CI2611 - Algoritmos y estructuras I

Parcial 3

Daniel Delgado

Abr-Jul 2025

Contenido

1 Resumen Parcial 3

1.1 Recursión

1.1.1 Procedimientos y Funciones Recursivas

proc $p\ (in\ x;\ in-out\ y;\ out\ z)$

$\{P_{def}\}$

$\{Q_{def}\}$

$\{Cota : t\}$

[

...

$\{P_{llam}\}\ p(E, a, b)\ \{Q_{llam}\}$

...

]

func $f\ (x : T) \to T'$

$\{P_{def}\}$

$\{Q_{def}\}$

$\{Cota : t\}$

[⟨ Cuerpo que debe tener una llamada a f ⟩

$>> E$

]

1.1.2 Correctitud de Recursividad

Prueba 1

Se debe demostrar la siguiente tripleta, igualando la función de cota a una constante. Esta demostración implica aplicar las técnicas que se vieron para el primer parcial de acuerdo al cuerpo del procedimiento.

$\{P_{def} \wedge t = X\}$

⟨ Cuerpo del procedimiento ⟩

$\{Q_{def}\}$

Prueba 2

Demostración de la disminución de la cota en cada llamada recursiva en la primera prueba.

2.1.$[P_{llam} \Rightarrow (P_{def} \wedge t < X)(x, y := E, a)]$

2.2.$[P_{llam}(a, b := A, B) \wedge Q_{def}(x, y_0, y, z := E(a, b := A, B), A, a, b) \Rightarrow Q_{llam}]$

Prueba 3

Demostración de función acotada.

$[P_{def} \Rightarrow t \geq 0]$

1.2 Invariante de Cola

1.2.1 Caso 1

Se tiene una función F de la siguiente forma,

$$F(X) = \begin{cases} h_1(X) & \text{si } b_1(X) \\ h_2(X) & \text{si } b_2(X) \\ \dots & \\ h_m(X) & \text{si } b_m(X) \\ F(g_1(X)) & \text{si } \neg b_1(X) \wedge \neg b_2(X) \dots \neg b_m(X) \wedge c(X) \\ F(g_2(X)) & \text{si } \neg\neg b_1(X) \wedge \neg b_2(X) \dots \neg b_m(X) \wedge \neg c(X) \end{cases} \tag{1}$$

Donde,

$X = x_1, x_2, \dots, x_k$ con $k \in \mathbb{N}$

h_i con $i \in \mathbb{N}$ corresponde a los casos bases definidos por F.

$F(g_j(X))$ con $j \in \mathbb{N}$ corresponde a los casos recursivos definidos por F.

Se quiere derivar un programa cuya postcondición es $r = F(\overline{A})$ para algún $\overline{A} = (a_1, a_2, \dots, a_k)$. Se define el Invariante de cola como.

$$\text{Inv: } F(X) = F(\overline{A})$$

El programa derivado tiene la forma siguiente.

$$
\begin{array}{l}
[\\
\quad const\ x_1, x_2, \ldots, x_k : T; \\
\quad x := a_1, a_2, \ldots, a_k \\
\quad \{Inv : F(x_1, x_2, \ldots, x_k) = F(a_1, a_2, \ldots, a_k)\} \\
\qquad do\ \neg b_1(x_1, x_2, \ldots, x_k) \wedge \neg b_2(x_1, x_2, \ldots, x_k) \ldots \neg b_m(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\quad if\ c(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\qquad x := g_1(x_1, x_2, \ldots, x_k) \\
\qquad\quad [\,]\ \neg c(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\qquad x := g_2(x_1, x_2, \ldots, x_k) \\
\qquad\quad fi \\
\qquad od \\
\qquad if\ b_1(x_1, x_2, \ldots, x_k) \wedge \neg b_2(x_1, x_2, \ldots, x_k) \ldots \neg b_m(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\quad r = h_1(x_1, x_2, \ldots, x_k) \\
\qquad [\,]\ \neg b_1(x_1, x_2, \ldots, x_k) \wedge b_2(x_1, x_2, \ldots, x_k) \ldots \neg b_m(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\quad r = h_2(x_1, x_2, \ldots, x_k) \\
\qquad \ldots \\
\qquad [\,]\ \neg b_1(x_1, x_2, \ldots, x_k) \wedge \neg b_2(x_1, x_2, \ldots, x_k) \ldots b_m(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\quad r = h_m(x_1, x_2, \ldots, x_k) \\
\qquad [\,]\ \neg b_1(x_1, x_2, \ldots, x_k) \wedge \neg b_2(x_1, x_2, \ldots, x_k) \ldots \neg b_m(x_1, x_2, \ldots, x_k) \rightarrow \\
\qquad\quad skip \\
\qquad fi \\
\quad \{r = F(a_1, a_2, \ldots, a_k)\} \\
]
\end{array}
$$

Para el caso particular de un solo caso base, una sola variable y un solo caso recursivo.

$$F(x) = \begin{cases} h(x) & \text{si } b(x) \\ F(g(x)) & \text{si } \neg b(x) \end{cases} \tag{2}$$

Donde $h(x)$ es fácil de calcular en una sola iteración.

El invariante está definido por la expresión siguiente, con A un valor cualquiera en donde F está definido y represemta el valor de x para el cual se desea obtener un resultado del programa.

$$\text{Inv: } F(x) = F(A)$$

De donde se deriva el siguiente esquema de programa.

[

$const\ x : T;$

$x := A$

$\{Inv : F(x) = F(A)\}$

$do\ \neg b(x) \rightarrow$

$x := g(x)$

od

$r = h(x)$

$\{r = F(A)\}$

]

1.2.2 Caso 2

Se tiene una función F de la siguiente forma.

$$F(x) = \begin{cases} h(x) & \text{si } b(x) \\ F(g(x)) \otimes j(x) & \text{si } \neg b(x) \end{cases} \tag{3}$$

Donde $\otimes$ es una operación **asociativa** con **elemento neutro** e.

Se quiere derivar un programa cuya postcondición es $r = F(A)$ para algún A en el dominio de F.

El invariante de cola se define como sigue.

"Resultado Final" = "Lo que lleva" $\otimes$ "Lo que falta por calcular".

Lo cual se traduce en la expresión matemática siguiente.

$$\text{Inv: } F(A) = r \otimes F(x)$$

1. Se inicializan las variables r y x.

   $r, x := e, A$

   Esto garantiza que el invariante se cumple antes de entrar al bucle.

2. Si $b(x)$ se cumple entonces $F(x) = h(x)$ entonces se cumple.

   $F(A) = r \otimes F(x) \equiv F(A) = r \otimes h(x)$

   Esto garantiza que el invariante se cumple al finalizar el bucle.

3. Si $b(x)$ no se cumple entonces $F(x) = F(g(x)) \otimes j(x)$, por lo tanto se tiene lo siguiente.

   $F(A) = r \otimes F(x) \equiv r \otimes (F(g(x)) \otimes j(x)) \equiv (r \otimes j(x)) \otimes F(g(x))$

   $\equiv \quad \langle F(x) \rangle$

   $r \otimes (F(g(x)) \otimes j(x))$

   $\equiv \quad \langle Asociatividad \rangle$

   $(r \otimes j(x)) \otimes F(g(x))$

   El invariante se cumple siempre que se hagan las siguientes asignaciones dentro del bucle.

$$r := r \otimes j(x)$$

$$x := g(x)$$

   De esta manera el invariante se cumple durante el bucle.

Finalmente, el esquema del programa quedará.

[

$const\ x : T;$

$x, r := A, e;$

$\{Inv : F(A) = r \otimes F(x)\}$

$do\ \neg b(x) \rightarrow$

$r, x := r \otimes j(x), g(x)$

od

$\{F(A) = r \otimes a\}$

$r = h(x)$

$\{r = F(A)\}$

]

1.3 Recursión de Cola

Regularmente se busca transformar una función recursiva en una función con recursión de cola, para ello es necesario que la última llamada recursiva sea la última instrucción en la función, de manera que no hay operaciones pendientes después de la llamada recursiva.

Esquema general para el análisis.

1. **Identificar** las expresiones de la recursión original que hacen que no sea de cola.
2. **Sustituir** las llamadas recursivas por acumuladores, los cuales mantendrán actualizados los cálculos realizados en iteraciones pasadas. Estos acumuladores será uno por cada llamada recursiva y serán parámetros de la función. Se deberá indicar una expresión matemática que de forma general permita determinar el valor del acumulador: al inicio, al final y en una actualización del iterador.
3. **Agregar o Identificar (si ya existe)** en los parámetros un índice que permita determinar en qué iteración nos encontramos. Puede ser creciente o decreciente.
4. **Transformar** la llamada recursiva en una de cola de tal manera que el iterador disminuya o aumente en uno y el acumulador se actualice debidamente.
5. Identificar la expresión que representa el caso base la cual debe estar escrita en función del acumulador.
6. Identificar el valor inicial del acumulador de tal manera que el resultado sea igual al de la recursión normal.

— PROBLEMAS RESUELTOS —

2 Recursión

2.1 Ejercicios de clase

2.1.1 Practica Ene-Mar 2025

Demuestre el siguiente procedimiento recursivo.

proc $productoria(entrada\ i, N : int; A :\ array[0..N)\ of\ int; salida\ r : int)$

$\{Pre : 0 \leq i \leq N\}$;

$\{Post : r = (\prod k | i <= k < N : A[k])\}$;

$\{Cota : N - i\}$;

[

$if\ i = N \rightarrow r := 1$

[ ] $i < N \rightarrow$

$productoria(i + 1, N, A, r)$;

$r := A[i] * r$

fi

]

Solución

- Prueba 1.

  Se debe probar la siguiente tripleta.

  $\{P_{def} \wedge t = X\}\ S\ \{Q_{def}\}$

  Es decir,

$\{0 \leq i \leq N \wedge N - i = X\}$

$if\ i = N \rightarrow r := 1$

$[\,]\ i < N \rightarrow$

$productoria(i + 1, N, A, r);$

$r := A[i] * r$

fi

$\{r = (\prod k | i <= k < N : A[k])\}$

Se debe probar la selección, con las siguientes pruebas.

1.1 $P \Rightarrow B_0 \vee B_1$

1.2.0 $\{P \wedge B_0\}S_0\{Q\}$

1.2.1 $\{P \wedge B_1\}S_1\{Q\}$

Prueba 1.1.

$0 \leq i \leq N \wedge N - i = X \Rightarrow i = N \vee i < N$

Suposición del antecedente, empezando por true.

Suponemos:

$H_0 :\ 0 \leq i \leq N$

$H_1 :\ N - i = X$

$true$

$\equiv \quad \langle H0 \rangle$

$0 \leq i \leq N$

$\equiv \quad \langle a \leq b \leq c \equiv a \leq b \wedge b \leq c \rangle$

$0 \leq i \wedge i \leq N$

$\equiv \quad \langle p \wedge q \Rightarrow p \rangle$

$i \leq N$

$\equiv \quad \langle$ Aritmética $\rangle$

$i = N \vee i < N$

■

Prueba 1.2.0

Se debe probar la siguiente tripleta.

$\{0 \leq i \leq N \wedge N - i = X \wedge i = N\}\ r := 1\ \{r = (\prod k | i <= k < N : A[k])\}$

Por regla de la asignación, debemos probar el siguiente predicado.

$0 \leq i \leq N \wedge N - i = X \wedge i = N\ \Rightarrow (r = (\prod k | i <= k < N : A[k]))(r := 1)$

Por suposición del antecedente, empezando por el consecuente para alcanzar true.

Suponemos:

$H_0 :\ 0 \leq i \leq N$

$H_1 :\ N - i = XN$

$H_2 :\ i = N$

$(r = (\prod k | i <= k < N : A[k]))(r := 1)$

$\equiv$ $\langle$ Sustitucion Textual $\rangle$

$1 = (\prod k | i <= k < N : A[k])$

$\equiv$ $\langle$ Sustitución por $H2\rangle$

$1 = (\prod k | N <= k < N : A[k])$

$\equiv$ $\langle$ Rango vacío $(\prod k | false : x) = 1\rangle$

$1 = 1$

$\equiv$ $\langle$ Aritmética $\rangle$

$true$

■

Prueba 1.2.1

Se debe probar la siguiente tripleta.

$\{0 \leq i \leq N \wedge N - i = X \wedge i < N\}\ productoria(i+1, N, A, r); r := A[i] * r\ \{r = (\prod k | i <= k < N : A[k])\}$

Dado que se tiene una secuenciación se debe obtener la aserción intermedia, para lo cual podemos usar el precondición más débil para la asignación.

$wp(r := A[i] * r).(r = (\prod k | i <= k < N : A[k]))$

$\equiv$ $\langle$ Definición wp$\rangle$

$(r = (\prod k | i <= k < N : A[k]))(r := A[i] * r)$

$\equiv$ $\langle$ Sustitución textual$\rangle$

$A[i] * r = (\prod k | i <= k < N : A[k])$

De esta manera tenemos las siguientes tripletas.

$\{0 \leq i \leq N \wedge N - i = X \wedge i < N\}$

$productoria(i + 1, N, A, r);$

$\{A[i] * r = (\prod k | i <= k < N : A[k])\}$

$r := A[i] * r$

$\{r = (\prod k | i <= k < N : A[k])\}$

Como la precondición más débil asegura correctitud, se considera que la asigación ya está probada, por lo tanto, los queda probar la siguiente.

$\{0 \leq i \leq N \wedge N - i = X \wedge i < N\}$

$productoria(i + 1, N, A, r);$

$\{A[i] * r = (\prod k | i <= k < N : A[k])\}$

Dado que es un procedimiento recursivo se debe probar.

1. $P_{llam} \Rightarrow (P_{def} \wedge t < X)(x, y := E, a)$

2. $P_{llam}(a, b := A, B) \wedge Q_{def}(x, y_0, y, z := E(a, b := A, B), A, a, b) \Rightarrow Q_{llam}$

Prueba 1.2.1.1

$0 \leq i \leq N \wedge N - i = X \wedge i < N \Rightarrow (0 \leq i \leq N \wedge N - i < X)(i, N, A := i + 1, N, A)$

Suposición del antecedente, empezando por el consecuente para alcanzar true.

Suponiendo:

$H_0 :\ 0 \leq i \leq N$

$H_1 :\ N - i = X$

$H_2 :\ i < N$

$(0 \leq i \leq N \wedge N - i < X)(i, N, A := i + 1, N, A)$

$\equiv$ $\langle$ Sustitución textual$\rangle$

$0 \leq i + 1 \leq N \wedge N - (i + 1) < X$

$\equiv$ $\langle$ Aritmética$\rangle$

$0 \leq i + 1 \leq N \wedge N - i - 1 < X$

$\equiv$ $\langle$ Sustitución por $H_1\rangle$

$$
\begin{array}{ll}
 & 0 \leq i+1 \leq N \wedge X-1 < X \\
\equiv & \quad \langle\ a-1 < a \equiv true \rangle \\
 & 0 \leq i+1 \leq N \\
\equiv & \quad \langle\ a \leq b \leq c \equiv a \leq b \wedge b \leq c \rangle \\
 & 0 \leq i+1 \wedge i+1 \leq N \\
\equiv & \quad \langle\ \text{Por } H_2 :\ i < N \equiv true \Rightarrow i+1 \leq N \rangle \\
 & 0 \leq i+1 \\
\equiv & \quad \langle\ \text{Por } H_1 :\ 0 \leq i \equiv true \Rightarrow 0 \leq i+1 \rangle \\
 & true
\end{array}
$$

■

Prueba 1.2.1.2

3 Invariante de Cola

3.1 Ejercicios Kaldewaij

3.1.1 Ejemplo

Derivar programa para la exponenciación x^y.

$[$

$const\ A, B : int;$

$var\ r : int;$

$\langle Exponenciancion \rangle$

$\{r = A^B\}$

$]$

1. Se define la función F

$$F(x,y) = \begin{cases} 1 & \text{si } y = 0 \\ 1 \cdot F(x \cdot x, y\ div\ 2) & \text{si } y \neq 0 \wedge y \mod 2 = 0 \\ x \cdot F(x, y-1) & \text{si } y \neq 0 \wedge y \mod 2 = 1 \end{cases} \tag{4}$$

2. Se define el invariante: $P : F(A,B) = r \cdot F(x,y) \equiv A^B = r \cdot x^y$.

3. Variables y valores iniciales.

   EL operador principal de F es la multiplicación cuyo elemento neutro es 1, por lo tanto nuestra variable de resultador r se inicializa en 1, mientras que x se inicializa en su valor máximo A y y igualmente en su valor máximo B.

$$r, x, y := 1, A, B$$

4. Para validar que el invariante se cumple en todo momento verificamos.

   - Cuando $y \neq 0$

     $F(A,B) = r \cdot F(x,y)$

     $\equiv \quad \langle F(x,y) \text{ en } y \neq 0 \rangle$

     $F(A,B) = r \cdot 1$

$\equiv$ $\langle$ Operando $\rangle$

$F(A, B) = r$

$\equiv$ $\langle$ Definición de $F\rangle$

$A^B = r$

- Cuando $y = 0$.

  - Cuando $y \mod 2 = 0$.

    $F(A, B) = r \cdot F(x, y)$

    $\equiv$ $\langle$ Definición de $F\rangle$

    $F(A, B) = r \cdot F(x \cdot x, y \; div \; 2)$

    $\equiv$ $\langle P \rangle$

    $F(A, B) = r \cdot (x \cdot x)^y$

    $\equiv$ $\langle$ S.T sobre P $\rangle$

    $P(x, y := x \cdot x, y \; div \; 2)$

    Las actualizaciones son las siguientes.

    $$x, y := x \cdot x, y \; div \; 2$$

  - Cuando $y \mod 2 = 1$.

    $F(A, B) = r \cdot F(x, y)$

    $\equiv$ $\langle$ Definición de $F\rangle$

    $F(A, B) = r \cdot x \cdot F(x, y - 1)$

    $\equiv$ $\langle P \rangle$

    $F(A, B) = r \cdot x \cdot (x)^{y-1}$

    $\equiv$ $\langle$ S.T sobre P $\rangle$

    $P(r, y := r \cdot x, y - 1)$

    Las actualizaciones son las siguientes.

    $$r, y := r \cdot x, y - 1$$

Finalmente, el programa es el siguiente.

$$
\begin{array}{l}
[\\
\quad const\ A, B : int; \\
\quad var\ r, x, y : int; \\
\quad r, x, y := 1, A, B; \\
\quad \{Inv:\ r \cdot x^y = A^B \wedge y \geq 0\}\{Cota:\ y\} \\
\quad\quad do\ y \neq 0 \rightarrow \\
\quad\quad\quad if\ y\ \bmod 2 = 0 \rightarrow \\
\quad\quad\quad\quad x, y := x \cdot x, y\ div\ 2 \\
\quad\quad\quad [\,]\ y\ \bmod 2 = 1 \rightarrow \\
\quad\quad\quad\quad r, y := r \cdot x, y - 1 \\
\quad\quad\quad fi \\
\quad\quad od \\
\quad \{r = A^B\} \\
]
\end{array}
$$

3.1.2 Ejercicio 0

Derive un programa que calcule $A * B$ donde A y B son números naturales. Solo se permite el uso de div 2, mod 2, *2, suma y resta.

Solución

1. Se define la función recursiva.

$$F(x,y) = \begin{cases} 0 & \text{si } y = 0 \\ x + F(x, y-1) & \text{si } y \neq 0 \end{cases} \tag{5}$$

2. Se define como invariante el siguiente predicado.

$$P : F(A,B) = r + F(x,y) \equiv A \cdot B = r + x \cdot y$$

3. Valores iniciales.

   Para que el invariante sea cierto antes de la entrada en el bucle, se debe cumplir que $r = 0$ (elemento neutro de la suma, el cual es el operador principal de F), $x = A$, $y = B$. De esta manera, el invariante propuesto queda como se muestra a continuación.

   $A \cdot B = 0 + A \cdot B = A \cdot B$

4. Para validar que el invariante se cumple durante el bucle veamos qué pasa cuando $y \neq 0$.

   $F(A,B) = r + F(x,y)$

   $\equiv$ $\langle$ Definición de $F \rangle$

   $F(A,B) = r + x + F(x, y-1)$

   $\equiv$ $\langle$ S.T sobre P $\rangle$

   $P(r, y := r + x, y - 1)$

   Las actualizaciones son las siguientes.

$$x, y := r + x, y - 1$$

   Para validar que el invariante se cumple al finalizar el bucle veamos qué pasa cuando $y = 0$.

   $F(A,B) = r + F(x,y)$

   $\equiv$ $\langle$ Definición de $F \rangle$

$$F(A, B) = r + 0$$

$\equiv$ $\langle$ Aritmética $\rangle$

$$F(A, B) = r$$

Entonces, se dice que r almacena el resultado deseado, al momento de finalizar el bucle.

Finalmente, el programa es el siguiente.

$[$

$const\ A, B : int;$

$var\ r, x, y : int;$

$r, x, y := 0, A, B;$

$\{Inv :\ A \cdot B = r + x \cdot y \wedge y \geq 0\}\{Cota :\ y\}$

$do\ y \neq 0 \rightarrow$

$r, y := r + x, y - 1$

od

$\{r = A \cdot B\}$

$]$

Finalmente, el programa es el siguiente.

$$
\begin{array}{l}
[\\
\quad const\ N : int; \\
\quad var\ n, r : int; \\
\quad n := N; \\
\quad \{Inv :\ fusc(n) = fusc(N) \wedge N \geq 0\}\{Cota :\ n\} \\
\quad\quad do\ n \neq 0 \wedge n \neq 1 \rightarrow \\
\quad\quad\quad if\ n\ \bmod 2 = 0 \rightarrow \\
\quad\quad\quad\quad n := n\ div\ 2 \\
\quad\quad\quad [\,]\ n\ \bmod 2 = 1 \rightarrow \\
\quad\quad\quad\quad n := n + n\ div\ 2 \\
\quad\quad\quad fi \\
\quad\quad od \\
\quad\quad if\ n = 0 \rightarrow \\
\quad\quad\quad r = 0 \\
\quad\quad [\,]\ n = 1 \rightarrow \\
\quad\quad\quad r = 1 \\
\quad\quad fi \rightarrow \\
\quad \{r = fusc(N)\} \\
]
\end{array}
$$

3.1.3 Ejercicio 4

Resolver.

[

$const\ N : int$;

$const\ f : array[0..N)of int$;

$var\ r : bool$;

S

$\{r \equiv (\exists\ i | 0 \leq i < N : f[i] = 0)\}$

]

Definiendo,

$G(n) \equiv (\exists i | n \leq i < N : f[i] = 0)$ para $0 \leq n \leq N$

y derivando la relación de recurrencia para G.

Solución

1. Se define la función recursiva.

$$G(n) \equiv \begin{cases} true & \text{si } f[n] = 0 \\ false & \text{si } f[n] \neq 0 \wedge n = 0 \\ G(n-1) & \text{si } f[n] \neq 0 \wedge n \neq 0 \end{cases} \tag{6}$$

2. Invariante de cola para este caso.

$$G(n) \equiv G(N) \wedge 0 \leq n \leq N$$

3. El programa final queda.

$$
\begin{array}{l}
[\\
\quad const\ N : int; \\
\quad const\ f : array[0..N)\ of\ int; \\
\quad var\ r : bool; \\
\quad var\ n : int; \\
\quad \{N \geq 0\} \\
\quad\quad n := N; \\
\quad\quad \{Inv : G(n) \equiv G(N) \wedge 0 \leq n \leq N\}\{Cota : n\} \\
\quad\quad do\ f[n] \neq 0 \wedge n \neq 0 \rightarrow \\
\quad\quad\quad n := n - 1 \\
\quad\quad od \\
\quad\quad r := f[n] = 0 \\
\quad \{r \equiv (\exists\ i | 0 \leq i < N : f[i] = 0)\} \\
]
\end{array}
$$

3.1.4 Ejercicio 2 (página 81)

La función A está definida en los números naturales.

$A(0) = 1$

$A(2n) = 2 * A(n)\ n \geq 1$

$A(2n+1) = n + A(2n)\ n \geq 0$

Derivar un programa para el cálculo de $A(N)$, $N \geq 0$

Solución

1. Se define la función recursiva.

$$A(n) \equiv \begin{cases} 1 & \text{si } n = 0 \\ 2 * A(n) & \text{si } n \geq 0 \wedge n \mod 2 = 0 \\ n + A(2n) & \text{si } n \geq 1 \wedge n \mod 2 = 1 \end{cases} \tag{7}$$

2. Invariante de cola.

$$P : F(N) = r + F(n)$$

3. Para el cumplimiento de este Invariante antes del bucle las variables r y n deben ser inicializadas como se muestra a continuación.

$$r, n := 0, N$$

4. Cuando $n \neq 0$, estamos dentro del bucle. Se tiene dos posibilidades.

   - Si $n \mod 2 = 0$, se debe cumplir el invariante.

     $F(N) = r + F(n)$

     $\equiv$ $\langle$ Definición de $F\rangle$

     $F(N) = r + 2 * A(n)$

3.2 Ejercicios Tarea 7 Abr-Jul 2015

3.2.1 Ejercicio 1

La definición recursiva de los números de Catalán es

$$C_n = \begin{cases} 1 & \text{si } n = 0 \\ \frac{2(2n-1)}{n+1} C_{n-1} & \text{si } n > 0 \end{cases} \tag{8}$$

Escriba una función recursiva que implemente en GCL esta definición.

Escriba un programa que calcule los primeros N números de catalán y los alamacene en un arreglo A.

Pruebe que la función recursiva es correcta.

Solución

1. La función recursiva se da en el resultado.

$$C(n) = \begin{cases} 1 & \text{si } n = 0 \\ \frac{2(2n-1)}{n+1} C(n-1) & \text{si } n > 0 \end{cases} \tag{9}$$

2. El invariante de cola es

$$P : C(N) = r * C(n)$$

3. Valores iniciales.

   Dado que el operador principal de $C(n)$ es la múltiplicaicón y para que el invariante sea cierto antes de la entrada en el bucle se debe inicializar con $r = 1$ (elemento neutro de la multiplicación), luego con $n = N$ el invariante se cumple.

   $r * C(n) = 1 * C(N) = C(N)$

4. Se verifica el invariante dentro del bucle cuando $n > 0$.

   $C(N) = r * C(n)$

   $\equiv$ $\langle$ Definición de $C(n)\rangle$

   $C(N) = r * \frac{2(2n-1)}{n+1} C(n-1)$

   $\equiv$ $\langle$ Definición de $P\rangle$

   $P(r, n : r * \frac{2(2n-1)}{n+1}, n-1)$

   Las actualizaciones de r y n son las siguientes.

$$r, n := r * \frac{2(2n-1)}{n+1}, n-1$$

Ahora verifiquemos que pasa con el invariante al finalizar el bucle.

$C(N) = r * C(n)$

$\equiv$ $\langle$ S.T $n = 0\rangle$

$C(N) = r * C(0)$

$\equiv$ $\langle$ Definición de $C\rangle$

$C(N) = r * 1$

$\equiv$ $\langle$ Aritmética $\rangle$

$C(N) = r$

Por lo tanto al finalizar el bucle r almacena el valor de $C(N)$.

5. La función que calcula el número de catalán para los primeros N enteros.

func $catalan(N : int) \rightarrow float$

$\{pre : N \geq 0\}$

$\{post : (\forall k | 0 \leq k < n :)\}$

[

$var\ r : float;$

$var\ n : int;$

$r, n := 1, N;$

$\{Inv : C(N) = r * C(n)\}\{Cota : n\}$

$do\ n \neq 0 \rightarrow$

$A[N - n] := r$

$r, n := r * \frac{2(2n-1)}{n+1}, n - 1;$

od

$\{r = C(N)\}$

$>> r$

]

6. Programa que calcula los N primeros números de Catalán y los alamacena en un arreglo A de tamaño N.

$$
\begin{array}{l}
[\\
\quad const\ N : int; \\
\quad const\ A : array[0..N)\ of\ float; \\
\quad var\ r : float; \\
\quad var\ n : int; \\
\quad\quad \{N \geq 0\}; \\
\quad\quad r, n := 1, N; \\
\quad\quad \{Inv : C(N) = r * C(n)\}\{Cota : n\} \\
\quad\quad do\ n \neq 0 \rightarrow \\
\quad\quad\quad A[N - n] := r \\
\quad\quad\quad r, n := r * \frac{2(2n-1)}{n+1}, n - 1; \\
\quad\quad od \\
\quad \{r = C(N)\} \\
]
\end{array}
$$

3.2.2 Ejercicio 2-c

Dado dos vectores A y B, halle una función recursiva para decir si son iguales.

Solución

func $igualesRec(N : int, n : int, A : array[0..N)\ of\ int, B : array[0..N)\ of\ int) \rightarrow bool$

$\{pre : 0 \leq n\}$

$\{post : iguales \equiv (\forall k | 0 \leq k < n : A[k] = B[k])\}$ // Cuando va desde el caso base a los recursivos

[

$var\ r : bool;$

$var\ r := true;$

$if\ n = 0 \rightarrow r := r \wedge A[n] = B[n]$

$[\,]\ n \neq 0 \rightarrow r := r \wedge igualesRec(N, n-1, A, B)$

$>> r$

]

Esta función debe ser llamada por otra función de la siguiente forma.

func $iguales(N : int, A : array[0..N)\ of\ int, A : array[0..N)\ of\ int) \rightarrow bool$

$\{pre : N \geq 0\}$

$\{post : iguales \equiv (\forall k | 0 \leq k < N : A[k] = B[k])\}$

[

$var\ n : int;$

$var\ res : bool;$

$if\ N = 0 \rightarrow res := true;$

$[\,]\ N > 0 \rightarrow n := N - 1;\ res := igualesRec(N, n, A, B);$

$n := N - 1;$

$res := igualesRec(N, n, A, B)$

$fi;$

$>> res$

]

Otra forma de expresar el caso base.

func $igualesRec(N : int, n : int, A : array[0..N)\ of\ int, B : array[0..N)\ of\ int) \rightarrow bool$

$\{pre : 0 \leq n\}$

$\{post : iguales \equiv (\forall k | 0 \leq k < n : A[k] = B[k])\}$ // Cuando va desde el caso base a los recursivos

[

$var\ r : bool;$

$if\ n = -1 \rightarrow r := true$

$[\,]\ n \geq 0 \rightarrow r := A[n] = B[n] \wedge igualesRec(N, n-1, A, B)$

$>> r$

]

3.2.3 Ejercicio 2-e

Dada una matriz de tamaño NxN devolver la suma de los elementos de la diagonal.

Solución

func $sumaDiagRec(N : int, n : int, A : array[0..N)x[0..N)\ of\ int) \rightarrow int$

$\{pre : n \leq N\}$

$\{post : sumaDiagRec \equiv (\sum k | 0 \leq k < n : A[k][k])\}$

[

$var\ r : int;$

$var\ r := 0;$

$if\ n = -1 \rightarrow r := 0$

$[\ ]\ n \geq 0 \rightarrow r := A[n][n] + sumaDiagRec(N, n-1, A)$

fi

$>> r$

]

4 Recursión de Cola

4.1 Ejercicios de clase

4.1.1 Ejercicio de clase 1

Convertir la función de sumatoria de un arreglo de enteros en una recursión de cola.

func $sum(N : int, A : array[0..N)\ of\ int, i : int) \to int$

[

$var\ r : int;$

$if\ i = N \to r := 0$

$[\,]\ i < N \to r := A[i] + sum(A, N, i + 1)$

fi

$>> r$

]

Solución

1. **Llamadas recursivas**: Identificar la llamada recursiva.

   $sum(A, N, i + 1)$

2. **Acumuladores**: Definir el acumulador en base a la llamada recursiva, la cual corresponde a la suma de cada uno de los elementos del arreglo desde 0 hasta i esto es posible representarlo matemáticamente como una sumatoria.

   $a = (\sum k | 0 \leq k < i : A[k])$

3. **Iterador**: Dado que en cada llamada recursiva i aumenta en 1 entonces este i puede ser usado como iterador, el cual ya existe en la llamada recursiva.

   Se agrega el acumulador a la definición de la nueva suma.

   func $sumCola(N : int, A : array[0..N)\ of\ int, i : int, a : int) \to int$

   $\{a = (\sum k | 0 \leq k < i : A[k])\}$

4. **Actualización de** a: Veamos qué pasa en $i + 1$.

$$
\begin{aligned}
& (\sum k | 0 \leq k < i : A[k])(i := i + 1) \\
\equiv \quad & \langle S.T \rangle \\
& (\sum k | 0 \leq k < i + 1 : A[k]) \\
\equiv \quad & \langle \text{ Último término } \rangle \\
& (\sum k | 0 \leq k < i : A[k]) + A[i] \\
\equiv \quad & \langle \text{ Definición de } a \rangle \\
& a + A[i]
\end{aligned}
$$

Esto corresponde a la expresión de a en $i + 1$.

5. **Cuerpo de la función**: se transforma la llamada recursiva en una de cola.

   $r := A[i] + sum(A, N, i + 1) \rightarrow r := sumCola(A, N, i + 1, a + A[i])$

6. **Casos bases**: Se toma el valor final del iterador, cuando $i = N$ se tiene.

   $a = (\sum k | 0 \leq k < N : A[k])$

   Lo cual corresponde a la suma de todos los elementos de A que es lo que soluciona el problema planteado.

7. **Valores iniciales**: Se verifica la expresión de a en $i = 0$.

   $a = (\sum k | 0 \leq k < 0 : A[k]) = 0$

La función finalmente queda.

func $sumCola(N : int, A : array[0..N)\ of\ int, i : int, a : int) \rightarrow int$

[

$var\ r : int;$

$if\ i = N \rightarrow r := a$

$[\,]\ i < N \rightarrow r := sumCola(A, N, i + 1, a + A[i])$

fi

$>> r$

]

La suma de todos los elementos de un arreglo A se hace con la siguiente llamada.

$sumCola(N, A, 0, 0)$

4.1.2 Ejercicio de clase 2

Convertir la función de Fibonacci en una recursión de cola.

func $fib(n : int) \rightarrow int$

[

$var\ r : int;$

$if\ n = 0 \rightarrow r := 0$

$[\,]\ n = 1 \rightarrow r := 1$

$[\,]\ n > 1 \rightarrow r := fib(n-1) + fib(n-2)$

fi

$>> r$

]

Solución

1. **Llamadas recursivas**: Se identifican las llamadas recursivas que en este caso son dos.

   $fib(n-1), fib(n-2)$

2. **Acumuladores**: Como se tienen dos llamadas recursivas se deben definir dos acumuladores.

   $a \rightarrow fib(n-1)$

   $b \rightarrow fib(n-2)$

   Se debe definir una expresión para a y b que caracterice estos acumuladores en todo momento del programa.

   Dado que la función ya está definida en $n = 0$ ($fib(0) = 0$) y en $n = 1$ ($fib(1) = 1$), entonces es posible hacer el cambio de variable $n := n + 2$ quedando las expresiones para a y b de la siguiente manera.

   $a \rightarrow fib(n+1)$

   $b \rightarrow fib(n)$

   Por último, las expresiones que definen a a y b en todo punto de la recursión son las siguientes.

   $a = fib(N - n + 1)\ \forall\ 0 \leq n \leq N$

   $b = fib(N - n)\ \forall\ 0 \leq n \leq N$

3. **Iterador**: El iterador es n.

   $n : int$

4. Expresión para la nueva función.

   func $fibCola(n : int, a : int, b : int) \rightarrow int$

   $\{a = fib(N - n + 1) \wedge b = fib(N - n)\}$

5. **Valores iniciales**: Dada la expresión matemática de a y b conviene que la función empiece en $n = N$.

   $a = fib(N - N + 1) = fib(1) = 1$

   $b = fib(N - N) = fib(0) = 0$

6. **Casos bases**: Dado que el valor inicial es $n = N$ entonces se toma $n = 0$ como un caso base, para lo cual se cumple.

   $a = fib(N - 0 + 1) = fib(N + 1)$

   $b = fib(N - 0) = fib(N)$

   De esta manera, notamos que b tendrá el valor final de la función.

   Cuando $n = 1$ se tiene.

   $a = fib(N - 1 + 1) = fib(N)$

   $b = fib(N - 1) = fib(N - 1)$

   En este caso base a contiene el valor deseado.

   Por lo tanto,

   $$n = 0 \Rightarrow b = fib(N)$$

   $$n = 1 \Rightarrow a = fib(N)$$

7. **Actualización de acumuladores**: Como el valor incial de n resultó conveniente en N, entonces n es decreciente, por lo tanto, los acumuladores se actualizan en $n = n - 1$.

   Para a:

   $fib(N - n + 1)$

   $\equiv$ $\langle$ Sustitución Textual $n := n - 1\rangle$

   $fib(N - (n - 1) + 1)$

   $\equiv$ $\langle$ Aritmética $\rangle$

$fib(N - n + 2)$

$\equiv$ $\langle$ Definición fibonacci $fib(n) = fib(n-1) + fib(n-2)\rangle$

$fib((N - n + 2) - 1) + fib((N - n + 2) - 2)$

$\equiv$ $\langle$ Aritmética $\rangle$

$fib(N - n + 1) + fib(N - n)$

$\equiv$ $\langle$ Definición de $a, b\rangle$

$a + b$

Para b:

$fib(N - n)$

$\equiv$ $\langle$ Sustitución Textual $n := n - 1\rangle$

$fib(N - (n - 1))$

$\equiv$ $\langle$ Aritmética $\rangle$

$fib(N - n + 1)$

$\equiv$ $\langle$ Definición de $a\rangle$

a

Por lo tanto, las actualizaciones de los acumuladores son las siguientes.

$$a, b := a + b, a$$

Finalmente, la función de fibonacci en recursión de cola queda como sigue.

func $fibCola(n : int, a : int, b : int) \rightarrow int$

[

$var\ r : int;$

$if\ n = 0 \rightarrow r := b$

$[\,]\ n = 1 \rightarrow r := a$

$[\,]\ n > 1 \rightarrow r := fibCola(n - 1, a + b, a)$

fi

$>> r$

]

El cáculo de fibonacci para cualquier entero N se obtiene al llamar a la función como sigue, usando los valores iniciales calculados.

$$fibCola(N, 1, 0)$$

4.1.3 Ejercicio de practica Ene-Mar 2025

Usando la técnica de recursión de cola se quiere que derive uan fución recursiva de cola a partir de la siguiente función recursiva.

func $sumaDigits(n : int) \rightarrow int$

[

$var\ r : int;$

$var\ r := 0;$

$if\ n = 0 \rightarrow r := 0$

$[\,]\ n > 0 \rightarrow r := n \mod 10 + sumaDigits(n\ div\ 10)$

fi

$>> r$

]

Solución

1. Identificar las llamadas recursivas.

   $sumaDigits(n\ div\ 10)$

2. Se define el parámetro a que representará la llamada recursiva.

   $a = (\sum k|0)$

4.2 Ejercicios Kaldewaij

4.2.1 Ejercicio 3

La función fusc está definida en los números naturales.

$fusc(0) = 0, fusc(1) = 1$

$fusc(2 * n) = fusc(n)$

$fusc(2 * n + 1) = fusc(n) + fusc(n + 1)$

Derive un programa que calcule fusc(N) con $N \geq 0$.

Solución

1. Se define la función recursiva.

$$fusc(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fusc(n) & \text{si } n \neq 0 \wedge n \neq 1 \wedge n \mod 2 = 0 \\ fusc(n) + fusc(n+1) & \text{si } n \neq 0 \wedge n \neq 1 \wedge n \mod 2 = 1 \end{cases} \tag{10}$$

2. Se define como invariante el siguiente predicado.

$$P : fusc(n) = fusc(N)$$

$$Q : N \geq 0$$