METODOLOGÍA DE LA PROGRAMACIÓN
Grado en Ingeniería Informática de Gestión y Sistemas de Información
Escuela de Ingeniería de Bilbao (UPV / EHU)
Departamento de Lenguajes y Sistemas Informáticos
Curso 1º -- Curso académico 2018-19
Tema 2 – Documentación – Grupo 01
2 puntos
Solución del parcial del 14 de febrero de 2019

<u>Ejercicio 1</u> (1,100 puntos)

a) **mayor_igual(G(1..r), H(1..r), x)** $\equiv \forall k\ ((1 \le k \le r \wedge G(k) = 1) \rightarrow H(k) \ge x)$

b) **bits(I(1..r))** $\equiv \forall k\ (1 \le k \le r \rightarrow (I(k) = 0 \vee I(k) = 1))$

c) **division(D(1..r), $(d_1, d_2, \ldots, d_r)$, E(1..r), $(e_1, e_2, \ldots, e_r)$, F(1..r), $(f_1, f_2, \ldots, f_r)$, pos)** $\equiv$

$(0 \le pos \le r) \wedge$
$\forall k\ (1 \le k \le pos \rightarrow$
$((f_k = 1 \rightarrow (D(k) = d_k / e_k \wedge E(k) = (((d_k / e_k) + 1) * e_k) - d_k \wedge F(k) = f_k))$
$\wedge$
$(f_k \ne 1 \rightarrow (D(k) = d_k \wedge E(k) = e_k \wedge F(k) = f_k))$
$))$

Otra posibilidad:

$(0 \le pos \le r) \wedge$
$\forall k\ ((1 \le k \le pos \wedge f_k = 1) \rightarrow$
$(D(k) = d_k / e_k \wedge E(k) = (((d_k / e_k) + 1) * e_k) - d_k \wedge F(k) = f_k))$
$\wedge$
$\forall k\ ((1 \le k \le pos \wedge f_k \ne 1) \rightarrow (D(k) = d_k \wedge E(k) = e_k \wedge F(k) = f_k))$

Otra posibilidad: (utilizando disyunción)

$(0 \le pos \le r) \wedge$
$\forall k\ (1 \le k \le pos \rightarrow$
$((f_k = 1 \wedge D(k) = d_k / e_k \wedge E(k) = (((d_k / e_k) + 1) * e_k) - d_k \wedge F(k) = f_k)$
$\vee$
$(f_k \ne 1 \wedge D(k) = d_k \wedge E(k) = e_k \wedge F(k) = f_k)))$

d) Las aserciones se darán en el orden en el que es más natural formularlas:

(1) {Precondición} $\equiv$ {$n \ge 1 \wedge$ bits(C(1..n)) $\wedge$ mayor_igual(C(1..n), A(1..n), 0) $\wedge$ mayor_igual(C(1..n), B(1..n), 1) $\wedge$
$\forall k\ (1 \le k \le n \rightarrow (A(k) = a_k \wedge B(k) = b_k \wedge C(k) = c_k))$}

En la precondición se indica que los vectores A, B y C tendrán por lo menos un elemento, C(1..n) solo contiene ceros y unos, y posición a posición, siempre que en C(1..n) se tenga un 1, en A(1..n) se tiene un valor mayor o igual que 0 y en B(1..n) se tiene un valor mayor o igual que 1. Además, se indica que representaremos los valores iniciales de A, B y C mediante *a*'s, b's y c's con los correspondientes subíndices.

(11) {Postcondición} ≡
{division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), n)}

En la postcondición se indica que se han hecho todas las operaciones hasta la posición n incluida, es decir, se han recorrido los vectores y se han hecho todos los cambios necesarios.

(2) {Aserción intermedia} ≡ {(1) $\wedge$ i = 1}
Tras inicializar la variable i con el valor 1, en el punto (2) se cumple todo lo que se ha dicho en el punto (1) y, además, el valor de i es 1.

(3) {Invariante} ≡
{$(1 \le i \le n + 1) \wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)}

En el invariante se indica que i siempre tendrá un valor comprendido entre 1 y n + 1 y que se han hecho todos los cambios hasta la posición i – 1 incluida, es decir, estamos situados en la posición i pero todavía no se ha realizado lo que corresponde a la posición i.

(4) {Aserción intermedia} ≡
{$(1 \le i \le n) \wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)}

Por haber entrado en el while, sabemos que se ha cumplido la condición del while y que i no es n + 1.

(5) {Aserción intermedia} ≡
{$(1 \le i \le n) \wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)
$\wedge$ C(i) = 1 $\wedge$ A(i) = a_i $\wedge$ B(i) = b_i $\wedge$ C(i) = c_i}

Por haber entrado en la rama **then** del **if** sabemos que C(i) es igual a 1. Además, podemos asegurar que en A(i), B(i) y C(i) se tienen los valores iniciales a_i, b_i y c_i. Indicamos esto último porque ahora empieza el proceso de modificación de A(i), B(i) y C(i) y conviene decir cuál es el valor de A(i), B(i) y C(i) tras cada paso de modificación.

El punto (5) se puede abreviar como sigue:

(5) ≡ {(4) $\wedge$ C(i) = 1 $\wedge$ A(i) = a_i $\wedge$ B(i) = b_i $\wedge$ C(i) = c_i}

(6) {Aserción intermedia} ≡
{$(1 \le i \le n) \wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)

$\wedge$ C(i) = 1 $\wedge$ A(i) = a_i $\wedge$ B(i) = b_i $\wedge$ C(i) = c_i $\wedge$ aux = A(i) / B(i)}

Tras ejecutarse la asignación aux : = A(i) / B(i); sabemos que ahora el valor de aux es igual a A(i) / B(i). En en A(i), B(i) y C(i) se conservan los valores iniciales a_i, b_i y c_i. De momento los cambios completos están hechos hasta la posición i – 1 y en la posición i estamos a medias, por lo que en el predicado *division* seguimos poniendo i – 1.

El punto (6) se puede abreviar como sigue:

(6) ≡ {(5) $\wedge$ aux = A(i) / B(i)}

(7) {Aserción intermedia} ≡

{(1 ≤ i ≤ n) $\wedge$

division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)

$\wedge$ C(i) = 1 $\wedge$ A(i) = a_i $\wedge$ B(i) = ((aux + 1) * b_i) – A(i) $\wedge$ C(i) = c_i $\wedge$ aux = A(i) / b_i }

Tras ejecutarse la asignación B(i) : = ((aux + 1) * B(i)) – A(i); sabemos que ahora el valor de B(i) es igual a ((aux + 1) * b_i) – A(i) y el valor de aux es igual a A(i) / b_i. En A(i) se conserva el valor inicial a_i y en C(i) se conserva el valor inicial c_i. De momento los cambios completos están hechos hasta la posición i – 1 y en la posición i estamos a medias, por lo que en el predicado *division* seguimos poniendo i – 1.

El punto (7) se puede abreviar como sigue:

(7) ≡ {(4) $\wedge$ C(i) = 1 $\wedge$ A(i) = a_i $\wedge$ B(i) = ((aux + 1) * b_i) – A(i) $\wedge$ C(i) = c_i $\wedge$ aux = A(i) / b_i }

(8) {Aserción intermedia} ≡

{(1 ≤ i ≤ n) $\wedge$

division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)

$\wedge$ C(i) = 1 $\wedge$ A(i) = aux $\wedge$ B(i) = ((aux + 1) * b_i) – a_i $\wedge$ C(i) = c_i $\wedge$ aux = a_i / b_i
}

Tras ejecutarse la asignación A(i) : = aux; sabemos que ahora el valor de A(i) es igual a aux el valor de B(i) es igual a ((aux + 1) * b_i) – a_i, el valor de C(i) es c_i y el valor de aux es a_i / b_i.

El punto (8) se puede abreviar como sigue:

(8) ≡ {(4) $\wedge$ C(i) = 1 $\wedge$ A(i) = aux $\wedge$ B(i) = ((aux + 1) * b_i) – a_i $\wedge$ C(i) = c_i $\wedge$ aux = a_i / b_i }

Ahora los cambios completos están hechos hasta la posición i incluida por lo que en el predicado *division* podemos poner i. Por consiguiente, el punto (8) se puede reformular como sigue:

(8) ≡

{(1 ≤ i ≤ n) $\wedge$

division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i)

$\wedge$ C(i) = 1 $\wedge$ aux = a_i / b_i }

Al poner i en el predicado ya no hace falta poner A(i) = aux $\wedge$ B(i) = ((aux + 1) * b_i) – a_i $\wedge$ C(i) = c_i porque eso ya está dicho en el predicado *division*.

(9) {Aserción intermedia} ≡
{(1 ≤ i ≤ n) $\wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i)}

La diferencia entre (8) y (9) es que en (8) sabemos que se ha ido por la rama **then** y, por tanto, podemos asegurar que se cumple C(i) = 1 $\wedge$ aux = a_i / b_i, pero en (9) no sabemos si se ha ido por la rama **then** o no y, por ello, no podemos asegurar que se cumpla C(i) = 1 $\wedge$ aux = a_i / b_i, ya que si no se ha cumplido la condición del **if**, la fórmula C(i) = 1 $\wedge$ aux = a_i / b_i no será cierta. Pero bien yendo por la rama **then** o bien saltando la rama **then**, ahora ya están hechos todos los cambios correspondientes a la posición i y por ello en el predicado *division* ponemos i.

(10) {Aserción intermedia} ≡
{(2 ≤ i ≤ n + 1) $\wedge$
division(A(1..n), (a_1, a_2, …, a_n), B(1..n), (b_1, b_2, …, b_n), C(1..n), (c_1, c_2, …, c_n), i – 1)}

Al incrementar en uno el valor de i, los límites del intervalo crecen también en uno y, además, en el predicado hay que volver a poner i – 1.

(12) E = n + 1 – i

La expresión cota nos indica, siempre que estemos en el punto donde se cumple el invariante, cuántas vueltas quedan por dar como máximo. Como el vector se recorre de izquierda a derecha, E es "el último valor que tomará i" menos "i". Por consiguiente, en este caso la expresión E es la distancia entre n + 1 y la variable i. Cuando i crece, la distancia disminuye y el número de vueltas pendientes decrece. En este programa E nos indica cuántas vueltas quedan exactamente.

Las partes coloreadas resaltan los cambios que hay de una aserción a otra y los aspectos más relevantes de cada fórmula.

<u>Ejercicio 2</u> (0,900 puntos)

a) **multiplo(x, y)** $\equiv$ {x mod y = 0}

b) Las aserciones se darán en el orden en el que es más natural formularlas:

(1) {Precondición} $\equiv$ $\{p \geq 1 \wedge q \geq p + 1 \wedge v \geq q + 1 \wedge w \geq v\}$

En la precondición se indican las características de los cuatro datos de entrada: p, q, v y w.

(11) {Postcondición} $\equiv$ $\{mc = N\ k\ (v \leq k \leq w \wedge multiplo(k, p) \wedge multiplo(k, q))\}$

En la postcondición se indica que en la variable *mc* se tienen contabilizados todos los números k que pertenecen al intervalo [v..w] y son múltiplos tanto de p como de q.

(2) {Aserción intermedia} $\equiv$ $\{(1) \wedge h = v - 1\}$

(3) {Aserción intermedia} $\equiv$ $\{(2) \wedge mc = 0\}$

(4) {Invariante} $\equiv$ $\{ (v - 1 \leq h \leq w) \wedge$
$mc = N\ k\ (v \leq k \leq h \wedge multiplo(k, p) \wedge multiplo(k, q))\}$

En el invariante se indica que h siempre tendrá un valor comprendido entre v – 1 y w y que en la variable *mc* se tienen contabilizados todos los números k que pertenecen al intervalo [v..h] y son múltiplos tanto de p como de q.

(5) {Aserción intermedia} $\equiv$ $\{ (v - 1 \leq h \leq w - 1) \wedge$
$mc = N\ k\ (v \leq k \leq h \wedge multiplo(k, p) \wedge multiplo(k, q))\}$

Por haber entrado en el while sabemos que se ha cumplido la condición del while y que h es menor o igual que w – 1.

(6) {Aserción intermedia} $\equiv$ $\{ (v - 1 \leq h \leq w - 1) \wedge$
$mc = N\ k\ (v \leq k \leq h \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h + 1, p) \wedge multiplo(h + 1, q)\}$

Tras asignar el valor *(((h + 1)* ***mod*** *p) = 0)* ***<u>and</u>*** *(((h + 1)* ***mod*** *q) = 0)* a la variable aux, el valor de la variable aux coincide con el valor de la fórmula *multiplo(h + 1, p)* $\wedge$ *multiplo(h + 1, q)*.

(7) {Aserción intermedia} $\equiv$ $\{ (v - 1 \leq h \leq w - 1) \wedge$
$mc = N\ k\ (v \leq k \leq h \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h + 1, p) \wedge multiplo(h + 1, q) \wedge$
$aux = True\}$

Por haber entrado en la rama then del if, sabemos que el valor de la variable aux es True. En vez de poner aux = True se puede poner también solo aux.

{Aserción intermedia} ≡ { $(v - 1 \le h \le w - 1) \wedge$
$mc = N\, k\, (v \le k \le h \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h + 1, p) \wedge multiplo(h + 1, q) \wedge$
aux}

(8) {Aserción intermedia} ≡ { $(v - 1 \le h \le w - 1) \wedge$
$mc = N\, k\, (v \le k \le h + 1 \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h + 1, p) \wedge multiplo(h + 1, q) \wedge$
aux}

Tras sumar 1 a la variable mc, en *mc* se tienen contabilizados todos los números k que pertenecen al intervalo [v..h + 1] y son múltiplos tanto de p como de q.

(9) {Aserción intermedia} ≡ { $(v - 1 \le h \le w - 1) \wedge$
$mc = N\, k\, (v \le k \le h + 1 \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h + 1, p) \wedge multiplo(h + 1, q)$}

La diferencia entre los puntos (8) y (9) es que en el (8) se sabe que se ha entrado en la rama **then** y, por tanto, se cumple que el valor de la variable aux es True, mientras que en el punto (9) no se sabe si se ha entrado en la rama **then** y, por consiguiente, no se sabe si el valor de aux es True o False. Pero, en cualquier caso, en *mc* se tienen contabilizados todos los números k que pertenecen al intervalo [v..h + 1] y son múltiplos tanto de p como de q.

(10) {Aserción intermedia} ≡ { $(v \le h \le w) \wedge$
$mc = N\, k\, (v \le k \le h \wedge multiplo(k, p) \wedge multiplo(k, q)) \wedge$
$aux \leftrightarrow multiplo(h, p) \wedge multiplo(h, q)$}

Tras incrementarse el valor de h, cambian los límites de su intervalo y el significado de mc y aux.

(12) $E = w - h$

La expresión cota nos indica, siempre que estemos en el punto donde se cumple el invariante, cuántas vueltas quedan por dar como máximo: E es "el último valor que tomará h" menos "h". La expresión E es al fin y al cabo la distancia entre w y la variable h. Cuando h crece, la distancia disminuye y el número de vueltas pendientes decrece. También en este caso E nos indica cuántas vueltas quedan exactamente.