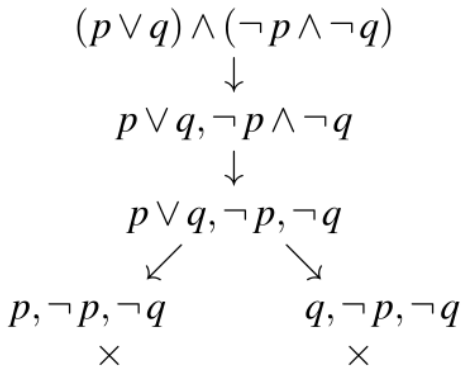
  - `>tautologia $ Disy (Var ‘‘p’’) (Neg $ Var ‘‘p’’)`
    `True`
  - `>tautologia $ Disy (Var ‘‘q’’) (Var ‘‘r’’)`
    `False`

2. Tableaux semánticos

Un tableaux semántico es un árbol que sirve para determinar si una fórmula tiene un modelo, es decir, un estado de las variables que satisface la fórmula. El nodo raíz contiene a la lista de fórmulas iniciales, cada nodo interno contiene una lista de fórmulas, y cada hoja contiene una lista de literales. Si una hoja contiene literales complementarias, ésta se marca con un TACHE; de lo contrario, se marca con una BOLITA.

Mostramos dos ejemplos:





Las reglas para construir un tableaux son las siguientes:

- α-reglas:
  - Si un nodo contiene una lista de la forma $[\neg\neg f : fs]$, cambiarla por la lista $[f : fs]$.
  - Si la lista que contiene es de la forma $[(f_1 \wedge f_2) : fs]$, crear un nodo hijo que contenga a la lista $[f_1 : (f_2 : fs)]$.
  - Si la lista que contiene es de la forma $[\neg(f_1 \vee f_2) : fs]$, crear un nodo hijo que contenga a la lista $[\neg f_1 : (\neg f_2 : fs)]$.
  - Si la lista que contiene es de la forma $[\neg(f_1 \to f_2) : fs]$, crear un nodo hijo que contenga a la lista $[f_1 : (\neg f_2 : fs)]$.
  - Si la lista que contiene es de la forma $[(f_1 \leftrightarrow f_2) : fs]$, crear un nodo hijo que contenga a la lista $[(f_1 \to f_2) : ((f_2 \to f_1) : fs)]$.
- β-reglas:
  - Si la lista que contiene es de la forma $[\neg(f_1 \wedge f_2) : fs]$, crear dos nodos hijo, uno que contenga a la lista $[\neg f_1 : fs]$ y otro que contenga a la lista $[\neg f_2 : fs]$.
  - Si la lista que contiene es de la forma $[(f_1 \vee f_2) : fs]$, crear dos nodos hijo, uno que contenga a la lista $[f_1 : fs]$ y otro que contenga a la lista $[f_2 : fs]$.
  - Si la lista que contiene es de la forma $[(f_1 \to f_2) : fs]$, crear dos nodos hijo, uno que contenga a la lista $[\neg f_1 : fs]$ y otro que contenga a la lista $[f_2 : fs]$.
  - Si la lista que contiene es de la forma $[\neg(f_1 \leftrightarrow f_2) : fs]$, crear dos nodos hijo, uno que contenga a la lista $[\neg(f_1 \to f_2) : fs]$ y otro que contenga a la lista $[\neg(f_2 \to f_1) : fs]$.

2.1. Algoritmo para construir tableaux (Ben-Ari)

Entrada: Una fórmula proposicional φ.
Salida: Un tableaux **T** cuyas hojas están todas etiquetadas con TACHE o BOLITA.
Inicialmente, T consiste únicamente del nodo raíz que contiene a la fórmula φ.
Repetir lo siguiente tanto como sea posible: elige una hoja h que aún no haya sido marcada y aplica una de las siguientes reglas:

- *Si la lista de fórmulas que contiene h está conformada únicamente por literales, hacer lo siguiente:*
  - *Si contiene literales complementarias, marcar h con* TACHE.
  - *Si no contiene literales complementarias, marcar h con* BOLITA.
- *En otro caso, elegir una fórmula en la lista contenida en h que no sea una literal; dependiendo de la fórmula, aplicar α-reglas o β-reglas y repetir en todos los nodos generados.*

Decimos que un tableaux está *cerrado* si todas sus hojas están marcadas con TACHE. Un tableaux cerrado indica que la fórmula inicial es una contradicción, pues no tiene modelos.

Para comprobar que una fórmula φ es una tautología, basta hacer el tableaux para $\neg\varphi$, si todas sus ramas se cierran, entonces φ es una tautología.

Para representar tableaux semánticos en Haskell Utilizaremos el tipo:

```
data Tableaux = Void | R1 [Form] Tableaux | R2 Tableux Tableaux | Tache | Bolita
```

Void sirve para indicar cuáles hojas aún no han sido marcadas. *R1* sirve para encapsular a la fórmula inicial y a todos los nodos generados con α-reglas. *R2* sirve para los nodos generados a partir de β-reglas.

- `creaTableaux::[Form]->Tableaux`

  Crea un tableaux a partir de una lista de fórmulas proposicionales.

  - `>creaTableaux [Conj (Disy (Var ‘‘p’’) (Var ‘‘q’’)) (Conj (Neg $ Var ‘‘p’’) (Neg $ Var ‘‘q’’))]`
    ```
    R1 [Conj (Disy (Var "p") (Var "q")) (Conj (Neg (Var "p")) (Neg (Var "q")))]
        (R1 [Conj (Neg (Var "p")) (Neg (Var "q")),Disy (Var "p") (Var "q")]
          (R1 [Disy (Var "p") (Var "q"),Neg (Var "p"),Neg (Var "q")]
            (R2
              (R1 [Neg (Var "p"),Neg (Var "q"),Var "p"] Tache)
              (R1 [Neg (Var "p"),Neg (Var "q"),Var "q"] Tache))))
    ```

- `cerrado::Tableaux->Bool`

  Indica si un tableaux tiene todas sus hojas etiquetadas con TACHE.

  - `>cerrado $ creaTableux $ Conj (Disy (Var ‘‘p’’) (Var ‘‘q’’)) (Conj (Neg $ Var ‘‘p’’) (Neg $ Var ‘‘q’’))`
    `True`
  - `>cerrado $ creaTableux $ Conj (Disy (Var ‘‘p’’) (Var ‘‘q’’)) (Var ‘‘r’’)`
    `False`

- `tautologia_tableaux::Form->Bool`

  Indica si una fórmula es tautología a partir del tableaux generado por dicha fórmula.

  - `>tautologia_tableaux $ Disy (Var ‘‘p’’) (Neg $ Var ‘‘p’’)`
    `True`
  - `>tautologia_tableaux $ Disy (Var ‘‘q’’) (Var ‘‘r’’)`
    `False`