TRANSFORMACIÓN DE ALGORITMOS RECURSIVOS

Se trata de transformar la siguiente función recursiva no final a sus versiones recursiva final e iterativa.

```
entero función fun (E Vect: y, E Vect: z, E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
   si i=n entonces
      devolver y[i]*z[i]
   si_no
      devolver  y[i]*z[i] + 5*fun(y, z, n, i+1)
   fin_si
   {devuelve ∑_{α=i}^{n} (y[α]* z[α])*5^{α-i} }
fin_función
```

1. Función RECURSIVA NO FINAL a Función RECURSIVA FINAL

Esquema general de una Función Recursiva No Final

```
tipo función f_rec( X̄ )
inicio
    si caso_base?( X̄ ) entonces
        devolver sol( X̄ )
    si_no
        devolver comb(f_rec(suc( X̄ )), X̄ )
fin_si
fin_función
```

- El parámetro formal $\bar{X}$ debe entenderse como una tupla de parámetros $x_1, x_2, \ldots, x_n$.
- **caso_base?($\bar{X}$):** es una expresión lógica que determina si $\bar{X}$ cumple la condición para acabar el proceso recursivo.
- **sol($\bar{X}$):** es una función o expresión para calcular la solución de la función recursiva cuando se cumple la condición del caso base.
- **suc($\bar{X}$):** es una función o expresión para determinar el sucesor de cada parámetro de la tupla $\bar{X}$ **.**
- **comb:** función o expresión que combina el valor devuelto por la función recursiva f_rec con todos o algunos parámetros del subalgoritmo.

Función Recursiva No Final del Ejercicio

```
entero función fun (E Vect: y, E Vect: z,
                    E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
   si i=n entonces
      devolver y[i]*z[i]
   si_no
      devolver  y[i]*z[i] + 5*fun(y,z,n,i+1)
   fin_si
{devuelve  Σ_{α=i}^{n} (y[α] * z[α]) * 5^{α−i} }
fin_función
```

- *tupla de parámetros formales* $\bar{X}$ **≡ y, z, n, i**
- **caso_base?(y, z, n, i) ≡ i=n**
- **sol(y, z, n, i) ≡ y[i]*z[i]**
- **suc(y, z, n, i) ≡ y, z, n, i+1**
- **comb(y, z, n, i, fun) ≡ y[i]*z[i] + 5*fun**

a) Generalización:

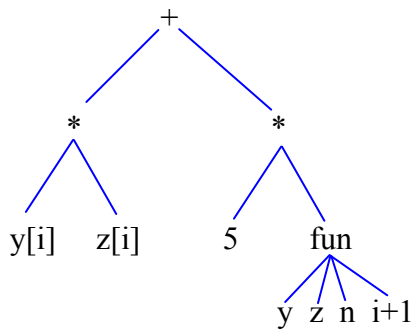
- La inmersión f_recFinal se obtiene considerando la expresión del caso general de la función a transformar:

  f_rec($\overline{x}$) = comb(f_rec(suc($\overline{x}$)), $\overline{x}$)

Función sumergida: **f_rec(y,z,n,i)= y[i]*z[i] + 5*fun(y, z, n, i+1)**

- Para encontrar la función inmersora es necesario añadir los parámetros de inmersión necesarios. Para ello es posible representar mediante un árbol sintáctico la expresión del caso general :

Árbol Sintáctico (función sumergida)



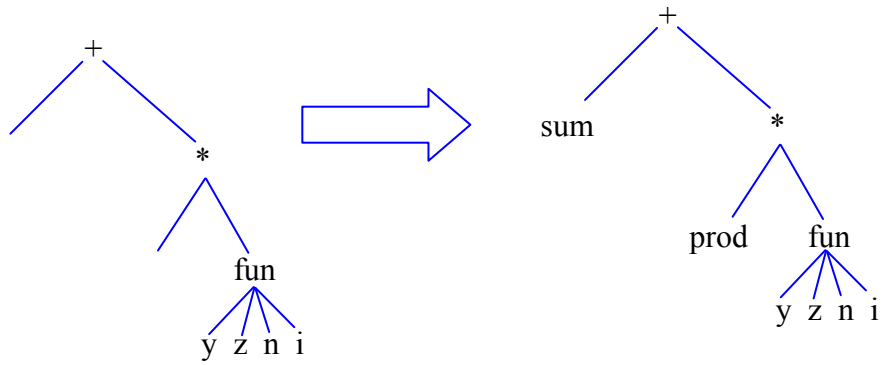
a) Se conserva el camino que va desde la raíz hasta la invocación a fun.

*b) Cada **subárbol** lateral a este camino se sustituye por un parámetro de inmersión.*

c) En la invocación a fun se sustituye cada parámetro sucesor por el original.

d) En el resto de las ramas: Sustitución por parámetros de inmersión

Árbol Sintáctico con los parámetros de inmersión



- Añadir los parámetros de inmersión como argumentos de la Función Final

Función inmersora: **fun_final(y,z,n,i,sum,prod)=sum+prod*fun(y,z,n,i)**

Si la función *comb* tiene elemento neutro w_0:

- **Elemento neutro de la suma: 0**
- **Elemento neutro del producto: 1**

Se obtienen los valores con los que realizar la llamada inicial a la función recursiva final: f_recFinal($\overline{x}$, w_0)

fun_final(y,z,n,i,0,1) =0+1*fun(y,z,n,i)

Por tanto fun_final es una generalización de fun, que se comporta de forma similar para los valores iniciales de los parámetros de inmersión.

Y la función que realiza la llamada inicial a la función recursiva final:

```
entero función llamada (E Vect: y, E Vect: z, E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
        devolver fun_final(y, z, n, i, 0, 1)
fin_función
```

b) Desplegado

1. Incorporación de los parámetros de inmersión siguiendo el mismo análisis de casos que en la función recursiva no final

```
entero función fun_final (E Vect: y, E Vect: z, E entero: n, E entero: i, E entero:
sum, E entero: prod)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
   si i=n entonces
      devolver   sum + prod * (y[i]*z[i])
   si_no
      devolver  sum + prod* (y[i]*z[i] + 5*fun(y, z, n, i+1) )
   fin_si
{devuelve \sum_{α=i}^{n} (y[α] * z[α]) * 5^{α-i} }
fin_función
```

2. Reorganización del caso general:

Aplicación de la propiedad asociativa del producto y reorganización de los términos obtenidos, encaminando el caso general para encontrar los sucesores de los parámetros de inmersión que deben ir en la llamada Recursiva Final:

(sum + prod *(y[i]*z[i])) + **(prod* 5)***fun(y, z, n, i+1)

La función Recursiva Final sería la siguiente (aún dependiente de la función recursiva no final):

entero **función fun_final** (E Vect: y, E Vect: z, E entero: n, E entero: i, **E entero: sum, E entero: prod**)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
  si i=n **entonces**
    devolver **sum + prod * (y[i]*z[i])**
  si_no
    devolver **(sum + prod *(y[i]*z[i]))** + **(prod* 5)***fun(y, z, n, i+1)
  fin_si
{devuelve $\sum_{\alpha=i}^{n}(y[\alpha]*z[\alpha])*5^{\alpha-i}$ }
fin_función

c) Plegado

Los sucesores de los parámetros de inmersión utilizados en la invocación a la llamada recursiva son:

suc(sum)= **sum + prod *y[i]*z[i]**
suc(prod)= **prod* 5**

Por tanto, la Versión Recursiva Final sería:

```
entero función fun_final (E Vect: y, E Vect: z, E entero: n, E entero: i, E entero: sum,
                          E entero: prod)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
   si i=n entonces
      devolver  sum + prod * (y[i]*z[i])
   si_no
      devolver  fun_final(y, z, n, i+1, sum + prod *y[i]*z[i],  prod* 5 )
   fin_si
{devuelve ∑_{α=i}^{n} (y[α] * z[α]) * 5^{α−i} }
fin_función
```

2. Función RECURSIVA NO FINAL a ITERATIVA

Realiza la transformación de la función Recursiva No Final a Iterativa siguiendo el esquema siguiente y posteriormente realiza la optimización del código.

```
tipo función f_iter( x̄ )
var  res, c
inicio
    c←0
    mientras  ¬ caso_base? ( x̄ ) hacer
          c ← c + 1
          x̄  ← suc ( x̄ )
    fin_mientras
     res ← sol(x)
    mientras c ≠0 hacer
          c ← c – 1
          x ← suc^-1 ( x̄ )
          res ← comb (res,  x̄ )
    fin_mientras
    devolver res
fin_función
```

- La variable local *c* sirve para contar el número de llamadas recursivas que se producen en la versión recursiva no final.
- La variable local *res* será la encargada de acumular los resultados.
- La función iterativa empezará a realizar los cálculos desde el caso base, por tanto hay que llegar con cada parámetro al caso base (primer bucle mientras) y después reconstruir el proceso recursivo mediante un proceso iterativo (segundo bucle mientras).

```
entero función fun_iterativa (E Vect: y, E Vect: z, E entero: n, E entero: i)
 {y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
 var
    entero: res,  c
 inicio
    c←  0
    mientras  ¬ (i=n) hacer
        c ← c + 1
        i ← i+1
    fin_mientras
    res ←  y[i]*z[i]
    mientras c ≠0 hacer
        c ← c – 1
        i ←  i – 1
        res ←  y[i]*z[i] + 5*res
    fin_mientras
    devolver res
{devuelve Σ_{α=i}^{n} (y[α] * z[α]) * 5^(α−i) }
fin_función
```

OPTIMIZACIÓN DE LA FUNCIÓN ITERATIVA

- Para eliminar el primer bucle mientras:
  - El número de veces que se realiza la llamada recursiva será n-i veces.
  - El parámetro i, el último valor que toma en el caso base es n, es posible realizar directamente esta inicialización.

entero **función** **fun_iterativa** (E Vect: y, E Vect: z, E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
var
  entero: res, c
inicio
  c ← n – i
  i ← n
  res ← y[i]*z[i]
  mientras c ≠0 **hacer**
    c ← c – 1
    i ← i – 1
    res ← y[i]*z[i] + 5*res
  fin_mientras
  devolver *res*
{devuelve $\sum_{\alpha=i}^{n}(y[\alpha]*z[\alpha])*5^{\alpha-i}$ }
fin_función

3. Función RECURSIVA FINAL a ITERATIVA

```
tipo función f_iter( x̄ )
inicio
        mientras ¬ caso_base? ( x̄ ) hacer
                x̄ ←suc ( x̄ )
        fin_mientras
        devolver sol( x̄ )
fin_función
```

- El bucle mientras se ejecuta tantas veces como llamadas recursivas se producen en la versión recursiva final, calculando los sucesores de los parámetros, obteniendo así el resultado a devolver.

Realiza la transformación de la función recursiva final a Iterativa siguiendo el esquema anterior y posteriormente realiza la optimización del código.

Al ser una transformación de la recursiva final, también va acompañada de la función que hace la primera llamada para la inicialización de los parámetros de inmersión.

```
entero función fun_iterativa (E Vect: y, E Vect: z, E entero: n, E entero: i,
                                E entero: sum, E entero: prod)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n}
inicio
   mientras ¬ (i=n) hacer
     sum ← sum + prod *y[i]*z[i]
     prod ← prod* 5
     i ←  i+1
   fin_mientras
   devolver sum + prod * (y[i]*z[i])
{devuelve Σ_{α=i}^{n} (y[α] * z[α]) * 5^{α−i} }
fin_función


entero función llamada_iterativa (E Vect: y, E Vect: z, E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
inicio
        devolver fun_iterativa(y, z, n, i, 0, 1)
fin_función
```

OPTIMIZACIÓN DE LA FUNCIÓN ITERATIVA

- Los parámetros de inmersión pueden ser eliminados como parámetros formales y ser declarados como variables locales en la sección ***var*** e inicializarlos al comienzo del cuerpo de la función.
- La función que realiza la llamada inicial a la función iterativa inicializando los parámetros de inmersión ya no es necesaria y puede ser, por tanto, eliminada.

```
entero función fun_iterativa (E Vect: y, E Vect: z, E entero: n, E entero: i)
{y = A[1..n] ∧ z = B[1..n] ∧ 1≤ i≤n }
var
    entero: sum, prod
inicio
    sum ←0
    prod ←1
    mientras ¬ (i=n) hacer
      sum ← sum + prod *y[i]*z[i]
      prod ← prod* 5
      i ←  i+1
   fin_mientras
   devolver  sum + prod * (y[i]*z[i])
{devuelve Σ_{α=i}^{n} (y[α] * z[α]) * 5^{α−i} }
fin_función
```

Referencias:

- Peña Marí, Ricardo; (1998) Diseño de Programas. Formalismo y Abstracción. Prentice Hall.
- Castro Rabal, Jorge; Cucker Farkas, Felipe (1993). Curso de programación. McGraw-Hill / Interamericana de España, S.A.
- Bálcazar José Luis (2001). Programación Metódica. McGraw-Hill.