Introducción a los Algoritmos - 2do. cuatrimestre 2011

Guía 3: Lógica Proposicional

Docentes: Araceli Acosta, Walter Alini, Luciana Benotti, Ezequiel Orbe

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.

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 órden en que asocie los operandos), y según las reglas de precedencia, tal como hemos visto hasta ahora.*

Los operadores $\wedge$, $\vee$, $\equiv$, $\not\equiv$ 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	$=, \leqslant, \geqslant$	*operadores de comparación*
6	$\neg$	*negación*
7	$\vee \wedge$	*disyunción y conjunción*
8	$\Rightarrow \Leftarrow$	*implicación y consecuencia*
9	$\equiv \not\equiv$	*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) $((True \wedge False) \Rightarrow False) \equiv False$.
   b) $2 = 3 \vee 3 = 4 \vee a * a + 2 \leqslant b + 7$.
   c) $(x \wedge y \equiv a) \wedge z \leqslant w$.
   d) $x + 3 \Rightarrow y$.
   e) $(x + 3 = y) \wedge \neg z$.
   f) $a \vee b = 3 + y$.

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) $a \geqslant b \wedge 3 + 2 < 4 \;\Rightarrow\; c \equiv b + 1 = 2.$

   b) $a + 2 \geqslant c \Rightarrow 3 + 2 < b \;\equiv\; c \equiv b = 2 * a.$

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

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

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

$\neg p$	`not p`
$p \wedge q$	`p && q`
$p \vee q$	`p || q`
$p \equiv q$	`p == q`

   a) $((True \wedge True) \vee False) \equiv False \vee True$

   b) $7 > 4 \wedge 4 > 7$

   c) $7 > 4 \vee 4 > 7$

   d) $5 > 3 \wedge 3 > 1 \Rightarrow 5 > 1$

   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 $implies : Bool \to Bool \to Bool$ que se comporte como $\Rightarrow$?

6. 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 intéprete 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) $\neg(p \vee \neg q) \Rightarrow (\neg(r \equiv (p \wedge q)) \Rightarrow \neg(p \vee \neg q))$

7. 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) $p \Rightarrow (q \Rightarrow p)$

8. En `haskell` no existen funciones predefinidas para los operdores $\Rightarrow$, $\not\equiv$ y $\Leftarrow$. Definí funciones que se comporten como estos operadores.

9. 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 $entre0y9 : Int \to Bool$, 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:

$$\begin{aligned} entre0y9.n \doteq (\ & 0 \leqslant n \wedge n \leqslant 9 \to True \\ & \square\ 0 > n \vee n > 9 \to False \\ &) \end{aligned}$$

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

$$entre0y9.n \doteq 0 \leqslant n \wedge n \leqslant 9$$

10. 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* : $Int \to Int \to Bool$, que dados dos enteros devuelve *True* si ambos son positivos.

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

Solución:

$$ambospositivos : Int \to Int \to Bool$$
$$ambospositivos.n.m \doteq (n \geqslant 0) \wedge (m \geqslant 0)$$

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

$$\begin{aligned} & ambospositivos.2.(-5) \\ \equiv & \{ \text{ def. de } ambospositivos \ \} \\ & \underline{(2 \geqslant 0)} \wedge (-5 \geqslant 0) \\ \equiv & \{ \text{ aritmética } \} \\ & True \wedge (-5 \geqslant 0) \\ \equiv & \{ \text{ Elemento neutro de } \wedge \ \} \\ & -5 \geqslant 0 \\ \equiv & \{ \text{ aritmética } \} \\ & False \end{aligned}$$

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

a) *esMultiplo* : $Int \to Int \to Bool$ que dado enteros n y m devuelve *True* si n es múltiplo de m.
Por ejemplo: $esMultiplo.16.4 \equiv True$

b) *esVaciaOPrimer0* : $[Int] \to Bool$ que dada una lista decide si es vacía o bien su primer elemento es 0.
Por ejemplo: $esVaciaOPrimer0.[0,1,2] \equiv True$

c) *bisiesto* : $Int \to Bool$ 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* : $[Int] \to Bool$ que decide si una lista está ordenada de menor a mayor.
Por ejemplo: $ordenada.[1,2,3,2] \equiv False$

e) *capicua* : $[A] \to Bool$, 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* : $Int \to [[A]] \to Bool$ 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) *listasVacias* : $[[A]] \to Bool$ que dada una lista de listas, determina si al menos una de ellas es vacía.
Por ejemplo: $listasVacias.[[1,2,3],[\,],[5]] \equiv True$