TRANSFORMACIÓN DE ALGORITMOS RECURSIVOS

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

```
entero funcion 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
```

1. De Función Recursiva NO FINAL a Función Recursiva FINAL

Función Recursiva NoFinal

```
tipo funcion f_rec( x̄ )
inicio
    si caso_base?( x̄ ) entonces
        devolver sol( x̄ )
    si_no
        devolver comb(f_rec(suc( x̄ )), x̄ )
fin_si
fin_funcion
```

- El parámetro formal $\bar{x}$ debe entenderse como una tupla $x_1, x_2, \ldots, x_n$ de parámetros.
- **caso_base?($\bar{x}$):** es una expresión lógica que determina si 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 da el 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 del Ejercicio

```
entero funcion 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
```

- $\bar{x}$ **={ y, z, n, i }** *tupla de parámetros formales*
- **caso_base?(** $\bar{x}$ **):** **i=n**
- **sol(** $\bar{x}$ **):** **y[i]*z[i]**
- **suc(** $\bar{x}$ **):**

  suc(y)=y,
  suc(z)=z,
  suc(n)=n,
  suc(i)=i+1

- **comb(f_rec(suc(** $\bar{x}$ **)),** $\bar{x}$ **):**
  y[i]*z[i] + 5*fun(y, z, n, i+1)

a) Generalización:

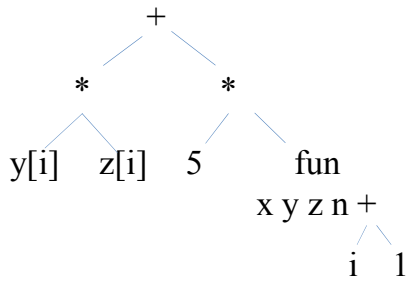
- La inmersión f_recFinal se obtiene considerando la expresión del caso general de la función a transformar:

  f_rec(x) = comb(f_rec(suc(x)), x)

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

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

Árbol Sintáctico (función sumergida)



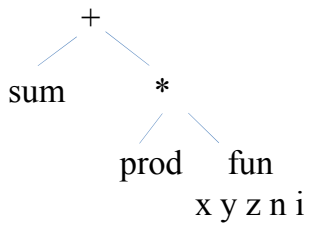
a) Se conserva el camino que va desde la raíz hasta la invocación a fun.

*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 *w0*:

- **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(*x*, *w0*)

Llamada Inicial:

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.

b) Desplegado

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

```
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
```

2. Reorganización de los términos obtenidos

Aplicación de la propiedad distributiva del producto y Reorganización de los términos obtenidos

```
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
```

Aplicación de la propiedad asociativa para encontrar los sucesores de los parámetros de inmersión que deben ir en la llamada Recursiva Final.

```
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
```

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 la siguiente:

```
entero funcion 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 Σ_{α=i}^{n} (y[α] * z[α]) * 5^{α−i} }
fin_función
```

Y la función que realiza la llamada inicial a esta función:

```
entero funcion 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
```

2. De Función Recursiva No FINAL a ITERATIVA No FINAL

Realiza la transformación de la función recursiva no final a Iterativa siguiendo el esquema siguiente y posteriormente realiza la optimización del código.

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

- La variable local *c* sirve para contar el número de veces que se realiza la llamada recursiva.
- 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 funcion 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
                y ← y
                z← z
                n ← n
                i ← i+1
        fin_mientras
        res ←  y[i]*z[i]
        mientras c ≠0 hacer
                c ← c – 1
                y ←  y
                z ←  z
                n ←  n
                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 NO FINAL

- Para eliminar el primer bucle mientras:
  - El número de veces que se realiza la llamada recursiva será n-i veces.
  - Si suc(x)=x entonces se pueden eliminar esas instrucciones.
  - El parámetro i, el último valor que toma en el caso base es n, es posible realizar directamente esta inicialización.
- Para optimizar el segundo bucle mientras:
  - Si $suc(x)^{-1}$=x entonces se pueden eliminar esas instrucciones.

entero **funcion** **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. De Función Recursiva FINAL a ITERATIVA FINAL

```
tipo funcion f_iter(x)
inicio
        mientras ¬ caso_base? (x) hacer
                x ←suc (x)
        fin_mientras
        devolver sol(x)
fin_funcion
```

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

entero **funcion** **fun_iterativafinal** (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**
n ← **n**
z ← **z**
y ← **y**
fin_mientras
{devuelve $\sum_{\alpha=i}^{n}(y[\alpha]*z[\alpha])*5^{\alpha-i}$ }
fin_función

entero **funcion** **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_iterativafinal(y, z, n, i, 0, 1)**
fin_función

OPTIMIZACIÓN DE LA FUNCIÓN ITERATIVA FINAL

- Si suc(x)=x entonces se pueden eliminar esas instrucciones.
- Los parámetros de inmersión pueden ser variables locales y ser inicializados a los valores asignados en la función que realiza la llamada inicial a la función recursiva final(valores neutros).
- La función llamada inicial puede, por tanto, ser eliminada.

```
entero funcion fun_iterativafinal (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
{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.