UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructuras de Datos I
Primer parcial – Primer cuatrimestre de 2022

1a	1b	2a	2b	3a	3b	4
10	25	15	12	5	10	15

Apellido Pagés Nombre Julieta Belén
LU 1691/21 Número de comisión 5
Cantidad de hojas entregadas ....

El parcial se aprueba con 70 puntos.

A

Ejercicio 1. [35 puntos] Dado el siguiente programa:

$P_c : \{i = |s| - 1 \land |t| = 0\}$

```
while (i >= 0) do
  t := ⟨s[i]⟩ ++ t ++ ⟨s[|s| - 1 - i]⟩;
  i := i - 1
endwhile
```

$Q_c : \{t = s \text{ ++ } s\}$

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

a) [10 puntos] $(I \land \neg B) \Rightarrow Q_c$

b) [25 puntos] $\{I \land B\}$ ⟨cuerpo del ciclo⟩ $\{I\}$

Recordar que la función aux subseq$(s : seq\langle T\rangle, i : \mathbb{Z}, j : \mathbb{Z}) : seq\langle T\rangle$ está definida si y sólo si $0 \leq i \leq |s|$ y $0 \leq j \leq |s|$, y denota la sublista de s comprendida entre las posiciones i (inclusive) y j (exclusive). Se puede usar sin demostrar el hecho de que si $0 \leq i \leq j \leq k \leq |s|$ entonces $\text{subseq}(s, i, k) = \text{subseq}(s, i, j) \text{ ++ } \text{subseq}(s, j, k)$.

Ejercicio 2. [35 puntos] En este ejercicio llamamos *intervalo* a un par (a, b) de números reales $\mathbb{R} \times \mathbb{R}$ tales que $a < b$. Decimos que un intervalo (a, b) *cubre* aquellos puntos que se encuentran entre a y b inclusive[1]. Por ejemplo, el intervalo $(0, 1)$ cubre al $1/2$ y también al 0 y al 1. Dada una secuencia de intervalos $s = [(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)]$ decimos que s cubre un punto p si alguno de los intervalos (a_i, b_i) cubre el punto p.

a) [15 puntos] Especificar un predicado que, dadas dos secuencias de intervalos s y t, es verdadero si y sólo si s y t cubren exactamente los mismos puntos. Por ejemplo, las secuencias $s = [(0, 3), (2, 5)]$ y $t = [(1, 2), (3, 5), (0, 3)]$ cubren exactamente los mismos puntos, ya que tanto s como t cubren un punto p si y sólo si $0 \leq p \leq 5$.

b) [20 puntos] Especificar el problema que recibe una secuencia de intervalos s y devuelve la secuencia s' más corta que cubre exactamente los mismos puntos que s. Además, la secuencia s' debe estar ordenada de menor a mayor (comparando los extremos izquierdos de los sucesivos intervalos). Por ejemplo, si $s = [(2, 4), (0, 1), (3, 5)]$ la respuesta debe ser $s' = [(0, 1), (2, 5)]$. Cuando decimos que s' debe ser "la más corta" nos referimos a que la secuencia tenga la menor longitud posible.

[1] Es decir, el intervalo es cerrado en ambos extremos. En la notación matemática usual este mismo intervalo se nota $[a, b]$.

Ejercicio 3. [15 puntos] Dados el siguiente programa S y la siguiente especificación:

```
if (x < s[0]) then
  s := ⟨x⟩ ++ s
else
  s := s ++ ⟨x⟩
endif
```

```
proc extenderEscalera (inout s: seq⟨ℤ⟩, in x : ℤ) {
      Pre {(esEscalera(s) ∧ |s| > 0)
           ∧_L (x = s[0] − 1 ∨ x = s[|s| − 1] + 1)}
      Post {esEscalera(s)}
}
pred esEscalera (s: seq⟨ℤ⟩) {
    (∃k : ℤ)(∀i : ℤ)(0 ≤ i < |s| ⟶_L s[i] = k + i)
}
```

a) [5 puntos] Calcular la precondición más débil del programa S con respecto a la postcondición: *wp(S; Post)*.

b) [10 puntos] Demostrar que el programa es correcto con respecto a la especificación propuesta.

Ejercicio 4. [15 puntos] El programa que se transcribe más abajo pretende determinar la longitud del *fragmento* más largo en un texto. Los fragmentos son porciones del texto que no contienen punto y coma. Por ejemplo, en el siguiente texto el fragmento más largo es `"Mercurio"`, de ocho letras:

`"Mercurio;Venus;Tierra;Marte;Júpiter"`

La especificación formal es la siguiente:

```
proc calcularFragmentoMásLargo (in s: seq⟨Char⟩, out n : ℤ) {
      Pre {True}
      Post {(∃i, j : ℤ)(esFragmento(s, i, j) ∧ n ≤ j − i) ∧ (∀i, j : ℤ)(esFragmento(s, i, j) ⟶ n ≥ j − i)}
}
pred esFragmento (s: seq⟨Char⟩, i, j : ℤ) {
    0 ≤ i ≤ j ≤ |s| ∧_L (∀k : ℤ)(i ≤ k < j ⟶_L ¬esPuntoYComa(s[k]))
}
pred esPuntoYComa (c: Char) {
    c = ';'
}
```

El programa es el siguiente:

```
int calcularFragmentoMasLargo(vector<char> s) {
  int n = 0;
  int i = 0;
  int f = 0; // Longitud del fragmento actual
  while (i < s.size()) {
      if (f > n) {
          n = f;
      }
      if (s[i] == ';') {
          f = 0;
      } else {
          f++;
      }
      i++;
  }
  return n;
}
```

Escribir un *test* que encuentre el defecto presente en el código. Es decir, escribir una entrada tal que cumple con la precondición pero tal que el resultado de ejecutar el código no cumple la postcondición. (Justificar la respuesta).

Julieta Pagés Comisión 5 | a) 10 | b) 25

① $P_c \equiv \{i = |s|-1 \wedge |T| = 0\}$ $\quad Q_c \equiv T = S ++ S$

$B \equiv i \geq 0$ $\quad \neg B \equiv i < 0$

$I \equiv -1 \leq i \leq |s|-1 \wedge T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i-1)$ ✓

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

$(I \wedge \neg B) \equiv i < 0 \wedge -1 \leq i \leq |s|-1 \wedge T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i-1)$

$\equiv i = -1 \wedge T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i-1)$

$\rightarrow T = subseq(S, -1+1, |s|) ++ subseq(S, 0, |s|-1+1) \rightarrow T = S ++ S \equiv Q_c$

(Por la igualdad "$i=-1$" reemplazo las "i" por "-1") ✓

b) $\{I \wedge B\}$ ⟨cuerpo del ciclo⟩ $\{I\} \leftrightarrow \{I \wedge B\} \rightarrow wp(\text{cuerpo del ciclo}, I)$

$S_1 \equiv T := \langle S[i] \rangle ++ T ++ \langle S[|s|-1-i] \rangle$ $\quad S_2 \equiv i := i-1$

$S \equiv S_1 ; S_2$

Axioma 3

$wp(S, I) \equiv wp(S_1, wp(S_2, I)) \equiv wp(T := \langle S[i] \rangle ++ T ++ \langle S[|s|-1-i] \rangle, wp(i := i-1, I))$

Axioma 1

$\equiv wp(T := \langle S[i] \rangle ++ T ++ \langle S[|s|-1-i] \rangle, -1 \leq i-1 \leq |s|-1 \wedge T = subseq(S, i+1-1, |s|) ++ subseq(S, 0, |s|-(i-1)-1))$

Axioma 1

$\equiv 0 \leq i \leq |s| \wedge \langle S[i] \rangle ++ T ++ \langle S[|s|-i-1] \rangle = subseq(S, i, |s|) ++ subseq(S, 0, |s|-i)$

$\{I \wedge B\} \equiv -1 \leq i \leq |s|-1 \wedge T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i) \wedge i \geq 0$

$\equiv 0 \leq i \leq |s|-1 \wedge T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i)$

Quiero ver que $\{I \wedge B\} \rightarrow wp(S, I)$

✓ $0 \leq i \rightarrow 0 \leq i$ $\quad$ ✓ $i \leq |s|-1 \rightarrow i \leq |s|$

(*) $\equiv T = subseq(S, i+1, |s|) ++ subseq(S, 0,$ (para ... tuve que)

(Sigue atrás)

- $T = subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i) \rightarrow$

$\langle S[i] \rangle ++ \underbrace{subseq(S, i+1, |s|) ++ subseq(S, 0, |s|-i)}_{=T} = subseq(S, i, |s|) ++ subseq(S, 0, |s|-i) ++ \langle S[|s|-i-1] \rangle$

o sea que $\{I \wedge B\} \Rightarrow wp(S, I)$ → ES lo mismo ya que al primer "subseq(S, i+1, |s|)" le concateno antes a $\langle S[i] \rangle$, o sea que empieza desde "i" en vez de desde "i+1". y a la inversa al segundo "subseq(S, 0, |s|-i)" le concateno "$\langle S[|s|-i-1] \rangle$", que es como tener la misma secuencia pero que sí incluye al "$\langle S[|s|-i-1] \rangle$" ✓

Julieta Pagés Comisión 5

a) 15 | b) 12

(2) a) Pred cubrenLoMismo(S: seq<ℝ×ℝ>, T: seq<ℝ×ℝ>){
(∀r:ℝ)(cubre(S, P) <-> cubre(T, P))
}

Pred cubre(S: seq<ℝ×ℝ>, P: ℝ){
(∃i:ℤ)(0≤i<|S| ∧L (S[i])0 ≤ P ≤ (S[i])0)
}

b) Proc ordenar(inout T S: seq<ℝ×ℝ>)
Pre{ S = S0, |S|>0} ⊛ ← Falta pedir que sean pares ℝ×ℝ
Post{(S=S0=⟨⟩) ∨ (|S|>0 ∧L cubrenLoMismo(S, S0) ∧ ordenadaMasCorta(S))}

Aclaración
incluyo el caso de S=⟨⟩ porque el enunciado no dice lo contrario y no es estrictamente necesario para que esté ordenada que tenga elementos. Interpreto "si" comparar. Pero si tenía que interpretar a la lista vacía como lista ordenada, en Post habría que agregar (|S|=0 ∨ (S=S0=⟨⟩ ∨ todo que ya se en mi Post))

⊛ Por el enunciado interpreto que es necesario para lo que pide que haya al menos un elemento para comparar. Pero si tenía que interpretar a la lista vacía como lista ordenada, en Post habría que agregar (S=S0=⟨⟩ ∨ todo que ya se en mi Post)

Pred ordenadaMasCorta(S: seq<ℝ×ℝ>)
ordenada(S) ∧ (∀T: seq<ℝ×ℝ>)((|T|>0 ∧L → para que no se indefina cubrenLoMismo(S,T)) si T=⟨⟩) ∧ ordenada(T)
→ |T| ≥ |S|)
}

Pred ordenada(S: seq<ℝ×ℝ>){
(∀i:ℤ)(0≤i<|S|-1 →L (S[i])0 ≤ (S[i+1])0)
}

Recordá revisar en la pre y post que cumplan propiedades que quieres especificar, como que sean tuplas

Julieta Pagés Comisión 5

③ a) $S1 \equiv S := \langle x \rangle ++ s$ $\quad S2 \equiv S := s ++ \langle x \rangle$

$S \equiv$ if $S1$ else $S2$ endif $\quad B \equiv x < s[0]$

$wp(S, Post) \overset{\text{Axioma 4}}{\equiv} (x < s[0] \wedge S = \langle x \rangle ++ s) \vee (x \geq s[0] \wedge S = s ++ \langle x \rangle)$

b) $\{Pre\}\ S\ \{Post\} \leftrightarrow Pre \rightarrow wp(S)$

~~$Pre \equiv esEscalera(s) \wedge |s| > 0 \equiv (\exists k : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] = k + i)$~~

$Pre \equiv esEscalera(s) \wedge |s| > 0 \wedge_L (x = s[0] - 1 \vee x = s[|s| - 1] + 1)$

~~$\equiv (\exists s : \mathbb{Z})(\forall i : \mathbb{Z}$~~

③ a) $S1 \equiv S := \langle x \rangle ++ s$ $\quad S2 \equiv S := s ++ \langle x \rangle$

$S \equiv$ if $\quad B \equiv x < s[0]$

$wp(S, Post) \equiv (B \wedge wp(S1, Post)) \vee (\neg B \wedge wp(S2, Post))$

$\equiv (x < s[0] \wedge wp(S := \langle x \rangle ++ s, esEscalera(s))) \vee (x \geq s[0] \wedge wp(S := s ++ \langle x \rangle, esEscalera(s)))$

$\equiv (x < s[0] \wedge esEscalera(\langle x \rangle ++ s)) \vee (x \geq s[0] \wedge esEscalera(s ++ \langle x \rangle))$

b) $(\{Pre\}\ S\ \{Post\}) \leftrightarrow (Pre \rightarrow wp(S))$

$Pre \equiv (esEscalera(s) \wedge |s| > 0) \wedge_L (x = s[0] - 1 \vee x = s[|s| - 1] + 1)$

e datos)

* Si $x < s[0]$

$((\exists k : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] = k + i) \wedge |s| > 0) \wedge_L (x = s[0] - 1 \vee x = s[|s| - 1] + 1)$

• Si $X < S[0]$

$$(\exists K \cdot \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L S[i] = K + i) \wedge |s| > 0) \wedge_L \underbrace{(X = S[0] - 1 \vee X = S[|s|-1] + 1)}_{(*)}$$

Pero esEscalera(s) $\rightarrow S[|s|-1] + 1 > S[0] > X$ (por hipótesis) $\therefore X \neq S[|s|-1] + 1$

y como necesito que la Pre sea True, esto implica que $X = S[0] - 1$

(dem: $S[|s|-1] + 1 = K + |s| > K + 0 = S[0]$ } $K + (\text{algo mayor a } 0) > K$

(Por Pre) ← (Por Pre $|s| > 0$) (Por Pre)

Con esto en mente:

Si $X < S[0]$ entonces

$$\text{Pre} \equiv (\exists K : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L S[i] = K + i) \wedge |s| > 0) \wedge_L (X = S[0] - 1)$$

$$\rightarrow (\exists K : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < \underbrace{|\langle x \rangle ++ s|}_{= |s| + 1} \rightarrow_L (\langle x \rangle ++ s)[i] = K + i)$$

✓ • hasta $1 \leq i < |s| + 1$ la implicación es directa ya que estoy comparando los mismos elementos $((\langle x \rangle ++ s)[i] = S[i-1])$ $(\forall i \; 1 \leq i < |s|+1)$

✓ • Para $i = 0$ $(\langle x \rangle ++ s)[0] = X = S[0] - 1 = (K+1) - 1 = K = K + 0 = K + i$ (como se quería)

(por lo visto en el ítem anterior) $\rightarrow (S[0] = (\langle x \rangle ++ s)[1])$

• Si $X \geq S[0]$

De manera similar acá esEscalera(s) $\rightarrow S[0] = K + 0 < K + |s|$

$X \neq S[0] - 1$ ya que $S[0] - 1 < S[0] \leq X$ y como la Pre tiene que ser True, para que se cumpla (*), $X = S[|s|-1] + 1$ tiene que ser verdadero.

Con esto en mente:

Si $X \geq S[0]$

$$\text{Pre} \equiv (\exists K : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L S[i] = K + i) \wedge |s| > 0) \wedge_L (X = S[0 - 1])$$

$$\rightarrow (\exists K : \mathbb{Z})(\forall i : \mathbb{Z})(0 \leq i < \underbrace{|s ++ \langle x \rangle|}_{= |s| + 1} \rightarrow_L (s ++ \langle x \rangle)[i] = K + i)$$

✓ • Para $0 \leq i < |s|$ la implicación es directa $(S[i] = (s ++ \langle x \rangle)[i])$ $(\forall i : \mathbb{Z})$ $(0 \leq i < |s|)$

✓ • Para $i = |s|$ $(s ++ \langle x \rangle)[|s|] = X = S[|s|-1] + 1 = K + (|s|-1) + 1 = K + |s|$

(por el ítem anterior, que ya demostré)

(4) testEncuentraDefecto:

entrada: S = <Hola>

salida esperada: n = 4

sin embargo la salida obtenida es n = 3

Esto se debe a que en la última vuelta del ciclo, llega a identificar que el largo de la secuencia es 4, pero al no volver a entrar, no evalúa n = F, por lo tanto n queda con el valor "antiguo", 3.

Para ser más visual:

1° vuelta: ambos "if" son false → n=0 ∧ F=1 ↘ i=0

2° vuelta: 1° if es true, 2° false → n=1 ∧ F=2 ↘ i=1

3° vuelta: 1° if es True, 2° false → n=2 ∧ F=3 ↘ i=2

4° vuelta: 1° if es True, 2° false → n=3 ∧ F=4 ↘ i=3

↘ i=4

Al salir de la 4° vuelta i=4=|s| ∴ la guarda del ciclo no se cumple, y se da por finalizado. Sin embargo, como vemos n = 3 ≠ 4.

Simplificando, el código "se olvida" de tener en cuenta el último "f++" del ciclo, y entonces, aunque F=4>3=n, no va a "actualizar" el valor de n porque el ciclo terminó. ✓