UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I
Recuperatorio del primer parcial– 28/06/2021

Ejercicio 1. Dado el siguiente ciclo con sus correspondientes pre y postcondición:

$P_c : \{|s| = 2 * |t| \wedge i = 0\}$

```
while (i < s.size()) do
  t[i div 2] := s[i] + s[i + 1];
  i := i + 2
endwhile
```

$Q_c : \{|s| = 2 * |t| \wedge_L (\forall k : \mathbb{Z})(0 \leq k < (|s| \texttt{ div } 2) \rightarrow_L t[k] = s[2 * k] + s[2 * k + 1])\}$

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

a) $P_c \Rightarrow I$

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

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

Recordar que $(n \texttt{ div } m)$ *denota la división entera entre* n *y* m*; por ejemplo* $7 \texttt{ div } 2 = 3$.

Ejercicio 2. En este ejercicio llamamos *permutación* a una lista de n números enteros en la que cada número entre 0 y $n-1$ aparece exactamente una vez. Por ejemplo, $[0, 1, 2, 3, 4]$, $[4, 3, 2, 1, 0]$ y $[3, 4, 2, 1, 0]$ son permutaciones de 5 elementos.

Dada una lista de n elementos, llamamos *intercambio* a una tupla $\langle i, j \rangle$ donde i y j son índices de la lista. Un intercambio representa la acción de intercambiar de lugar los elementos de la lista que se encuentran en las posiciones indicadas. Por ejemplo, si tomamos la lista $[3, 4, 2, 1, 0]$ y aplicamos el intercambio $\langle 0, 1 \rangle$ obtenemos la lista $[4, 3, 2, 1, 0]$. Si $i = j$, el intercambio no modifica la lista.

Una permutación de n elementos siempre se puede ordenar aplicando una secuencia de intercambios, por ejemplo:

$$[3, 4, 2, 1, 0] \xrightarrow{\langle 0,4 \rangle} [0, 4, 2, 1, 3] \xrightarrow{\langle 1,3 \rangle} [0, 1, 2, 4, 3] \xrightarrow{\langle 3,4 \rangle} [0, 1, 2, 3, 4]$$

En un caso como este decimos que, a partir de la permutación inicial $[3, 4, 2, 1, 0]$, la lista de intercambios $[\langle 0, 4 \rangle, \langle 1, 3 \rangle, \langle 3, 4 \rangle]$ **ordena** la permutación $[3, 4, 2, 1, 0]$ y **genera** la siguiente cadena:

$$[[3, 4, 2, 1, 0], [0, 4, 2, 1, 3], [0, 1, 2, 4, 3], [0, 1, 2, 3, 4]]$$

Observar que una lista de m intercambios genera una cadena de $m + 1$ elementos. Se pide:

a) Especificar el predicado `pred esPerm(s :` $seq\langle\mathbb{Z}\rangle$`)` que indica si una secuencia es o no una permutación.

b) Especificar el predicado `pred genera(cadena :` $seq\langle seq\langle\mathbb{Z}\rangle\rangle$`, is :` $seq\langle\mathbb{Z} \times \mathbb{Z}\rangle$`)` que determina si la secuencia de secuencias `cadena` tiene como primer elemento una permutación, y el resto está generado por la secuencia de intercambios `is`, tal como se ilustra en el ejemplo de arriba.
Importante: Este predicado no debe indefinirse para ninguna combinación de entradas.

c) Especificar el problema que recibe como entrada una permutación y devuelve como salida alguna lista de intercambios que la ordena. Puede asumir un predicado `pred ordenada?(s :` $seq\langle\mathbb{Z}\rangle$`)` que indica si una secuencia está ordenada.
Nota: la lista de intercambios que ordena una permutación no es única, pero se puede suponer sin demostrarlo que siempre existe.

Ejercicio 3. Dados el siguiente programa S en SmallLang y la siguiente especificación:

```
if (i mod 2 == 0) then
  i := 2 * m[i div 2]
else
  i := 2 * m[i div 2] + x
endif
```

```
proc calcularSiguienteMúltiplo (in m: seq⟨ℤ⟩, in x: ℤ, inout i: ℤ) {
      Pre {i = i0 ∧ 1 ≤ i < |m| ∧ (∀k : ℤ)(0 ≤ k < |m| ⟶L m[k] = k * x)}
      Post {i = i0 * x}
}
```

a) Calcular la precondición más débil del programa S con respecto a la postcondición de la especificación: *wp(S; Post).*

b) Demostrar que el programa es correcto con respecto a la especificación.

Ejercicio 4.

Dados el siguiente programa y su especificación:

```
    void ordenar(vector<int>>& s) {
L1      int i = 0;
L2      while (i < s.size()) {
L3          if (i == 0) {
L4              i = i + 1
            }
L5          else if (s[i] >= s[i - 1])) {
L6              i = i + 1
            }
L7          else {
L8              s[i] = s[i - 1];
L9              s[i - 1] = s[i];
L10             i = i - 1;
            }
        }
L11     return;
    }
```

```
proc ordenar (inout s: seq⟨ℤ⟩) {
      Pre {
        s = S_0
      }
      Post {
        (∀x ∈ ℤ)(cuenta(x, s) = cuenta(x, S_0)) ∧
        (∀i, j ∈ ℤ)(0 ≤ i ≤ j < |s| ⟶_L s[i] ≤ s[j])
      }
      aux cuenta (x: ℤ, xs: seq⟨ℤ⟩) : ℤ =
        Σ_{i=0}^{|xs|-1} if x == xs[i] then 1 else 0 fi;
}
```

Importante: Cada caso de test propuesto debe contener la entrada y el resultado esperado.

a) Describir el diagrama de control de flujo (*control-flow graph*) del programa.

b) Escribir un conjunto de casos de test (o *test suite*) que cubra todas las sentencias. Mostrar qué líneas cubre cada test. Este conjunto de tests ¿cubre todas las decisiones? (Justificar).

c) Escribir un *test* que encuentre el defecto presente en el código (una entrada que cumple la precondición pero tal que el resultado de ejecutar el código no cumple la postcondición).

d) ¿Es posible escribir para este programa un *test suite* que cubra todas las decisiones pero que **no encuentre** el defecto en el código? En caso afirmativo, escribir el test suite; en caso negativo, justificarlo.