Facultad de Informática

Estructuras de Datos y de la Información

Grupo A, Diciembre 2002

Ejercicios sobre algoritmos recursivos

Ejercicio 1 (La potencia entera reconsiderada) Aplicando la misma idea vista en clase para la división entera con coste logarítmico, se puede llegar al siguiente programa para calcular la potencia n-ésima de un número:

fun potencia(a : *entero*; n : *natural*) **dev** (p : *entero*) =
  caso $n = 0 \rightarrow 1$
    □ $n \geq 0 \rightarrow$ sea $p' = potencia(a, n \text{ div } 2)$ en
      caso $par(n) \rightarrow p' * p'$
        □ $\neg par(n) \rightarrow a * p' * p'$
      fcaso
  fcaso
ffun

Llevar a cabo la verificación formal y el estudio del coste de este programa.

Ejercicio 2 Llevar a cabo la verificación formal y el estudio del coste de la versión recursiva no final del producto escalar *iprodesc* visto en clase:

fun iprodesc(a, b : *vect*; j, n : *entero*) **dev** (p : *entero*) =
  caso $j = n + 1 \rightarrow 0$
    □ $j < n + 1 \rightarrow a[j] * b[j] + iprodesc(a, b, j + 1, n)$
  fcaso
ffun

Ejercicio 3 Llevar a cabo la verificación formal y el estudio del coste de la versión recursiva no final de la raiz cuadrada entera $iraiz_1$ vista en clase:

fun $iraiz_1(n, a$: *entero*) **dev** (r : *entero*) =
  caso $a^2 > n \rightarrow 0$
    □ $a^2 \leq n \rightarrow$ sea $r' = iraiz_1(n, 2 * a)$ en
      caso $n < (r' + a)^2 \rightarrow r'$
        □ $n \geq (r' + a)^2 \rightarrow r' + a$
      fcaso
  fcaso
ffun

Ejercicio 4 Llevar a cabo la verificación formal y el estudio del coste de la versión recursiva final de la raiz cuadrada entera $iraiz_2$ vista en clase:

fun $iraiz_2(n, a$: *entero*) **dev** (r : *entero*) =
  caso $(a + 1)^2 > n \rightarrow a$
    □ $(a + 1)^2 \leq n \rightarrow iraiz_2(n, a + 1)$

fcaso
ffun

Ejercicio 5 Definimos la sucesión de números naturales

$$\begin{aligned} \mathsf{fob}_0(x,y) &= x \\ \mathsf{fob}_1(x,y) &= y \\ \mathsf{fob}_n(x,y) &= \mathsf{fob}_{n-2}(x,y) + \mathsf{fob}_{n-1}(x,y),\ n \geq 2. \end{aligned}$$

Demostrar la corrección de la siguiente función recursiva:

$\{\mathsf{cierto}\}$
fun $\mathsf{gfob}(n, x, y : \mathit{nat})$ **dev** $r, s : \mathit{nat}$
  caso $n = 0 \rightarrow \langle x, y \rangle$
    $\square$ $n > 0 \rightarrow \mathsf{gfob}(n-1, y, x+y)$
  fcaso
ffun
$\{r = \mathsf{fob}_n(x,y), s = \mathsf{fob}_{n+1}(x,y)\}$

Ejercicio 6 Diseñar una función recursiva para calcular multiplicaciones según el siguiente método conocido como *del campesino egipcio*:

$$\mathsf{mult}(x,y) = \begin{cases} 0 & y = 0 \\ x & y = 1 \\ \mathsf{mult}(2 * x, y \ \mathbf{div}\ 2) & y \geq 2,\ y \ \mathbf{mod}\ 2 = 0 \\ \mathsf{mult}(2 * x, y \ \mathbf{div}\ 2) + x & y \geq 2,\ y \ \mathbf{mod}\ 2 = 1 \end{cases}$$

Demostrar su corrección y calcular su coste.

Ejercicio 7 Realizar una inmersión final de $raiz$, escogiendo en $R(n,a) \equiv a^2 < n \wedge n < (a+1)^2$ la segunda conjunción como precondición y la primera como condición B_t. Realizar el diseño, verificación y estudio del coste.

Ejercicio 8 Verificar formalmente y estudiar el coste de la versión final de la raiz cuadrada $iraiz_3$ vista en clase;

fun $iraiz_3(n, a, b : \mathit{entero})$ **dev** $(r : \mathit{entero}) =$
  caso $b = a + 1 \rightarrow a$
    $\square$ $b \neq a + 1 \rightarrow$ **sea** $m = (a + b)$ **div** 2 **en**
      caso $m^2 \leq n \rightarrow iraiz_3(n, m, b)$
        $\square$ $m^2 > n \rightarrow iraiz_3(n, a, m)$
      fcaso
  fcaso
ffun

Ejercicio 9 Realizar las siguientes inmersiones para el producto escalar *prodesc* visto en clase. Derivar la precondición y postcondición de la función inmersora y llevar a cabo el diseño y verificación de la misma:

- Una no final, utilizando la sustitución $j = n$.

- Una final, expresando $R(a, b, n, s)$ como $s = \sum_{\xi=1}^{n+1-1} a[\xi] \cdot b[\xi]$ y sustituyendo la expresión $n + 1$ de $R(a, b, n, s)$ por la variable i.
- Una final, sustituyendo esta vez la expresión 1 de $R(a, b, n, s)$ por la variable i.

Ejercicio 10 Consideremos la siguiente especificación de un algoritmo para sumar todos los elementos de un vector dado:

$\{0 \leq n \leq 1000\}$
fun suma-vector(v : *vect*; n : *entero*) **dev** s : *entero*
$\{s = (\sum_{i=1}^{n} v[i])\}$.

1. Realizar una *inmersión no final* sustituyendo la constante 1 por una nueva variable j.
2. Realizar una *inmersión final* sustituyendo la constante 1 por una nueva variable j.
3. Realizar otra *inmersión final*, escribiendo la postcondición de la forma $s = (\sum_{i=1}^{(n+1-1)} v[i])$ y sustituyendo la expresión $n + 1$ por una nueva variable j.
4. Derivar un algoritmo recursivo *múltiple* de coste lineal (con respecto a la longitud del vector) sustituyendo las constantes 1 y n por variables nuevas c y f, respectivamente.

Ejercicio 11 Aplicar la técnica de desplegado y plegado al siguiente programa, que calcula el cociente y el resto de la división entera:

fun divide(a, b : *entero*) **dev** $(q, r$: *entero*$) =$
  caso $a < b \rightarrow < 0, a >$
    $\square$ $a \geq b \rightarrow$ sea $< q', r' >= divide(a - b, b)$ en $< q' + 1, r' >$
  fcaso
ffun

Ejercicio 12 Realizar un seguimiento manual de las llamadas recursivas desencadenadas por $pot(a, 13)$ y por $iipot(a, 13, 1, 1, a)$ observando las diferencias en la forma de calcular el resultado, donde *pot* es el algoritmo recursivo lineal no final que calcula una potencia con coste logarítmico (en términos del número de multiplicaciones) y *iipot* es la correspondiente inmersión final obtenida por desplegado y plegado. Comparar las llamadas desencadenadas en el segundo caso con las transformaciones realizadas en la ejecución de la versión iterativa.

Ejercicio 13 Aplicar la técnica de desplegado y plegado al programa *iprodesc* visto en clase (y mostrado en el ejercicio 2) para calcular el producto escalar desde j hasta n.

Ejercicio 14 Realizar una inmersión, desplegado y plegado para obtener una versión recursiva lineal final del algoritmo que calcula el factorial de un número natural n. Realizar un seguimiento manual de las llamadas recursivas desencadenadas en ambos casos sobre el número 6. Pasar a iterativa la versión recursiva lineal final.

Ejercicio 15 1. Pasar a iterativo el algoritmo recursivo lineal final de búsqueda binaria.

2. Pasar a iterativos el algoritmo recursivos lineal no final de la división entera del ejercicio 11.

Ejercicio 16 El n-ésimo número de Fibonacci se define según la siguiente definición recursiva:

$$\mathsf{fib}(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ \mathsf{fib}(n-1) + \mathsf{fib}(n-2) & n \geq 2. \end{cases}$$

Demostrar que el coste del algoritmo recursivo múltiple que resulta de esta definición está en $\Omega(2^{n/2}) \cap O(2^n)$ (suponiendo que las operaciones aritméticas son básicas), es decir, que este algoritmo tiene coste exponencial. Realizar una inmersión por eficiencia que dé lugar a un algoritmo recursivo lineal de coste también lineal.

Ejercicio 17 Dados $1 < b < 10$ y $a \geq 0$, diseñar una función recursiva que devuelva un número natural x tal que su expresión decimal coincida con la representación de a en base b; por ejemplo, para $b = 3$ y $a = 14$, la función ha de devolver $x = 112$.

Ejercicio 18 Una entidad bancaria proporciona un número de identificación personal (un entero positivo) a cada uno de sus usuarios. El usuario de una tarjeta debe teclear su número de identificación para poder utilizarla. Los terminales de la entidad traducen dicho número a una forma cifrada (también un entero positivo). Si ésta coincide con la forma cifrada grabada en la tarjeta, el usuario podrá utilizarla para llevar a cabo las operaciones deseadas.

Un número de identificación cuya representación decimal sea $d_1\ d_2 \ldots d_n$ (donde n es el número de cifras y cada cifra d_i es un valor numérico comprendido entre 0 y 9; por ejemplo para 2145, $n = 4$ y $d_1 = 2, d_2 = 1, d_3 = 4, d_4 = 5$) se transforma en su correspondiente clave de la siguiente manera:

- Si $n = 1$ la clave es $d_1 * 7$.
- Si $n > 1$ la clave es $(d_n\ mod\ 3) + d_n * c$ siendo c la clave de $d_1\ d_2 \ldots d_{n-1}$.

Se pide:

(a) Escribe una función recursiva que transforme un número de identificación dado en su clave. (No hace falta especificarla.)

(b) Aplica la técnica de plegado-desplegado para obtener una versión recursiva final de la función del apartado anterior. Detalla cuidadosamente cada uno de los pasos.

(c) Transforma la función obtenida en el apartado anterior a una versión iterativa (no hace falta proporcionar el invariante).

Ejercicio 19 A continuación se muestra la especificación de una función que, dado un vector a y un entero n, calcula su máximo:

$$\{1 \leq n \leq 1000\}$$
$$\textbf{fun}\ maximo(a : vect; n : entero)\ \ \textbf{dev}\ \ (x : entero)$$
$$\{(\forall \alpha \in \{1..n\}.x \geq a[\alpha]) \wedge (\exists \beta \in \{1..n\}.x = a[\beta])\}$$

Supongamos que disponemos de una función *max* que dados dos enteros cualesquiera, devuelve el máximo entre ellos.

(a) Llevar a cabo una inmersión no final de *maximo* para generar una versión recursiva no final *imaximo* de la misma.

(b) Verificar formalmente la función del apartado anterior.

(c) Llevar a cabo una inmersión final de la función *maximo* para generar una versión recursiva final *iimaximo* de la misma (Indicación: además de añadir el parámetro acumulador, debilitar sustituyendo una constante por una nueva variable j).

Ejercicio 20 Se desea definir una función que calcule, en forma codificada, el número de veces que aparecen las cifras 0 y 1 en un número binario x dado. Si m es el número de veces que aparece la cifra 0 en x y n es el número de veces que aparece la cifra 1 en x, entonces la función devolverá $2^m * 3^n$.

(a) Escribe una función recursiva no final *CeroUno*, que tome un número natural x formado exclusivamente por ceros y unos, y devuelva la codificación anteriormente descrita.

Por ejemplo, si $x = 10100$, entonces $m = 3$, $n = 2$ y la función devuelve $2^3 * 3^2 = 72$.

(b) Aplica la técnica de plegado-desplegado para obtener una versión recursiva final de la función del apartado anterior. Detalla cuidadosamente cada uno de los pasos.

Ejercicio 21 En todos los apartados de este ejercicio vamos a suponer que un polinomio con coeficientes enteros $p = \sum_{i=0}^{n} c_i x^i$ se representa como un vector $p[0..n]$ de forma que $p[i] = c_i$.

(a) Especificar una función que dado un polinomio p en forma de vector y un entero v, calcule el valor $p(v)$ del polinomio en ese entero.

(b) Derivar un algoritmo recursivo para la especificación del apartado anterior realizando una inmersión no final que sustituya la constante 0 por una nueva variable k, teniendo en cuenta que $v^i = v^{i-0}$, de forma que ese cero también ha de ser generalizado.

(c) Calcular el número exacto de multiplicaciones que realiza el algoritmo del apartado anterior, resolviendo la recurrencia apropiada.

(d) Pasar a iterativo el algoritmo recursivo lineal no final del apartado (b).

(e) Aplicar la técnica de plegado-desplegado para obtener una versión recursiva lineal final del algoritmo del apartado (b).

(f) Calcular el número exacto de multiplicaciones que realiza el algoritmo del apartado (e), resolviendo la recurrencia apropiada.

(g) Pasar a iterativo el algoritmo recursivo lineal final del apartado (e).