Metodología de la Programación, II/Itis
Septiembre 2008

Ingeniería Técnica en Informática de Sistemas/ Ingeniería en Informática
Facultad de Informática de San Sebastián

1. (2 puntos) Documenta con las aserciones que se marcan el siguiente programa que deja en el array C(1..n+m) el resultado de concatenar ambos arrays A(1..n) y B(1..m), es decir, inserta en C(1..n+m) los elementos de A seguidos de los de B:

Pre= {$n \geq 1 \wedge m \geq 1$}

i:=1; j:=1; k:=1;

(1) INV { $1 \leq i \leq n+1 \wedge 1 \leq j \leq m+1 \wedge k=i+j-1 \wedge \forall l\ (1 \leq l < i \rightarrow C(l)= A(l))$

$\wedge\ \forall l\ (1 \leq l < j \rightarrow C(l+i-1)= B(l))$ }

E= (n-i)+(m-j)

while i ≤ n or j≤ m **loop**

if i ≤ n **then**

C(k):= A(i);

(2) {$1 \leq \underline{i \leq n} \wedge 1 \leq j \leq m+1 \wedge k=i+j-1 \wedge \forall l\ (1 \leq \underline{l \leq i} \rightarrow C(l)= A(l))$

$\wedge\ \forall l\ (1 \leq l < j \rightarrow C(l+i-1)= B(l))$ }

i:= i+1;

else

C(k):= B(j);

j:= j+1;

end if;

(3) {$1 \leq \underline{i \leq n+1} \wedge 1 \leq j \leq m+1 \wedge \underline{k+1}=i+j-1 \wedge$

$\forall l\ (1 \leq \underline{l < i} \rightarrow C(l)= A(l)) \wedge \forall l\ (1 \leq l < j \rightarrow C(l+i-1)= B(l))$ }

k:=k+1;

end loop;

(4) { $\forall l\ (1 \leq l \leq n \rightarrow C(l)= A(l)) \wedge \forall l\ (1 \leq l \leq m \rightarrow C(l+n)= B(l))$ }

2. (2'5 puntos) Deriva formalmente un programa que dado un array A(1..n) calcule la siguiente suma:

$A(1) + (A(1)*A(2)) + (A(1)*A(2)*A(3)) + \cdots + (A(1)*A(2)*\cdots*A(n))$

usando la siguiente especificación pre-post e invariante:

Pre: $n \geq 1$

Post: $s = \sum_{k=1}^{n} \left(\prod_{h=1}^{k} A(h) \right)$

Inv: $0 \leq i \leq n \wedge s = \sum_{k=1}^{i} \left(\prod_{h=1}^{k} A(h) \right) \wedge aux = \prod_{h=1}^{i} A(h)$

Expresión cota: n-i

INICIALIZACIÓN

1. $\{0 \le n \wedge 0 = \sum_{k=1}^{0} (\prod_{h=1}^{k} A(h)) \wedge 1 = \prod_{h=1}^{0} A(h) \}$

i:= 0; s:= 0; aux:= 1;

$\{ \ 0 \le i \le n \ \wedge s = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h) \ \}$ **A.A. (hacia atrás) ≡ INV**

2. $(n \ge 1) \rightarrow (0 \le n \wedge 0 = \sum_{k=1}^{0} (\prod_{h=1}^{k} A(h)) \wedge 1 = \prod_{h=1}^{0} A(h))$

FINALIZACIÓN

B ≡ (i < n)

¬B ≡ (i = n)

$INV \wedge \underline{\neg B} \equiv (\underline{i = n} \wedge s = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h)) \rightarrow$

$\{ s = \sum_{k=1}^{n} (\prod_{h=1}^{k} A(h)))\} \equiv$ **Post**

No hace falta ninguna sentencia

CUERPO DE LA ITERACIÓN y TERMINACIÓN:

$INV \wedge \underline{B} \equiv (0 \le \underline{i < n} \wedge s = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h)) \rightarrow$

$\{ 0 \le \underline{i+1 \le n} \wedge s = \sum_{k=1}^{i+1-1} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i+1-1} A(h) \}$

i := i+1;

$\{ 0 \le \underline{i \le n} \wedge s = \sum_{k=1}^{\underline{i-1}} (\prod_{h=1}^{k} A(h)) \wedge \underline{aux} = \prod_{h=1}^{\underline{i-1}} A(h) \}$ **A.A.** $\rightarrow$

$\{ 0 \le i \le n \wedge s = \sum_{k=1}^{i-1} (\prod_{h=1}^{k} A(h)) \wedge \underline{aux * A(i)} = \prod_{h=1}^{\underline{i}} A(h) \} \rightarrow$

aux := aux * A(i) ;

$\{ 0 \le i \le n \wedge \underline{s} = \sum_{k=1}^{\underline{i-1}} (\prod_{h=1}^{k} A(h)) \wedge \underline{aux} = \prod_{h=1}^{i} A(h) \}$ **A.A.** $\rightarrow$

$\{ 0 \le i \le n \wedge \underline{s + aux} = \sum_{k=1}^{\underline{i}} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h) \} \rightarrow$

s := s + aux;

$\{ 0 \le i \le n \wedge \underline{s} = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h) \} \rightarrow$

TERMINACIÓN

(a) $\mathbf{INV \wedge B \rightarrow E}$ $\{ 0 \le i < n \wedge s = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h) \}$

$\rightarrow$ **n-i ≥ 0** $\in N \rightarrow$ **n-i ≡ E**

(b) {INV ∧ B ∧ E=v} I {E<v}

- **(INV ∧ n-i=v) → (n-(i+1) <v)**
- **{ n-(i+1) <v} i:= i+1 {n-i<v}**

ITERACIÓN DOCUMENTADA:

Pre: { n ≥ 1 }

i:= 0; s:= 0; aux:= 1;

{ $0 \le i \le n \wedge s = \sum_{k=1}^{i} (\prod_{h=1}^{k} A(h)) \wedge aux = \prod_{h=1}^{i} A(h)$ **}**

<u>while</u> i<n <u>loop</u>

i := i+1;

{ $0 \le \underline{i \le n} \wedge s = \sum_{k=1}^{\underline{i-1}} (\prod_{h=1}^{k} A(h)) \wedge \underline{aux} = \prod_{h=1}^{\underline{i-1}} A(h)$ **}**

aux := aux * A(i) ;

{ $0 \le i \le n \wedge \underline{s} = \sum_{k=1}^{\underline{i-1}} (\prod_{h=1}^{k} A(h)) \wedge \underline{aux} = \prod_{h=1}^{i} A(h)$ **}**

s := s + aux;

<u>end loop;</u>

Post: $\{ s = \sum_{k=1}^{n} (\prod_{h=1}^{k} A(h)) \}$

3. (2 puntos) Utilizando el método de **Burstall**, diseña un algoritmo iterativo que compute la siguiente función recursiva:

$$f(n) = \begin{cases} \langle\,\rangle & \text{si } n = 0 \\ 0 \bullet f(n-1) & \text{si } n>0 \wedge \text{par}(n) \\ 1 \bullet f(n-1) & \text{si } n>0 \ \text{impar}(n) \end{cases}$$

PRE ≡ { n ≥ 0 }

POST ≡ { s = f(n) }

INVARIANTE (relación de recurrencia):

$f(\underline{5}) = 1 \bullet f(4) = 1 \bullet (0 \bullet f(3)) = (\langle 1 \rangle \ @ \ \langle 0 \rangle) \ @ \ f(3) = \underline{\langle 1, 0 \rangle} \ @ \ f(\underline{3}) =$

$\langle 1, 0 \rangle \ @ \ (1 \bullet f(2)) = \underline{\langle 1, 0, 1 \rangle} \ @ \ f(\underline{2}) = \langle 1, 0, 1 \rangle \ @ \ (0 \bullet f(1)) =$

$= \underline{\langle 1, 0, 1, 0 \rangle} \ @ \ f(\underline{1}) = \langle 1, 0, 1, 0 \rangle \ @ \ (1 \bullet f(0)) = \underline{\langle 1, 0, 1, 0, 1 \rangle} \ @ \ f(\underline{0}) =$

$\underline{\langle 1, 0, 1, 0, 1 \rangle} \ @ \ \langle\,\rangle = \langle 1, 0, 1, 0, 1 \rangle \Rightarrow$ **INV ≡ { f(n)= s @ f(m) }**

INICIALIZACIÓN:

Pre ≡ { n ≥ 0 }

s:= ⟨ ⟩;

m:= n;

f(n)= s @ f(m) = ⟨ ⟩ @ f(n) = f(n)

FINALIZACIÓN:

¬B ≡ (m = 0)

f(n)= s @ f(m) = s @ f(0) = s @ ⟨ ⟩ = s ⇒ s □se l resultado

{s = f(n) } ≡ Post

CUERPO DE LA ITERACIÓN (desplegado/plegado):

B ≡ (m > 0)

if par(m) then

{ f(n)= s @ f(m) = s @ (⟨0⟩ • f(m-1)) = (s @ ⟨0⟩) @ f(m-1)
s' m'

s:= s@ ⟨0⟩;
m:= m-1;

else

{ f(n)= s @ f(m) = s @ (⟨1⟩ • f(m-1)) = (s @ ⟨1⟩) @ f(m-1)
s' m'

s:= s@ ⟨1⟩;
m:= m-1;

end if;

Como **m:= m-1;** está en las dos partes del condicional, mejoraremos el diseño en el siguiente apartado.

ITERACIÓN DOCUMENTADA:

function f_it(n: integer) return s: secuencia(integer) is

Pre ≡ { n ≥ 0 }
s:= ⟨ ⟩;
m:= n;
{ f(n)= s @ f(m) } ≡ INV

while m> 0 loop

if par(m) then

{ f(n)= s @ f(m) = s @ (⟨0⟩ • f(m-1)) = (s @ ⟨0⟩) @ f(m-1)
s' m'

s:= s@ ⟨0⟩;

else

{ f(n)= s @ f(m) = s @ (⟨1⟩ • f(m-1)) = (s @ ⟨1⟩) @ f(m-1)
s' m'

s:= s@ ⟨1⟩;

end if;

m:= m-1;

end loop;

Post ≡ {s = f(n) }

4. **(1'5 puntos) Verifica** la corrección de la siguiente función recursiva que calcula el número combinatorio:

```
function num_combinatorio (n, k: natural) return c: natural is
Pre: {1 ≤ k ≤ n}
   if k = n then c := 1;
   else  c := num_combinatorio(n – 1, k );
         c := c * n ;
         c := c/(n-k);
   end if
Post: {c = n! / k! (n – k)!}
```

CASO(S) SIMPLE(S)

1. $(1 \leq k \leq n \wedge k = n) \rightarrow (k = n \wedge 1 = n! / n! (n-n)!) \rightarrow (1 = n! / k! (n-k)!)$

2. {1 = n! / k! (n-k)!} c := 1; {c = n! / k! (n – k)!} A. A. ≡ Post

CASO(S) INDUCTIVO(S)

Hipótesis de inducción:

{1 ≤ k ≤ n-1} c := num_combinatorio(n – 1, k); {c = (n-1)! / k! (n - k - 1)!}

1. $(1 \leq k < n) \rightarrow$ (1 ≤ k ≤ n-1)

2. { 1 ≤ k ≤ n-1} c := num_combinatorio(n – 1, k); {c = (n-1)! / k! (n – k - 1)!} h.i.

3. $(c = (n-1)! / k! (n - k - 1)!) \rightarrow$ (c * n = n! / k! (n - k - 1)!)

4. { c * n = n! / k! (n - k - 1)!} c := c * n ; { c = n! / k! (n - k - 1)!} A.A.

5. $(c = n! / k! (n - k - 1)!) \rightarrow$ (c / (n-k) = n! / k! (n - k)!)

6. { c / (n-k) = n! / k! (n - k)! } c := c/(n-k); { c = n! / k! (n - k)! } ≡ Post A.A.

VALIDACIÓN DE LA INDUCCIÓN:

Expresión cota = n - k

Caso simple

- $(k=n) \rightarrow (n-k=0) \in \mathbf{N}$

Caso inductivo

- $(1 \leq k < n) \rightarrow (n-k>0) \in \mathbf{N}$
- num_combinatorio (n, k) llama a num_combinatorio (n-1, k)
  - Se cumple que $n-k>0 \rightarrow n-k-1 \geq 0 \rightarrow n-k-1 \in \mathbf{N}$
  - **n-k-1 < n-k**

5. (**2 puntos**) Especifica ecuacionalmente las siguientes dos funciones:

(a) Función que decide si un elemento está en un árbol binario

sobre: arbin(T)
usa: bool

operación

esta: arbin(T) x T $\rightarrow$ bool

ecuaciones (usando en los dos parámetros las constructoras)

esta(vacio, t) = false

$$\text{esta(crear(r, ai, ad), t)} = \begin{cases} \text{true} & \text{si } r=t \\ \text{esta(ai, t)} \vee \text{esta(ad, t)} & \text{e.o.c} \end{cases}$$

(b) Función *estaS: T × secuencia(arbin(T))→ bool* que decide si un elemento se encuentra en algún árbol de la secuencia.

sobre: secuencia(arbin(T))
usa: bool

operación

estaS: secuencia(arbin(T)) x T $\rightarrow$ bool

ecuaciones

estaS(< >, t) = false

estaS(a•S, t) = esta(a, t) $\vee$ estaS(S, t)