Lógica Computacional, 2019-2
Nota de laboratorio 5: Semántica de la Lógica Proposicional*

Manuel Soto Romero

6 de marzo de 2019

Facultad de Ciencias, UNAM

En esta nota se revisa brevemente la implementación de los principales conceptos semánticos asociados al lenguaje de la Lógia Proposicional, lo cual permite entre otras cosas, verificar si un argumento es correcto o no.

1. Semántica formal

Definición 1. Un *estado* de las variables (proposicionales) es una función

$$I : VarP \to Bool$$

Usando la implementación de PROP de la Nota de laboratorio 4, es posible definir un estado I directamente como:

```
i :: ATOM -> Bool
```

o bien, indicando simplemente que un estado, es una variable que se encuentra definida como verdadera, mediante el sinónimo `Estado`:

```
type Estado = ATOM
```

Definición 2. Dado un estado de las variables $I : VarP \to Bool$, se define la interpretación de las fórmulas con respecto a I como la función $I^* : PROP \to Bool$ tal que:

1. $I^*(p) = I(p)$ para $p \in VarP$, es decir $I^* \restriction_{VarP} = I$.
2. $I^*(\top) = 1$
3. $I^*(\bot) = 0$
4. $I^*(\neg\varphi) = 1$ syss $I^*(\varphi) = 0$.
5. $I^*(\varphi \wedge \psi) = 1$ syss $I^*(\varphi) = I^*(\psi) = 1$.
6. $I^*(\varphi \vee \psi) = 0$ syss $I^*(\varphi) = I^*(\psi) = 0$.
7. $I^*(\varphi \to \psi) = 0$ syss $I^*(\varphi) = 1$ e $I^*(\psi) = 0$.

*Basado en las *Notas para el curso de Lógica Computacional* del Dr. Favio E. Miranda, et al., revisión 2017-2.

8. $I^*(\varphi \leftrightarrow \psi) = 1$ syss $I^*(\varphi) = I^*(\psi)$.

El Listado de código 1 muestra la implementación de la Definición 2 mediante la función `interp` (líneas 9 a 16). Esta función recibe una lista de los posibles estados para las variables y devuelve un valor booleano (0 o 1). Para buscar en la lista de posibles estados se define la función `busca` (líneas 2 a 6).

```
1  -- Función que busca el valor de una variable en una lista de estados.
2  busca :: String -> [Estado] -> Bool
3  busca v [] = False
4  busca v ((Var x):xs)
5      | v == x = True
6      | otherwise = busca v xs
7
8  -- Función que realiza la interpretación de fórmulas proposicionales.
9  interp :: PROP -> [Estado] -> Bool
10 interp (FA (Cte c)) _  = c
11 interp (FA (Var v)) xs = busca v xs
12 interp (Neg p) xs      = not (interp p xs)
13 interp (Conj p q) xs   = interp p xs && interp q xs
14 interp (Disy p q) xs   = interp p xs || interp q xs
15 interp (Impl p q) xs   = not (interp p xs) || interp q xs
16 interp (Syss p q) xs   = interp p xs == interp q xs
```

Listado de código 1: Interpretación de fórmulas proposicionales

2. Conceptos semánticos importantes

Antes de analizar algunas definiciones importantes, es necesario definir una función `estados` que dada una fórmula proposicional, obtiene una lista con todos los posibles estados de cada una de sus variables. La implementación de esta función se muestra en el Listado de código 2 (líneas 9 y 10) y hace uso de la función `subconjuntos` (líneas 2 a 5) que dada una lista obtiene todas las sublistas asociadas a la misma.

```
1  -- Función que obtiene las sublistas de una lista.
2  subconjuntos :: [a] -> [[a]]
3  subconjuntos []     = [[]]
4  subconjuntos (x:xs) = [x:ys | ys <- xss] ++ xss
5      where xss = subconjuntos xs
6
7  -- Función que obtiene los posibles estados de las variables de una
8  -- fórmula proposicional.
9  estados :: PROP -> [[Estado]]
10 estados p = subconjuntos (vars p)
```

Listado de código 2: Estado de las variables de una fórmula proposicional

Dada una fórmula proposicional φ es posible preguntarse ¿Cuántas interpretaciones hacen verdadera a φ? las posibles respuestas llevan a las siguientes definiciones:

Definición 3. Sea φ una fórmula proposicional. Entonces

- Si $I(\varphi) = 1$ para toda interpretación I, se dice que φ es una *tautología y se escribe* $\models \varphi$.
- Si $I(\varphi) = 1$ para alguna interpretación I, se dice que φ es *satisfacible* o que I es modelo de φ y se escribe $I \models \varphi$.
- Si $I(\varphi) = 0$ para alguna interpretación I, se dice que φ es *insatisfacible* o que I no es modelo de φ y se escribe $I \not\models \varphi$.
- Si $I(\varphi) = 0$ para toda interpretación I, se dice que φ es una *contradicción*.

La implementación de la Definición 3 puede darse mediante el uso de la función `modelos` que se define en el Listado de código 3 (lineas 2 y 3) que dada una fórmula φ regresa la lista de todos sus modelos, es decir, devuelve la lista con todos aquellos estados I tales que $I(\varphi) = 1$.

```
1 -- Lista de modelos de una fórmula proposicional.
2 modelos :: PROP -> [[Estado]]
3 modelos f = [i | i <- estados f, interp f i == True]
```

Listado de código 3: Modelos de una fórmula proposicionall

La implementación de la Definición 3 se deja como ejercicio al lector.

3. Correctud de argumentos lógicos

Se dice que un argumento es correcto si y sólo sí su fórmula proposicional asociada es una tautología. Por ejemplo, dado el siguiente argumento:

1. Si Javier estudia adecuadamente, entonces Javier aprueba el curso de Modelado y Programación.
2. Javier está estudiando adecuadamente.
3. Javier aprobará el curso de Modelado y Programación.

Al formalizarlo se obtiene:

$p =$ Javier estudia adeducadamente.
$q =$ Javier aprueba el curso de Modelado y Programación.

$$p \to q, p / \therefore q$$

Para probar que se trata de un argumento correcto, se debe probar que la fórmula $(p \to q) \wedge p \to q$ es una tautología, de donde se tienen las siguientes interpretaciones:

$I_1(p) = 0, I_1(q) = 0$ hacen $I_1((p \to q) \wedge p \to q) = 1$
$I_2(p) = 0, I_2(q) = 1$ hacen $I_2((p \to q) \wedge p \to q) = 1$
$I_3(p) = 1, I_3(q) = 0$ hacen $I_3((p \to q) \wedge p \to q) = 1$
$I_4(p) = 1, I_1(q) = 1$ hacen $I_4((p \to q) \wedge p \to q) = 1$

Por lo tanto como $I_i((p \to q) \wedge p \to q) = 1$ para toda interpretación I_i, se puede concluir que es una tautología y por lo tanto el argumento al que representa es correcto.

¿Cómo pueden usarse estos conceptos en Haskell para decidir si un argumento es correcto?

Ejemplo 1. Sea la fórmula proposicional $(p \to q) \wedge p \to q$ expresada en Haskell:

```
module Ejemplo1 where

-- Variables

-- Las estrellas emiten luz.
p = (FA (Var "p"))
-- Los planetas reflejan luz.
q = (FA (Var "q"))

-- Especificación formal

expr = Impl (Conj (Impl p q) p) q
```

Para saber si un argumento es correcto, basta verificar que es verdadero para toda interpretación, es decir que es una tautología.

```
sol = esTautologia expr
```

□