UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I
Primer parcial – Segundo cuatrimestre de 2021

Ejercicio 1. El funcionamiento del siguiente programa se basa en el hecho de que la suma de los primeros n números naturales verifica la igualdad $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$:

$P_c : \{n \geq 0 \wedge s = (n * (n + 1)) \text{ div } 2 \wedge t = n\}$

```
while (s > 0) do
  s := s - t;
  t := t - 1
endwhile
```

$Q_c : \{s = 0 \wedge t = 0\}$

proponer un invariante I para el ciclo y demostrar que se cumplen los siguientes puntos del teorema del invariante:

a) $(I \wedge \neg B) \Rightarrow Q_c$

b) $\{I \wedge B\}$ ⟨cuerpo del ciclo⟩ $\{I\}$

Resolución:
Obviamentemente lo primero es definir el invariante. Recordemos que es una fórmula que debe ser cierta: justo antes que empiece el ciclo, al comienzo y al final de cada iteración del ciclo y justo después que el ciclo termina. Además tiene que ser útil para demostrar el teorema del invariante.
En este caso la heurística de mirar la postcondición no sirve ¿Porqué?
Esa heurística está fundada en la idea en que al final el invariante se convierte en la post.
Aquí, habría que usar la idea dual, i.e. partir de la pre. Por eso se propondría:

$$I \equiv s = (t * (t + 1)) \text{ div } 2 \wedge t \geq 0$$

Es importante remarcar que es posible que la versión completa del invariante no les haya surgido desde el comienzo. Por ejemplo que hayan incluido $s \geq 0$. Pero al hacer las pruebas se hubieran dado cuenta que era innecesario porque por definición si $t \geq 0$ s debe ser mayor que cero.

Obviamente $P_c \longrightarrow I$ aunque eso no se pide que sea verificado.

Ahora probemos lo pedido:

a) $(I \wedge \neg B) \Rightarrow Q_c$. Nota que $\neg B \equiv s \leq 0$ y junto con I tenemos que $s = 0$. Entonces por el invariante tenemos $0 = (t * (t + 1)) \text{ div } 2 \wedge t \geq 0$. Así que la única solución posible para la ecuación $0 = (t * (t + 1)) \text{ div } 2$ es $t = 0$.

b) $\{I \wedge B\}$ ⟨cuerpo del ciclo⟩ $\{I\}$. Como siempre para ver que esa tripla de Hoare es válida calculamos la wp de cuerpo del ciclo respecto de I, i.e. $wp(s := s - t, wp(t := t - 1, I))$. Lo cual es relativamente simple porque corresponde a la aplicación del axioma 1 en cada caso.
$wp(t := t - 1, I) = def(t - 1) \wedge_L \{I\}^t_{t-1}) \equiv$
$\equiv True \wedge_L s = ((t - 1) * ((t - 1) + 1)) \text{ div } 2 \wedge (t - 1) \geq 0 \equiv$
$\equiv s = ((t - 1) * t) \text{ div } 2 \wedge t \geq 1 \equiv E_2$
Entonces calculemos $wp(s := s - t, E_2)$ aplicando el axioma 1 nuevamente:
$wp(s := s - t, E_2) = def(s - t) \wedge_L \{E_2\}^s_{s-t}) \equiv$
$\equiv True \wedge_L s - t = ((t - 1) * t) \text{ div } 2 \wedge t \geq 1 \equiv$
$\equiv s - t = ((t - 1) * t) \text{ div } 2 \wedge t \geq 1 \equiv E_1$

Veamos ahora que $I \wedge B$ fuerza la wp:

$$(s = (t * (t + 1)) \text{ div } 2 \wedge t \geq 0) \wedge s > 0 \Longrightarrow s - t = ((t - 1) * t) \text{ div } 2 \wedge t \geq 1$$

Notar que $s - t = ((t - 1) * t) \text{ div } 2 \equiv s = ((t - 1) * t) \text{ div } 2 + t \equiv (t * (t + 1)) \text{ div } 2$.
Notar que $s > 0$ implica $(t * (t + 1)) \text{ div } 2 > 0$, lo cual implica $t \geq 1$.

Ejercicio 2. El método de compresión por *run-length encoding* se basa en representar una **palabra** (secuencia de caracteres) a través de un **código**. Un código es una lista de pares, cada uno de los cuales contiene un fragmento de la palabra, acompañado del número de veces que dicho fragmento debe repetirse. Por ejemplo, la palabra `"axaxaxblablacaxax"` se puede representar con el código `[("ax",3),("bla",2),("c",1),("ax",2)]`. En general puede haber varios códigos para una misma palabra. Por ejemplo, la palabra `"banana"` se puede representar con los cuatro códigos siguientes, entre otros:

```
     [("banana",1)]  [("b",1),("an",2),("a",1)]
[("ba",1),("na",2)]  [("b",1),("a",1),("n",1),("a",1),("n",1),("a",1)]
```

Definimos el tipo Palabra como un renombre de $seq\langle \text{Char} \rangle$ y el tipo Código como un renombre de $seq\langle \text{Palabra} \times \mathbb{Z} \rangle$. Se pide:

a) Especificar el predicado `pred esDescompresión(`*cod* : `Código, ` *pal* : `Palabra)` que es verdadero si *pal* es la palabra que resulta de descomprimir el código *cod*.

b) Especificar el problema `proc comprimir(in` *pal* : `Palabra, out` *cod* : `Código)` que dada una palabra devuelve algún código que la representa. El código resultante no debe contener pares que contengan fragmentos vacíos (ej. `("",3)`) ni repeticiones nulas (ej. `("ax",0)`).

c) Especificar el problema `proc optimizarCódigo(inout` *cod* : `Código)` que, dado un código, lo modifica para que siga representando la misma palabra pero de tal modo que el código modificado sea **óptimo**. Un código c es óptimo cuando cualquier otro código que represente la misma palabra cuesta *al menos* lo mismo que cuesta c. El costo se computa de acuerdo con algún criterio (irrelevante a los efectos de este ejercicio). Se puede suponer ya definida una función auxiliar que determina el costo de un código dado, que es siempre un número entero positivo:

`aux costo(`*cod* : `Código)` : $\mathbb{Z}$

Resolución:

Este problema se puede resolver de muchas maneras distintas. Aquí se expondrá sólo una de ellas.

a) Veamos primero algunas ideas para abordar la especificación del predicado `esDescompresión`. Una primera observación es que la dimensión de la codificación corresponde a la cantidad de partes "distintas" en que divides a la palabra y cada elemento de la codificación (una tupla) corresponde a partes que se "repiten". Usemos esta consideración para hacer la especificación.

`pred esDescompresión(`*cod* : `Código, ` *pal* : `Palabra)` {
$|pal| = dim(cod) \wedge_L$
$(\forall i : \mathbb{Z})(0 \leq i < |cod| \rightarrow_L codSubsec(cod[i], subseq(pal, dimHasta(cod, i), dimHasta(cod, i+1)))$
}

Notar que es importante el chequeo de que el tamaño de la palabra y de la codificación son iguales para evitar la indefinición del resto.

`pred codSubsec(`c : `Palabra` $\times \mathbb{Z}$, p : `Palabra)` {
$c_o \neq$ `""` $\wedge_L\ c_1 > 0 \wedge_L (\forall j : \mathbb{Z})(0 \leq j < c_1 \rightarrow_L c_0 = subseq(p, |c_0| * j, |c_0| * (j+1)))$
}

`aux dim(`c : `Código)` : $\mathbb{Z} = \Sigma_{i=0}^{|c|-1} |c[i]_0| * c[i]_1$

`aux dimHasta(`c : `Código`, $d : \mathbb{Z}$) : $\mathbb{Z} = \Sigma_{i=0}^{d-1} |c[i]_0| * c[i]_1$

b)

`proc comprimir` (`in` pal : Palabra, `out` cod : Código) {
`Pre` {$0 <$ |pal|}
`Post` {`esDescompresión`(pal, cod)}
}

c)

`proc optimizarCódigo` (`inout` cod : Código) {
`Pre` {$0 <$ |cod| $\wedge_L$ cod $= Cod_0$}
`Post` {$(\exists$ pal : Palabra$)$(`esDescompresión`(pal, cod) $\wedge$ `esDescompresión`(pal, Cod_0) $\wedge$
$(\forall c$: Código$)$(`esDescompresión`(pal, c) $\rightarrow_L$ `costo`(cod) $\leq$ `costo`(c))}
}