

Facultad de Ciencias - UNAM  
Lógica Computacional 2026-2  
Práctica 2: Conceptos semánticos

Favio Ezequiel Miranda Perea      Patricio Ordoñez Blanco      Eduardo Vargas Pérez

24 de Febrero de 2026  
**Fecha de entrega:** 5 de Marzo de 2026

## 1 Introducción

En la parte teórica se ha estudiado la sintaxis y la semántica de la lógica proposicional. Esto se ha hecho (principalmente) desde una perspectiva computacional, por lo que ahora es turno de programar algunos conceptos vistos.

## 2 Objetivo

Por medio de la implementación práctica, se espera que logren reforzar algunos de los conceptos semánticos como lo son interpretación, modelo, tautología, satisfacibilidad, entre otros.

## 3 Desarrollo de la práctica

### 3.1 Sintaxis de la lógica proposicional

Para poder trabajar con la lógica proposicional en `Haskell`, hay que realizar una forma de representar las fórmulas. Para esto, se creará el siguiente tipo de dato algebraico:

```
data Prop = Var String | Cons Bool | Not Prop
          | And Prop Prop | Or Prop Prop
          | Impl Prop Prop | Syss Prop Prop
          deriving (Eq)
```

Adicionalmente, para facilitar la lectura de las fórmulas en la terminal, se debe agregar el siguiente código que le dice a *Haskell* cómo debe imprimir un tipo de dato `Prop`.

```

instance Show Prop where
.      show Cons True = "⊤"
.      show Cons False = "⊥"
.      show (Var p) = p
.      show (Not p) = "¬" ++ show p
.      show (Or p q) = "(" ++ show p ++ " ∨ " ++ show q ++ ")"
.      show (And p q) = "(" ++ show p ++ " ∧ " ++ show q ++ ")"
.      show (Impl p q) = "(" ++ show p ++ " → " ++ show q ++ ")"
.      show (Syss p q) = "(" ++ show p ++ " ↔ " ++ show q ++ ")"

```

Ahora, se definen las siguientes fórmulas proposicionales como variables atómicas:  $p, q, r, s, t, u$ .

```

p, q, r, s, t, u :: Prop
p = Var "p"
q = Var "q"
r = Var "r"
s = Var "s"
t = Var "t"
u = Var "u"

```

En esta práctica, un estado será implementado usando una lista. Para esto, se define el siguiente sinónimo:

```

type Estado = [String]

```

Por ejemplo, si se tiene una función de interpretación  $I_1$  tal que  $I_1(p) = I_1(q) = 1$  e  $I_1(r) = 0$ , entonces se implementa dicho estado como  $i_1 = ["p", "q"]$ . De manera análoga, el estado  $i_2 = ["r", "s", "t"]$  representa a una función de interpretación  $I_2$  tal que  $I_2(r) = I_2(s) = I_2(t) = 1$ . Dicho de otro modo, de presentarse otra variable diferente a las pertenecientes a la lista proporcionada, se supondrá que su valor bajo la función de interpretación es cero.

## Ejercicios

Implementa las siguientes funciones:

1. **[1.5 puntos]** Define la función **variables :: Prop -> [String]** tal que **variables  $f$**  devuelve el conjunto formado por todas las variables proposicionales que aparecen en  $f$ . Por ejemplo:

```

> variables (Impl (And (Var "p") (Var "q")) (Var "p"))
["p","q"]
> variables (And (Var "q") (Or (Var "r") (Var "p")))
["q","r","p"]

```

**Nota:** No deben haber elementos repetidos en el resultado.

2. **[2 puntos]** Define la función **interpretacion :: Prop -> Estado -> Bool** tal que **interpretacion  $f i$**  es la interpretación de la fórmula  $f$  bajo  $i$ . Por ejemplo:

```

> interpretacion (And (Var "q") (Or (Var "r") (Var "p"))) ["p"]
False

```

- ```
> interpretacion (And (Var "q") (Or (Var "r") (Var "p"))) ["p","q"]
True
```
3. **[1 punto]** Define una función que dada una fórmula proposicional, devuelve todos los estados posibles con los que podemos evaluar la fórmula. Por ejemplo:
- ```
> estadosPosibles (Or (Var "q") (And (Var "r") (Var "q")))
[[],["r"],["q"], ["q","r"]]
```
4. **[1.5 puntos]** Define una función que dada una fórmula proposicional, esta devuelve la lista de todos sus modelos. Por ejemplo:
- ```
> modelos (Or (Var "q") (And (Var "r") (Var "q")))
[["q"], ["q","r"]]
```
5. **[1 punto]** Define una función que dadas dos fórmulas  $\varphi_1$  y  $\varphi_2$  de la lógica proposicional, nos diga si  $\varphi_1$  y  $\varphi_2$  son equivalentes. Por ejemplo:
- ```
> sonEquivalentes (Or (Var "q") (Not (Var "p"))) (Impl (Var "p") (Var "q"))
True
> sonEquivalentes (And (Var "q") (Not (Var "p"))) (Impl (Var "p") (Var "q"))
False
```
6. **[0.5 puntos]** Define una función que dada una fórmula proposicional, nos diga si es una tautología. Por ejemplo:
- ```
> tautologia (Or (Var "p") (Not (Var "p")))
True
> tautologia (Or (Var "q") (Var "r"))
False
```
7. **[0.5 puntos]** Define una función que dada una fórmula proposicional, nos diga si es una contradicción. Por ejemplo,
- ```
> contradiccion (And (Var "p") (Not (Var "p")))
True
> contradiccion (Or (Var "q") (Var "r"))
False
```
8. **[2 puntos]** Definir una función que reciba una lista de fórmulas de la lógica proposicional y una fórmula. Esta función debe determinar si la fórmula recibida es consecuencia lógica de la lista de fórmulas recibida. Por ejemplo:
- ```
> consecuenciaLogica [Impl (Var "p") (Var "q"), Var "p"] (Var "q")
True
> consecuenciaLogica [Impl (Var "p") (Var "q"), Var "q"] (Var "p")
False
```

## Sugerencia

No se distribuyan los ejercicios entre el equipo, pues seguramente van a necesitar de los primeros ejercicios para resolver los últimos y es posible que programen doble si los hacen en desorden. Vean de qué manera pueden ocupar las primeras funciones para resolver el resto de la práctica.

## Limitaciones

Recuerden que toda función auxiliar que necesiten tiene que ser implementada por ustedes.

¡Buena suerte a todos! ☺☺☺