

Introducción a los Algoritmos - 1er. cuatrimestre 2011

Guía 3: Lógica Proposicional

Docentes: Araceli Acosta, Carlos Areces, Mariana Badano, Luciana Benotti, Javier Blanco, Paula Estrella, Pedro Sánchez Terraf, Mauricio Tellechea.

La lógica proposicional puede usarse como una formalización elemental del razonamiento deductivo. El objetivo general de esta guía es introducirnos a la lógica proposicional, como herramienta para describir problemas y razonar sobre los mismo.

En esta práctica, como en el resto de las prácticas de esta materia, los ejercicios marcados como (!) son algo mas complejos que el resto; y son modelos de los ejercicios que se tomarán en los parciales. Los ejercicios marcados como ☺ son para las almas inquietas que quieren explorar más alla de los contenidos que veremos en la materia.

Evaluación, sustitución sintáctica y tipado

La lógica proposicional amplía el lenguaje que venimos manejando, introduciendo dos constantes que representan los valores de verdad (*True* y *False*) y diversos operadores booleanos sobre estos valores ($\wedge, \vee, \neg, \equiv, \Rightarrow$). Esta lógica nos permite escribir expresiones que codifican propiedades mas interesantes, como por ejemplo que “si un número a es mayor a b y, además, b es mayor a c , entonces a tiene que ser mayor a c ”:

$$(a > b) \wedge (b > c) \Rightarrow (a > c)$$

Al complejizarse el lenguaje, es necesario establecer reglas de notación para poder escribir las expresiones con mayor claridad y sin ambigüedades. En particular, es posible eliminar paréntesis cuando los operadores involucrados son asociativos (i.e. no importa el orden en que asocie los operandos), y según las reglas de precedencia, tal como hemos visto hasta ahora.

Los operadores $\wedge, \vee, \equiv, \neq$ son asociativos. Esto nos permite establecer por ejemplo que las expresiones $p \wedge q \wedge r$, junto con $p \wedge (q \wedge r)$ y $(p \wedge q) \wedge r$ son equivalentes.

A la derecha se listan los nuevos operadores y aquellos vistos anteriormente, en orden de mayor a menor precedencia.

En los siguientes ejercicios comenzamos trabajando principalmente con los aspectos **sintácticos** del nuevo lenguaje, para asegurar una buena capacidad de lectura y escritura de las nuevas expresiones, y con la **semántica** de los nuevos operadores, es decir, su significado.

Niveles de Precedencia

1	$\sqrt{\cdot}, (\cdot)^2$	raíces y potencias
2	$*, /$	producto y división
3	\max, \min	máximo y mínimo
4	$+, -$	suma y resta
5	$=, \leq, \geq$	operadores de comparación
6	\neg	negación
7	$\vee \wedge$	disyunción y conjunción
8	$\Rightarrow \Leftarrow$	implicación y consecuencia
9	$\equiv \neq$	equivalencia y discrepancia

1. Sacá todos los paréntesis que sean *superfluos* según las reglas de precedencia de los operadores booleanos.

a) $((((a = b) \wedge (b = c)) \Rightarrow (a = c)) \equiv True)$.

b) $((p \Rightarrow q) \wedge (q \Rightarrow p)) \Rightarrow (p \equiv q)$.

c) $((p \wedge q) \vee (\neg r)) \Rightarrow (p \wedge (q \vee r))$.

2. Introducí paréntesis para hacer *explícita* la precedencia.

a) $p \vee q \Rightarrow r \equiv (p \Rightarrow r) \wedge (q \Rightarrow r)$.

b) $p \Rightarrow q \equiv p \vee q \equiv q$.

c) $p \Rightarrow q \equiv \neg p \vee q$.

3. ¿Están bien escritas las siguientes expresiones? Justificá a través de un árbol de tipado. Para evitar errores, hace explícita la precedencia introduciendo paréntesis, y construí una tabla con el tipo de las variables (cuando las expresiones estén bien tipadas).

- a) $(True \equiv a) \vee 5$.
- b) $((True \wedge False) \Rightarrow False) \equiv False$.
- c) $2 = 3 \vee 3 = 4 \vee a * a + 2 \leq b + 7$.
- d) $(x \wedge y \equiv a) \wedge z \leq w$.
- e) $x + 3 \Rightarrow y$.
- f) $(x + 3 = y) \wedge \neg z$.
- g) $a \vee b = 3 + y$.
- h) $x = (y = z)$.
- i) $(x = y) \equiv z$.
- j) $x = (y \equiv z)$.
- k) $a \wedge b \leq a$.

4. (!) ¿Están bien escritas las siguientes expresiones? Justificá a través de un árbol de tipado. Para evitar errores, hace explícita la precedencia introduciendo paréntesis, y construí una tabla del tipo de las variables (cuando las expresiones estén bien tipadas).

- a) $a \geq b \wedge 3 + 2 < 4 \Rightarrow c \equiv b + 1 = 2$.
- b) $a + 2 \geq c \Rightarrow 3 + 2 < b \equiv c \equiv b = 2 * a$.
- c) $\neg a * b + c = d \vee p \Rightarrow q \equiv r \Leftarrow s \wedge j = k + l * m$.
- d) $(a > b \wedge c \Rightarrow d) \triangleright (\#[a, b, c] > 0) \triangleright [True]$

5. (!) Representá como una fórmula de lógica proposicional los siguientes enunciados. Usá el símbolo de predicado p para representar $(a < b)$, usá q para representar $(b < c)$, y r para $(a < c)$.

- a) $a < b < c$
- b) Si $a < b$ entonces no es el caso que $(a \geq c)$.
- c) $(a \geq b$ y $b < c)$ o $(a \geq c)$
- d) No es el caso que $(a < b$ y $a < c)$
- e) (No es el caso que $(a < b$ y $(a < c$ ó $b < c))$) ó $(a \geq b$ y $a < c)$

6. Escribí una fórmula proposicional para cada una de las siguientes frases, utilizando una variable proposicional para cada sentencia atómica, aclarando siempre el significado escogido para cada variable. A veces es conveniente reescribir las frases para hacer mas claro su sentido. Por ejemplo, la frase

Hoy es martes o jueves

puede pensarse como dos oraciones unidas por una disyunción, pudiendo reescribirse como:

Hoy es martes **u** hoy es jueves

Entonces podemos formalizarla con la fórmula $p \vee q$, utilizando p para la primer sentencia y q para la segunda, es decir, definiendo:

$$p \doteq \text{ hoy es martes}$$

$$q \doteq \text{ hoy es jueves}$$

En lo posible utilizá un mismo símbolo de proposición para la misma sentencia atómica a lo largo de todo el ejercicio, para poder comparar las fórmulas.

- a) Hoy es martes y hay sol.
- b) Hoy es martes y no hay sol.
- c) Hoy no es ni miércoles ni viernes.
- d) Mañana, llueve o no llueve.
- e) Mañana será jueves y no será jueves.
- f) Hoy es viernes, pero tengo clases de Algoritmos.
- g) Yo no voy de vacaciones y Juan y Pedro tampoco.
- h) Juan vendrá a clase, y seguro que vendrán también María o Pedro.

- i) Si no tienes menos de 18 años ni tienes antecedentes puedes irte a vivir a otro país.
- j) No es verdad que si tienes menos de 16 años y consentimiento paterno te puedas casar.

7. Evaluá las siguientes expresiones *booleanas*, subrayando la subexpresión resuelta en cada paso, y justificado. Luego usá un intérprete de `haskell` para verificar los resultados. Por ejemplo:

$$\begin{aligned} & \underline{(False \vee True)} \wedge False \equiv False \\ \equiv & \{ \text{def. de } \vee \} \\ & \underline{True} \wedge False \equiv False \\ \equiv & \{ \text{def. de } \wedge \} \\ & \underline{False} \equiv False \\ \equiv & \{ \text{def. de } \equiv \} \\ & \underline{True} \end{aligned}$$

En `haskell` los distintos operadores booleanos se pueden escribir así:

- | | |
|---|--|
| a) $((True \wedge True) \vee False) \equiv False \vee True$ | $\neg p$ <code>not p</code> |
| b) $7 > 4 \wedge 4 > 7$ | $p \wedge q$ <code>p \&\& q</code> |
| c) $7 > 4 \vee 4 > 7$ | $p \vee q$ <code>p q</code> |
| d) $5 > 3 \wedge 3 > 1 \Rightarrow 5 > 1$ | $p \equiv q$ <code>p == q</code> |
| e) $False \Rightarrow 2 + 2 = 5$ | |
| f) $2 + 2 = 5 \Rightarrow True$ | |

Los tres últimos ítems no pueden codificarse directamente en `haskell` porque no disponemos de un operador \Rightarrow en ese lenguaje. ¿Cómo podemos definir una función *implies* : `Bool` \rightarrow `Bool` \rightarrow `Bool` que se comporte como \Rightarrow ?

8. (!) Evaluar las siguientes fórmulas proposicionales, subrayando la subexpresión resuelta en cada paso y justificando; utilizando la siguiente asignación $p \doteq True$, $q \doteq False$ y $r \doteq False$. Luego usá un intérprete de `haskell` para verificar los resultados.

- a) $p \Rightarrow (q \equiv r)$
- b) $\neg p \equiv (\neg q \vee (q \vee r))$
- c) $(q \Rightarrow (p \equiv \neg r)) \Rightarrow ((p \wedge q) \Rightarrow (r \equiv r))$
- d) $\neg(p \vee \neg q) \Rightarrow (\neg(r \equiv (p \wedge q)) \Rightarrow \neg(p \vee \neg q))$

9. (!) Decidí si cada una de las siguientes fórmulas proposicionales son válidas o no. En caso que una fórmula no sea válida, decidí si es satisfactible o no. En todos los casos justificá con una tabla de verdad, un ejemplo o un contraejemplo, según corresponda.

- a) p
- b) $p \equiv p$
- c) $p \equiv p \equiv p$
- d) $p \Rightarrow q$
- e) $p \Rightarrow (q \Rightarrow p)$

10. En `haskell` no existen funciones predefinidas para los operadores \Rightarrow , \neq y \Leftarrow . Definí funciones que se comporten como estos operadores.

11. (!) Redefiní las funciones de los ítems c) y g) del ejercicio 5, h), i), j) y o) del ejercicio 8 de la Guía 2, utilizando operadores booleanos para eliminar todos los análisis por casos. Por ejemplo, en el ítem a) del ejercicio 5 hay que definir una función *entre0y9* : `Int` \rightarrow `Bool`, que dado un entero devuelva *True* si el entero se encuentra entre 0 y 9. Una solución posible al momento de resolver la Guía 2 era:

$$\begin{aligned} \text{entre0y9.n} \doteq & (0 \leq n \wedge n \leq 9 \rightarrow True \\ & \quad \square 0 > n \vee n > 9 \rightarrow False \\ &) \end{aligned}$$

Si observamos con atención nos daremos cuenta que el valor de verdad devuelto en cada caso, se corresponde con el valor de la guarda $0 \leq n \wedge n \leq 9$. Por lo tanto ahora podemos definir la función *entre0y9* directamente como:

$$\text{entre0y9}.n \doteq 0 \leq n \wedge n \leq 9$$

12. (!) Definí funciones por recursión y/o composición para cada una de las siguientes descripciones. Luego evaluá manualmente la función para los valores de cada ejemplo justificando cada paso realizado. Programalas en `haskell` y verificá los resultados obtenidos. Por ejemplo:

Enunciado: *ambospositivos* : $Int \rightarrow Int \rightarrow Bool$, que dados dos enteros devuelve *True* si ambos son positivos.

Por ejemplo *ambospositivos*.2.(-5) = *False*

Solución:

ambospositivos : $Int \rightarrow Int \rightarrow Bool$
ambospositivos.*n.m* $\doteq (n \geq 0) \wedge (m \geq 0)$

`ambospositivos :: Int -> Int -> Bool`
`ambospositivos n m = (n >= 0) && (m >= 0)`

ambospositivos.2.(-5)
 $\equiv \{ \text{def. de ambospositivos} \}$
 $(2 \geq 0) \wedge (-5 \geq 0)$
 $\equiv \{ \text{aritmética} \}$
 $True \wedge (-5 \geq 0)$
 $\equiv \{ \text{Elemento neutro de } \wedge \}$
 $-5 \geq 0$
 $\equiv \{ \text{aritmética} \}$
 $False$

`Main> ambospositivos 2 (-5)`
`False`

- a) *esMultiplo* : $Int \rightarrow Int \rightarrow Bool$ que dado enteros *n* y *m* devuelve *True* si *n* es múltiplo de *m*.
 Por ejemplo: *esMultiplo*.16.4 $\equiv True$
- b) *esVacíaOPrimer0* : $[Int] \rightarrow Bool$ que dada una lista decide si es vacía o bien su primer elemento es 0.
 Por ejemplo: *esVacíaOPrimer0*.[0,1,2] $\equiv True$
- c) *bisiesto* : $Int \rightarrow Bool$ que determina si un año es bisiesto. Recordar que los años bisiestos son aquellos que son divisibles por 4 pero no por 100, a menos que también lo sean por 400.
 Por ejemplo: *bisiesto*.1900 $\equiv False$, y *bisiesto*.2000 $\equiv True$
- d) *ordenada* : $[Int] \rightarrow Bool$ que decide si una lista está ordenada de menor a mayor.
 Por ejemplo: *ordenada*.[1, 2, 3, 2] $\equiv False$
- e) *capicua* : $[A] \rightarrow Bool$, que determina si una lista es palíndromo, esto es, que se lee igual de un lado que del otro.
 Por ejemplo: *capicua*.[1, 2, 3, 2, 1] $\equiv True$
- f) *listasMayorQue* : $Int \rightarrow [[A]] \rightarrow Bool$ que dado un número *n* y una lista de listas, determina si cada una de ellas tiene al menos *n* elementos.
 Por ejemplo: *listasMayorQue*.3.[[1, 2, 3, 4], [0, 1], [5]] $\equiv False$
- g) *listasVacías* : $[[A]] \rightarrow Bool$ que dada una lista de listas, determina si al menos una de ellas es vacía.
 Por ejemplo: *listasVacías*.[[1, 2, 3], [], [5]] $\equiv True$