Algoritmos y Estructuras de Datos I - 2007
Práctico 3: Funciones Recursivas

Docentes: Silvina Smith, Renato Cherini, Valeria Rulloni,
Alejandro Peralta Frías, Mariana Bodano

18 de octubre de 2007

1. *Torres de Hanoi*. Se tienen tres postes -0, 1 y 2- y n discos de distinto tamaño. En la situación inicial se encuentran todos los discos ubicados en el poste 0 en forma decreciente según el tamaño, con el más grande en la base.

   El problema consiste en llevar todos los discos al poste 2, con las siguientes restricciones:

   (i) Se puede mover sólo un disco por vez (el que está más arriba en algún poste).

   (ii) No se puede apoyar un disco sobre otro de menor tamaño.

   Sea $B = \{0, 1, 2\}$. Definir una función $f : B \to B \to B \to Nat \to [(B, B)]$ tal que $f.a.b.c.n$ calcule la secuencia de movimientos para llevar n discos del poste a al poste c, pasando si es necesario por el poste b.

   Ejemplo: $f,0,1,2,2 = [(0, 1),\ (0, 2),\ (1, 2)]$

2. Sea $m : [Int] \to Int$ la función que devuelve el mínimo elemento de una lista de enteros. Especificar y derivar una definición recursiva para m.

3. Derivar la función que eleva un número natural al cubo, usando sólo sumas. La especificación es la obvia: $f.x = x^3$.

   Sugerencia: usar inducción y modularización varias veces.

4. Calcular la función de Fibolucci, $Fbl : Nat \to Nat$, especificada como sigue:

$$Fbl.n = (\sum i : 0 \leq i < n : fib.i * fib.(n - i))$$

   donde fib es la función de Fibonacci.

5. Sea f la función que resuelve el problema de las *Torres de Hanoi*. Calcular una definición recursiva para la función $t : B \to B \to B \to Nat \to ([(B, B)],\ [(B, B)],\ [(B, B)])$, tal que:

$$t.a.b.c.n = (f.a.b.c.n,\ f.b.c.a.n,\ f.c.b.a.n)$$

   Sugerencia: calcular $t.a.b.c,0$, $t.a.b.c,1$ y $t.a.b.c.(n + 2)$.

6. Especificar las funciones *abr* y *cie*, que cuentan –respectivamente– la cantidad de paréntesis que abren '(' o cierran ')' existentes en una lista xs, y demostrar que satisfacen las siguientes propiedades:

$$\begin{array}{rcl} abr.[\,] & = & 0 \\ abr.(\text{`(`} \triangleright xs) & = & 1 + abr.xs \\ abr.(\text{`)`} \triangleright xs) & = & abr.xs \end{array} \qquad \begin{array}{rcl} cie.[\,] & = & 0 \\ cie.(\text{`(`} \triangleright xs) & = & cie.xs \\ cie.(\text{`)`} \triangleright xs) & = & 1 + cie.xs \end{array}$$

7. Derivar una definición recursiva para la función $iguales : [A] \to Bool$, que determina si los elementos de una lista dada son todos iguales entre sí.

8. Derivar una definición recursiva para la función $creciente : [Int] \to Bool$, que determina si los elementos de una lista de enteros están ordenados en forma creciente.

9. Derivar una definición recursiva para la función $f : Int \to [Num] \to Bool$, que determina si el k-ésimo elemento de una lista de números aloja el mínimo valor de la misma.

10. Derivar una definición recursiva para la función $f : [Num] \to Bool$, la cual, dada una lista de naturales, determina si algún elemento de la lista es igual a la suma de todos los otros elementos de la misma.

11. Derivar una definición recursiva para la función $f : [Num] \to [Num] \to Num$, especificada como sigue:

$$f.xs.ys = (Min\ i, j : 0 \leq i < \#xs \wedge 0 \leq j < \#ys : |xs.i - ys.j|).$$

12. Derivar una definición recursiva para la función $f : [Char] \to [Char] \to Bool$, especificada como sigue:

$$f.xs.ys = (\exists\ as, bs, c, cs : xs = as +\!\!+ bs \wedge ys = as +\!\!+ (c \triangleright cs) : bs = [\,] \vee bs{,}0 < c)$$

Decir en palabras que significa que $f.xs.ys$ sea **true**.