Lógica Computacional, 2019-2
Nota de laboratorio 4: Sintaxis de la Lógica Proposicional*

Manuel Soto Romero

27 de febrero de 2019
Facultad de Ciencias, UNAM

Como todo sistema lógico, la Lógica Proposicional, cuenta con un lenguaje formal que se usa como medio de expresión mismo al que se conoce como sintaxis, de la misma forma cuenta con un mecanismo que proporciona significado al lenguaje formal dado por la sintaxis conocido como semántica y finalmente con una teoría de prueba. En esta nota se estudiará la sintaxis de la Lógica Proposicional desde un punto de vista complementamente práctico, a través de su definición en el lenguaje de programación Haskell.

1. El lenguaje de la Lógica Proposicional

Sea `ATOM` el conjunto de expresiones o fórmulas atómicas formado por:

- Las variables proposicionales: $p_1, ..., p_n, ...$
- Las constantes lógicas: $\bot, \top$

El lenguaje de la lógica proposicional, mejor conocido como `PROP` o `LPROP`, se define recursivamente como sigue:

1. Si $\varphi \in$ `ATOM` entonces $\varphi \in$ `PROP`. Es decir, toda fórmula atómica es una fórmula proposicional.
2. Si $\varphi \in$ `PROP` entonces $(\neg\varphi) \in$ `PROP`.
3. Si $\varphi, \psi \in$ `PROP` entonces
   $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi) \in$ `PROP`.

Este lenguaje puede expresarse en Haskell mediante la definición de tipos de datos. El Listado de código 1 muestra la definición de los tipos de datos `ATOM` y `PROP`. Estos tipos no derivan de la clase `Show`, pues más adelante se proporciona una implementación que hace más amigable al usuario, la impresión de los mismos.

Para representar al conjunto `ATOM` se define el tipo de dato del mismo nombre (líneas 4 y 5), se incluyen dos constructores: `Var` y `Cte`, el primero para representar variable proposicionales, mediante un parámetro de tipo `String` y el segundo para representar a las constantes lógica mediante un parámetro de tipo `Bool`.

Para representar el lenguaje `PROP` se define el tipo de dato del mismo nombre (líneas 9 a 14), se incluyen seis constructores, el primero permite representar fórmulas atómicas de tipo `ATOM`, y los restantes representan cinco operaciones básicas cuyos parámetros son de tipo `PROP`, es decir, se trata de un tipo recursivo.

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

```haskell
module Prop where

   -- Tipo de dato que representa el conjunto de fórmulas atómicas.
   data ATOM = Var String
             | Cte Bool

   -- Tipo de dato que representa al lenguaje de la lógica
   -- proposicional.
   data PROP = FA ATOM
             | Neg PROP
             | Conj PROP PROP
             | Disy PROP PROP
             | Impl PROP PROP
             | Syss PROP PROP
```

Listado de código 1: Representación de la sintaxis del lenguaje de la lógica proposicional en Haskell

Para mostrar a los tipos de datos `ATOM` y `PROP` en la línea de comandos, se implementa la función `show` correspondiente a través de la primitiva `instance`. El Listado de código 2 muestra la implementación para ambos tipos de datos.

```haskell
-- Implementación de la función show para el tipo ATOM.
instance Show ATOM where
   show (Var p) = p
   show (Cte c) = show c

-- Implementación de la función show para el tipo PROP.
instance Show PROP where
   show (FA f)     = show f
   show (Neg p)    = "no " ++ show p
   show (Conj p q) = "(" ++ show p ++ " & "  ++ show q ++ ")"
   show (Disy p q) = "(" ++ show p ++ " | "  ++ show q ++ ")"
   show (Impl p q) = "(" ++ show p ++ " -> "  ++ show q ++ ")"
   show (Syss p q) = "(" ++ show p ++ " <-> " ++ show q ++ ")"
```

Listado de código 2: Implementación de la función show de ATOM y PROP

La función `show` regresa siempre una cadena, por lo que una vez instanciados los tipos de datos `ATOM` y `PROP`, se pueden mostrar en consola. La firma de las dos funciones definidas es:

```haskell
show :: ATOM -> String
show :: PROP -> String
```

Una vez definidos estos tipos de datos y de hacerlos parte de la clase de tipos Show, es posible definir funciones sobre los mismos. Para definir funciones sobre estos tipos de datos, es recomendable usar la técnica de reconocimiento de patrones.

Observación 1. Es importante mencionar que la técnica de reconocimiento de patrones se basa en el principio de inducción estructural, por lo que la definición de funciones sigue un proceso similar al proceso de demostración usando este principio.

Ejemplo 1. Definir la función `con` tal que (`con p`) es el número de conectivos que figuran en la fórmula proposicional `p`. Por ejemplo,

```
con (FA (Cte True))                                  = 0
con (Neg (FA (Var "p")))                             = 1
con (Conj (FA (Cte True)) (Neg (FA (Var "p")))) = 2
```

Solución. Función `con`:

```
1 -- Número de conectivos de una fórmula proposocional.
2 con :: PROP -> Int
3 con (FA _)      = 0
4 con (Neg p)     = 1 + con p
5 con (Conj p q) = 1 + con p + con q
6 con (Disy p q) = 1 + con p + con q
7 con (Impl p q) = 1 + con p + con q
8 con (Syss p q) = q + con p + con q
```

Listado de código 3: Número de conectivos de una fórmula proposicional

□

2. Sustitución textual

Una operación elemental en cuanto a la sintaxis de la Lógica Proposocicional se refiere es la sustitución textual que se define recursivamente como sigue:

$\bot [p := \psi] = \bot$
$\top [p := \psi] = \top$
$p [p := \psi] = \psi$
$q [p := \psi] = q$ si $p \neq q$
$(\neg\varphi) [p := \psi] = \neg (\varphi [p := \psi])$
$(\varphi \wedge \chi) [p := \psi] = (\varphi [p := \psi] \wedge \chi [p := \psi])$
$(\varphi \vee \chi) [p := \psi] = (\varphi [p := \psi] \vee \chi [p := \psi])$
$(\varphi \to \chi) [p := \psi] = (\varphi [p := \psi] \to \chi [p := \psi])$
$(\varphi \leftrightarrow \chi) [p := \psi] = (\varphi [p := \psi] \leftrightarrow \chi [p := \psi])$

El Listado de código 4 muestra la implementación de esta operación en Haskell.

```haskell
-- Sustitución textual
sustTxt :: PROP -> (String, PROP) -> PROP
sustTxt (FA (Cte c)) _ = (FA (Cte c))
sustTxt (FA (Var v)) (x,y) = if v == x then y else (FA (Var v))
sustTxt (Neg p) st = Neg (sustTxt p st)
sustTxt (Conj p q) st = Conj (sustTxt p st) (sustTxt q st)
sustTxt (Disy p q) st = Disy (sustTxt p st) (sustTxt q st)
sustTxt (Impl p q) st = Impl (sustTxt p st) (sustTxt q st)
sustTxt (Syss p q) st = Syss (sustTxt p st) (sustTxt q st)
```

Listado de código 4: Sustitución textual

Ejemplo 2. Realizar la siguiente sustitución textual

$$(p \to q \wedge r)\,[q := s \wedge t]$$

Solución. Sustitución textual paso a paso:

$(p \to q \wedge r)\,[q := s \wedge t] = (p\,[q := s \wedge t] \to (q \wedge r)\,[q := s \wedge t])$
$(p\,[q := s \wedge t] \to (q \wedge r)\,[q := s \wedge t]) = (p \to (q\,[q := s \wedge t] \wedge r\,[q := s \wedge t]))$
$(p \to (q\,[q := s \wedge t] \wedge r\,[q := s \wedge t])) = (p \to s \wedge t \wedge r)$

En Haskell:

```
Prop> sustTxt (Impl (FA (Var "p")) (Conj (FA (Var "q")) (FA (Var "r")))) ("q",
Conj (FA (Var "s")) (FA (Var "r")))
(p -> ((s & t) & r))
```

□

Análogamente se define la sustitución simultánea de n variables por n fórmulas proposicionales:

$$\varphi\,[p_1, ..., p_n := \psi_1, ..., \psi_n]$$

```haskell
sustSimult :: PROP -> [(String, PROP)] -> PROP
```

¿Cómo se define recursivamente esta operación?

3. Especificación formal

A continuación se presental la traducción de algunos enunciados en lenguaje natural al lenguaje de la Lógica Proposicional a través de Haskell.

Ejemplo 3. Dados los siguientes enunciados en lenguaje natural, realizar la especificación formal correspondiente indicando el valor de cada variable utilizada y traducirlos a la expresión correspondiente en Haskell.

1. Si las estrellas emiten luz, entonces los planetas la reflejan y giran alrededor de ellas.
2. Las estrellas emiten luz o los planetas la reflejan y, por otra parte, los planetas giran al rededor de ellas.
3. Los planetas reflejan luz si y sólo si las estrellas la emiten y los planetas giran al rededor de ellas.
4. Si no es cierto que las estrellas emiten luz y que los planetas la reflejan, entonces éstos no giran al rededor de ellas.

Solución 1. Variables:

p Las estrellas emiten luz.
q Los planetas reflejan luz
r Los planetas giran al rededor de las estrellas.

Especificación formal:

1. $p \to (q \wedge r)$
2. $(p \vee q) \wedge r$
3. $q \leftrightarrow (p \wedge r)$
4. $\neg (p \wedge q) \to \neg r$

Traducción a Haskell:

```haskell
module Ejemplo3 where

-- Variables

-- Las estrellas emiten luz.
p = (FA (Var "p"))
-- Los planetas reflejan luz.
q = (FA (Var "q"))
-- Los planetas giran al rededor de las estrellas.
r = (FA (Var "r"))
```

```
11
12 -- Especificación formal
13
14 -- Si las estrellas emiten luz, entonces los planetas la reflejan
15 -- y giran alrededor de ellas.
16 expr1 = Impl p (Conj q r)
17
18 -- Las estrellas emiten luz o los planetas la reflejan y, por otra
19 -- parte, los planetas giran al rededor de ellas.
20 expr2 = Conj (Disy p q) r
21
22 -- Los planetas reflejan luz si y sólo si las estrellas la emiten y
23 -- los planetas giran al rededor de ellas.
24 expr3 = Syss q (Conj p r)
25
26 -- Si no es cierto que las estrellas emiten luz y que los planetas
27 -- la reflejan, entonces éstos no giran al rededor de ellas.
28 expr4 = Impl (Neg (Conj p q)) (Neg r)
```

Ejecución:

```
Ejemplo3> expr1
(p -> (q & r))
Ejemplo3> expr2
((p | q) & r)
Ejemplo3> expr3
(q <-> (p & r))
Ejemplo3> expr4
(no (p & q) -> no r)
```

□