Algoritmos y estructuras de datos I - 2009

Práctico 2: Funciones recursivas

Docentes: Mariana Badano, Silvina Smith, Valeria Rulloni.

1. Probar que las funciones *abr* y *cie*, la cuales cuentan –respectivamente– la cantidad de paréntesis que abren '(' o cierran ')' hay en una lista xs, satisfacen las siguientes propiedades:

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

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

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

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

5. Derivar una definición recursiva para la función $P : [Num] \mapsto Bool$, 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.

6. Calcular la función que eleva un nómero natural al cubo, usando sólo sumas. La especificación es la obvia: $f.x = x^3$. Sugerencia: usar inducción, modularización varias veces y luego técnica de tuplas para mejorar la eficiencia.

7. *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 (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 $f : B \mapsto B \mapsto B \mapsto Nat \mapsto [(B, B)]$ tal que $f.a.b.c.n$ calcule la secuencia de movimientos para llevar n discos del poste a al poste c, pasando por el poste b.

   Ejemplo: $f.0.1.2.2 = [(0, 1),\ (0, 2),\ (1, 2)]$

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

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

   Sugerencia: calcular $t.a.b.c.0$, $t.a.b.c.1$ (este "caso base" no es necesario pero contribuye a la comprensión) y $t.a.b.c.(n + 1)$.

9. Derivar una definición recursiva para la función $f : Int \mapsto [Num] \mapsto Bool$, 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 $f : [Num] \mapsto [Num] \mapsto Num$, especificada como sigue:

$$f.xs.ys =< \mathrm{Min}\, i, j : 0 \leq i < \#xs \wedge 0 \leq j < \#ys : |xs.i - ys.j| > .$$

11. Derivar una definición recursiva para la función $l : [Char] \mapsto [Char] \mapsto Bool$, especificada como sigue:

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

Decir en palabras cuándo $l.xs.ys$ es $true$.

12. Sea fib es la función de Fibonacci. Calcular la función de Fibolucci, $Fbl : Nat \mapsto Nat$, especificada por:

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