Algoritmos y Estructuras de Datos I - 1º cuatrimestre 2011

Práctico 1: Repaso

Docentes: Javier Blanco, Silvia Pelozo, Natalia Bidart, Demetrio Vilela, Walter Alini

El objetivo general de esta guía es retomar la práctica de:

- *Cálculo proposicional*
- *Cálculo de predicados*
- *Cuantificadores ($\forall$ y $\exists$)*
- *Funciones escritas en Haskell*

Se pretende lograr familiaridad con los axiomas y teoremas, sus formas de aplicación, y al mismo tiempo adquirir las habilidades necesarias para resolver las cuentas.

1. Demostrá los siguientes teoremas del cálculo proposicional, con máximo nivel de detalle. Justificá cada paso con el nombre de la regla (axioma y/o teorema) utilizada, y la correspondiente sustitución. Cuando la regla se aplique en una subfórmula, señalála subrayándola. El primer ejercicio corre por cuenta de la casa.

   a) *Asociatividad de $\not\equiv$*: $(p \not\equiv (q \not\equiv r)) \equiv ((p \not\equiv q) \not\equiv r)$

   $$
   \begin{array}{ll}
   & (p \not\equiv (q \not\equiv r)) \\
   \equiv & \{\ (\textbf{A6} \text{ - Definición de } \not\equiv)(P, Q := p, (q \not\equiv r))\ \} \\
   & \neg(p \equiv \underline{(q \not\equiv r)}) \\
   \equiv & \{\ (\textbf{A6} \text{ - Definición de } \not\equiv)(P, Q := q, r)\ \} \\
   & \neg(p \equiv \underline{\neg(q \equiv r)}) \\
   \equiv & \{\ (\textbf{A4} \text{ - Definición de } \neg)(P, Q := q, r)\ \} \\
   & \neg(p \equiv \neg q \equiv r) \\
   \equiv & \{\ (\textbf{A6} \text{ - Definición de } \not\equiv)(P, Q := (p \equiv \neg q), r)\ \} \\
   & \underline{(p \equiv \neg q)} \not\equiv r \\
   \equiv & \{\ (\textbf{A2} \text{ - Conmutatividad de } \equiv)(P, Q := p, \neg q)\ \} \\
   & \underline{(\neg q \equiv p)} \not\equiv r \\
   \equiv & \{\ (\textbf{A4} \text{ - Definición de } \neg)(P, Q := q, p)\ \} \\
   & \underline{\neg(q \equiv p)} \not\equiv r \\
   \equiv & \{\ (\textbf{A2} \text{ - Conmutatividad de } \equiv)(P, Q := p, q)\ \} \\
   & \underline{\neg(p \equiv q)} \not\equiv r \\
   \equiv & \{\ (\textbf{A6} \text{ - Definición de } \not\equiv)(P, Q := p, q)\ \} \\
   & (p \not\equiv q) \not\equiv r
   \end{array}
   $$

   b) *Asociatividad de la conjunción*: $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$

   c) *Idempotencia de la conjunción*: $p \wedge p \equiv p$

   d) *Neutro de la conjunción*: $p \wedge \mathit{true} \equiv p$

   e) *Absorción*: $p \wedge (p \vee q) \equiv p$

   f) *De Morgan para $\wedge$*: $\neg(p \wedge q) \equiv \neg p \vee \neg q$

2. Demostrá los siguientes teoremas del cálculo proposicional:

   a) *Debilitamiento para $\wedge$*: $p \wedge q \Rightarrow p$

   b) *Relación $\Rightarrow$ respecto a $\Leftarrow$*: $p \Rightarrow q \equiv q \Leftarrow p$

   c) *Intercambio para $\Rightarrow$*: $p \Rightarrow (q \Rightarrow r) \equiv p \wedge q \Rightarrow r$

   d) *Implicación distribuye a izquierda con $\equiv$*: $p \Rightarrow (q \equiv r) \equiv p \Rightarrow q \equiv p \Rightarrow r$

   e) *Doble implicación*: $(p \Rightarrow q) \wedge (q \Rightarrow p) \equiv p \equiv q$

   f) *Contrarrecíproca*: $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$

   g) *Modus ponens*: $p \wedge (p \Rightarrow q) \Rightarrow q$

   h) *Modus tollens*: $(p \Rightarrow q) \wedge \neg q \Rightarrow \neg p$

i) *Transitividad*: $(p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$

j) *Monotonía conjunción*: $(p \Rightarrow q) \Rightarrow (p \wedge r \Rightarrow q \wedge r)$

k) *Monotonía disjunción*: $(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$

3. Considerá las siguientes funciones:

   - `sumarALista :: Num a => a -> [a] -> [a]` que toma un número y una lista de números y le suma a cada elemento de la lista el primer parámetro. Por ejemplo:

     ```
     sumarALista 3 [4,6,7] = [7,9,10]
     ```

   - `encabezar :: a -> [[a]] -> [[a]]` que toma una expresión de tipo `a` y lo pone en la cabeza de cada lista del segundo parámetro. Por ejemplo:

     ```
     encabezar 3 [[2,1],[],[4,7]] = [[3,2,1],[3],[3,4,7]]
     ```

   a) Escribí, primero en papel y después en Haskell, `sumarALista` y `encabezar` usando caso base y caso inductivo.

   b) Escribí ambas funciones en Haskell utilizando el funcional *map*. No se puede usar más de una línea, para cada una, ni puede haber definiciones locales.

4. A partir de las siguientes especificaciones:

   - Dá el tipo de la función.
   - Expresá en lenguaje natural qué calcula la función.
   - Escribí en Haskell cada una de estas funciones, con su tipo y su definición. Es una buena oportunidad para escribirlas intuitivamente y probar su funcionamiento en la computadora. En prácticos siguientes vamos a formalizar un poco este mecanismo.

   a) $sum\_pares.n = \langle \sum i : 0 \leq i \leq n \wedge par.i : i \rangle$

   b) $sum\_cuads.xs = \langle \sum i : 0 \leq i < \#xs : xs.i * xs.i \rangle$

   c) $es\_cuadrado.n = \langle \exists x : 0 \leq x \leq n : x * x = n \rangle$

5. Escribí en Haskell el tipo y la definición de las siguientes funciones. En prácticos siguientes, también vamos a derivar estas funciones. El tipo de las funciones debe ser el más general posible.

   Ayuda: Para definir los tipos de las funciones, será necesarios ver a qué *clase* de tipos deben pertenecer. Ejemplo: para el caso de *minimo*, investigar la clase de tipos *Bounded*.

   a) *minimo*, que calcula el mínimo elemento de una lista.

   b) *iguales*, que determina si los elementos de una lista son todos iguales entre sí.

   c) *creciente*, que determina si los elementos de una lista están ordenados en forma creciente.

   d) *sum_ant*, que determina si algún elemento de una lista es igual a la suma de todos los elementos anteriores dentro de la misma.

   e) *pos_min*, que determina si el n-ésimo elemento de una lista contiene al mínimo valor de la misma.

6. Expresá en lenguaje natural cada una de las siguientes sentencias formales, donde xs y ys son listas de enteros:

   a) $\langle \forall x : x \in Num : \langle \exists y : y \in Num : x < y \rangle \rangle$

   b) $\langle \exists x : x \in Num : \langle \forall y : y \in Num : x < y \rangle \rangle$ ¿Cuál es la diferencia con la anterior?

   c) $\langle \forall x, z : x \in Num \wedge z \in Num \wedge x \neq z : \langle \exists y : y \in Num : x < y < z \rangle \rangle$

   d) $\langle \exists i : 0 \leq i < \#xs : xs.i = 0 \rangle$

   e) $\langle \forall i : 0 \leq i < \#xs : xs.i \geq 0 \rangle$

f) $\langle \exists i : 0 < i < \#xs : xs.(i-1) < xs.i \rangle$

g) $\langle \exists i : 0 \leq i < \#xs \min \#ys : xs.i \neq ys.i \rangle$

7. Intentaremos escribir en Haskell algunas expresiones que contengan cuantificadores. Para ello:

   a) (Para el curioso) Investigar qué es el valor *undefined* en Haskell, y para qué puede usarse.

   b) Investigar si existe el operador de implicancia en Haskell (Mirar concentradamente por un rato el Operador <=, pero sin tenerle demasiada confianza. Quien haya investigado el valor *undefined* sospechará dónde desconfiar primero).

   c) En caso de no existir, definir un operador para la implicación, que se comporte como es esperado, incluso para el valor *undefined*. Puntos extras si el operador es infijo.

   d) Investigar las siguientes funciones del *Prelude* que trabajan sobre listas: *any*, *all*.

   e) Con toda la información recabada escribir en Haskell las siguientes sentencias formales, y probarlas con ejemplos concretos:

      1) $\langle \exists i : 0 \leq i < \#xs : xs.i = 0 \rangle$
      2) $\langle \forall i : 0 \leq i < \#xs : xs.i \geq 0 \rangle$
      3) $\langle \forall i : 0 \leq i < \#xs : xs.i \, mod \, 2 = 0 \implies xs.i \, mod \, 6 = 0 \rangle$
      4) $\langle \exists i : 0 \leq i < \#xs : empieza\_con\_vocal \, xs.i \implies es\_capicua \, xs.i \rangle$