



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Práctica 3

2do cuatrimestre 2021 (virtual)

Algoritmos y Estructuras de Datos 1

Integrante	LU	Correo electrónico
Jonathan Bekenstein	348/11	jbekenstein@dc.uba.ar



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

<http://www.exactas.uba.ar>

Índice

1. Práctica 3	2
1.1. Ejercicio 1	2
1.2. Ejercicio 2	2
1.3. Ejercicio 3	3
1.4. Ejercicio 4	3
1.5. Ejercicio 5	4
1.6. Ejercicio 6	4
1.7. Ejercicio 7	5
1.8. Ejercicio 8	5
1.9. Ejercicio 9	6

1. Práctica 3

1.1. Ejercicio 1

1.1.1. Pregunta A

El problema es que la post condición se puede indefinir si *result* está fuera del rango de la secuencia. Y eso no puede suceder nunca, las pre y post condiciones solo pueden ser verdaderas o falsas, nunca indefinidas.

```
proc buscar (in l: seq(R), in elem: R, out result: Z) {  
    Pre {elem ∈ l}  
    Post {0 ≤ result < |l| ∧L l[result] = elem}  
}
```

1.1.2. Pregunta B

El problema es que se indefine al indexar $l[i - 1]$ cuando $i = 0$. Como queremos verificar que el elemento en el índice i sea el doble que el elemento en el índice $i - 1$, tenemos que arrancar a revisar desde $i = 1$. Si la secuencia tiene un único elemento, entonces no hay que revisar nada pues el primer número de la progresión geométrica no va a ser el doble de nadie.

```
proc progresionGeometricaFactor2 (in l: seq(Z), out result: Bool) {  
    Pre {True}  
    Post {result = True ↔ ((∀i: Z)(1 ≤ i < |l| →L l[i] = 2 * l[i - 1]))}  
}
```

1.1.3. Pregunta C

El problema es que en la post condición se pide $y \neq x$ pero en el contexto de esta especificación, x no está definido. En cambio, lo que habría que pedir es que $y \neq result$ o más simple aún, quitar esa condición y pedir $y \geq result$. A su vez, también falta especificar que $result \in l$ para garantizar que *result* realmente sea un elemento de la secuencia.

```
proc minimo (in l: seq(Z), out result: Z) {  
    Pre {True}  
    Post {result ∈ l ∧ (∀y: Z)(y ∈ l → y ≥ result)}  
}
```

1.2. Ejercicio 2

1.2.1. Pregunta A

Por ejemplo $l = \langle 1 \rangle$, $suma = 2$. Cumplen la pre condición que es simplemente *True* (o sea, cualquiera cosa cumple la pre condición). Pero no existe forma de cumplir con la post condición ya que no hay suficientes elementos en l para que sumados den 2.

1.2.2. Pregunta B

Sigue siendo inválida porque solo restringe el valor máximo y mínimo que puede tener *suma* pero no garantiza que efectivamente existan elementos en l que sumados den *suma*. Por ejemplo $l = \langle 1, 3 \rangle$, $suma = 2$. Con estos valores se cumple la pre condición: $\min_suma(l) \leq suma \leq \max_suma(l) \leftrightarrow 0 \leq 2 \leq 3$ pero no existen elementos en l que sumados den exactamente 2.

1.2.3. Pregunta C

$$(\exists s : seq(\mathbb{Z}))((\forall x : \mathbb{Z})(\#apariciones(x, s) \leq \#apariciones(x, l)) \wedge suma = \sum_{i=0}^{|s|-1} s[i])$$

1.3. Ejercicio 3

1.3.1. Pregunta A

- I) $x = 0 \rightarrow result \in \{0\}$
- II) $x = 1 \rightarrow result \in \{-1, 1\}$
- III) $x = 27 \rightarrow result \in \{-\sqrt{27}, \sqrt{27}\}$

1.3.2. Pregunta B

- I) $l = \langle 1, 2, 3, 4 \rangle \rightarrow result \in \{3\}$
- II) $l = \langle 15.5, -18, 4.215, 15.5, -1 \rangle \rightarrow result \in \{0, 3\}$
- III) $l = \langle 0, 0, 0, 0, 0, 0 \rangle \rightarrow result \in \{0, 1, 2, 3, 4, 5\}$

1.3.3. Pregunta C

- I) $l = \langle 1, 2, 3, 4 \rangle \rightarrow result = 3$
- II) $l = \langle 15.5, -18, 4.215, 15.5, -1 \rangle \rightarrow result = 0$
- III) $l = \langle 0, 0, 0, 0, 0, 0 \rangle \rightarrow result = 0$

1.3.4. Pregunta D

indiceDelPrimerMaximo y *indiceDelMaximo* tienen necesariamente la misma salida cuando no hay valores repetidos en la secuencia l . En estos casos, sería cuando $l = \langle 1, 2, 3, 4 \rangle$.

1.4. Ejercicio 4

1.4.1. Pregunta A

Incorrecta porque las 2 expresiones deberían estar unidas con un \vee , ya que sino es imposible que se cumplan ambas al mismo tiempo (pues piden $a < 0$ y también $a \geq 0$).

1.4.2. Pregunta B

Incorrecta porque la post condición no contempla el caso cuando $a = 0$.

1.4.3. Pregunta C

Correcta.

1.4.4. Pregunta D

Correcta.

1.4.5. Pregunta E

Incorrecta porque cuando $a \geq 0$, la implicación $a < 0 \rightarrow result = 2 * b$ resulta *True* pues no se cumple el antecedente. Y luego como las 2 implicaciones están unidas con un \vee , este *True* ya hace que toda la post condición sea *True* sin importar si efectivamente $result = b - 1$ como debería ser según la especificación. Pasa lo mismo de forma análoga cuando $a < 0$.

1.4.6. Pregunta F

Correcta.

1.5. Ejercicio 5

1.5.1. Pregunta A

Si recibe $x = 3$ devuelve $result = 9$, lo cual hace verdadera la post condición pues $9 > 3$.

1.5.2. Pregunta B

$$x = 0.5 \rightarrow result = 0.5^2 = 0.25 \not> 0.5$$

$$x = 1 \rightarrow result = 1^2 = 1 \not> 1$$

$$x = -0.2 \rightarrow result = (-0.2)^2 = 0.04 > -0.2$$

$$x = -7 \rightarrow result = (-7)^2 = 49 > -7$$

1.5.3. Pregunta C

```
proc unoMasGrande (in x: R, out result: R) {  
    Pre { $x < 0 \vee x > 1$ }  
    Post { $result > x$ }  
}
```

1.6. Ejercicio 6

1.6.1. Pregunta A

$$P3 > P1 > P2$$

1.6.2. Pregunta B

$$Q3 > Q1 > Q2$$

1.6.3. Pregunta C

Programa 1: $r := x * x$

Programa 2: $r := x * x + 1$

1.6.4. Pregunta D

- a) Cumple porque la nueva pre condición (P3) es más fuerte que la pre condición original (P1).
- b) No cumple porque la nueva pre condición (P2) es más débil que la pre condición original (P1).
- c) Cumple porque la nueva post condición (Q2) es más débil que la post condición original (Q1).

- d) No cumple porque la nueva post condición (Q3) es más fuerte que la post condición original (Q1).
- e) Cumple porque la nueva pre condición (P3) es más fuerte que la pre condición original (P1) y la nueva post condición (Q2) es más débil que la post condición original (Q1).
- f) No cumple porque la nueva pre condición (P2) es más débil que la pre condición original (P1).
- g) No cumple porque la nueva post condición (Q3) es más fuerte que la post condición original (Q1).
- h) No cumple porque la nueva pre condición (P2) es más débil que la pre condición original (P1) y además la nueva post condición (Q3) es más fuerte que la post condición original (Q1).

1.6.5. Pregunta E

Dado un algoritmo que cumple con una especificación, se puede reemplazar dicha especificación por otra y tener garantía que el algoritmo sigue cumpliendo si la nueva pre condición es más fuerte que la original y/o la nueva post condición es más débil que la original.

1.7. Ejercicio 7

1.7.1. Pregunta A

Sabiendo que vale la pre condición de $p1$ se puede afirmar que $x \neq 0$.

Luego, se puede dividir en 2 casos para ver cuándo vale la pre condición de $p2$:

- 1) Si $n > 0$ el antecedente de la implicación es falso y así la implicación resulta verdadera, sin importar el valor de x .
- 2) Si $n \leq 0$ la implicación resulta verdadera si $x \neq 0$. Esto vale pues sabemos que se cumple la pre condición de $p1$.

Por lo tanto vale la pre condición de $p2$.

Nota: Me parece poco formal esta "demostración".

1.7.2. Pregunta B

En esencia lo que me piden es probar que $Post_{p2} \rightarrow Post_{p1} \equiv (result = \lfloor x^n \rfloor \rightarrow x^n - 1 < result \leq x^n)$.

Esto depende del algoritmo usado para calcular la parte entera de x^n . Si se usa la función [techo](#), entonces la implicación vale pues $Post_{p2}$ es literalmente la definición de esa función. Pero si se usa otro algoritmo, por ejemplo la función [piso](#), entonces la implicación no siempre vale.

1.7.3. Pregunta C

No necesariamente, depende del algoritmo usado para calcular la parte entera de x^n .

1.8. Ejercicio 8

Notar que $Pre_{n-esimo1}$ compara con $<$, lo cual significa que no pueden haber 2 elementos iguales en la secuencia l . Por lo tanto, vale que $Pre_{n-esimo1} \rightarrow Pre_{n-esimo2}$.

Por otro lado, $Post_{n-esimo1}$ nos dice que $result \in l$ y además que está en la posición n . Debido a que $Pre_{n-esimo1}$ garantiza que la secuencia l está ordenada, la forma de obtener el índice de $result$ definida en $Post_{n-esimo2}$ en efecto nos va a dar el valor correcto para n .

Al revés no funciona porque $Pre_{n-esimo2}$ solo garantiza que los elementos de la secuencia l sean distintos entre sí, pero eso no implica que la secuencia esté ordenada. Por ejemplo $\langle 1, 3, 2 \rangle$ satisface $Pre_{n-esimo2}$ pero no $Pre_{n-esimo1}$.

1.9. Ejercicio 9

1.9.1. Pregunta A

Dado un número entero, decidir si es par.

```
proc esPar (in n:  $\mathbb{Z}$ , out r: Bool) {  
    Pre {True}  
    Post { $r = True \leftrightarrow n \bmod 2 = 0$ }  
}
```

1.9.2. Pregunta B

Dado un entero n y uno m , decidir si n es un múltiplo de m .

```
proc esMúltiplo (in n:  $\mathbb{Z}$ , in m:  $\mathbb{Z}$ , out r: Bool) {  
    Pre {True}  
    Post { $r = True \leftrightarrow n \bmod m = 0$ }  
}
```

1.9.3. Pregunta C

Dado un número real, devolver su inverso multiplicativo.

```
proc inversoMultiplicativo (in x:  $\mathbb{R}$ , out r:  $\mathbb{R}$ ) {  
    Pre { $x \neq 0$ }  
    Post { $r = 1/x$ }  
}
```

1.9.4. Pregunta D

Dada una secuencia de caracteres, obtener de ella solo los que son numéricos (con todas sus apariciones sin importar el orden de aparición).

```
proc subseqDeNumeros (in s:  $seq\langle Char \rangle$ , out r:  $seq\langle Char \rangle$ ) {  
    Pre {True}  
    Post {( $\forall c : Char$ )( $\#apariciones(r, c) = \text{if } '0' \leq c \leq '9' \text{ then } \#apariciones(s, c) \text{ else } 0$  fi)}}  
}
```

1.9.5. Pregunta E

Dada una secuencia de reales, devolver la secuencia que resulta de duplicar sus valores en las posiciones impares.

```
proc duplicarPosicionesImpares (in s:  $seq\langle \mathbb{R} \rangle$ , out r:  $seq\langle \mathbb{R} \rangle$ ) {  
    Pre {True}  
    Post {|r| = |s|  $\wedge (\forall i : \mathbb{Z})(0 \leq i < |r| \longrightarrow_L r[i] = \text{if } i \bmod 2 = 0 \text{ then } s[i] \text{ else } s[i] * 2$  fi)}}  
}
```

1.9.6. Pregunta F

Dado un número entero, listar todos sus divisores positivos (sin duplicados).

```
proc divisoresPositivos (in n:  $\mathbb{Z}$ , out r:  $seq(\mathbb{Z})$ ) {  
    Pre {True}  
    Post { $(\forall k : \mathbb{Z})((k > 0 \wedge n \bmod k = 0 \leftrightarrow k \in r) \wedge (\#apariciones(r, k) \leq 1))$ }  
}
```