



Guia 2

2do cuatrimestre 2024

Algoritmos y Estructuras de Datos I

Integrante	LU	Correo electrónico
Federico Barberón	112/24	jfedericobarberonj@gmail.com



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - Pabellón I

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Argentina

Tel/Fax: (54 11) 4576-3359

<http://exactas.uba.ar>

Índice

1. Guía 2	3
1.1. Ejercicio 1	3
1.2. Ejercicio 2	3
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	6
1.9. Ejercicio 9	6
1.10. Ejercicio 10	6
1.11. Ejercicio 11	7
1.12. Ejercicio 12	8
1.13. Ejercicio 13	9
1.14. Ejercicio 14	11
1.15. Ejercicio 15	11
1.16. Ejercicio 16	12
1.17. Ejercicio 17	12
1.18. Ejercicio 18	13

1. Guía 2

1.1. Ejercicio 1

Nombrar los siguientes predicados sobre enteros:

- a) `pred esCuadrado` ($x: \mathbb{Z}$) {
 $(\exists c: \mathbb{Z}) (c > 0 \wedge (c * c = x))$
}
- b) `pred esPrimo` ($x: \mathbb{Z}$) {
 $(\forall n: \mathbb{Z}) ((1 < n < x) \rightarrow_L (x \bmod n \neq 0))$
}

1.2. Ejercicio 2

Escriba los siguientes predicados sobre números enteros en lenguaje de especificación:

- a) Que sea verdadero si y sólo si x e y son coprimos.
`pred sonCoprims` ($x, y: \mathbb{Z}$) {
 $(\forall i: \mathbb{Z}) (i > 1 \rightarrow_L \neg(x \bmod i = 0 \wedge y \bmod i = 0))$
}
- b) Que sea verdadero si y es el mayor primo que divide a x .
`pred mayorPrimoQueDivide` ($x, y: \mathbb{Z}$) {
 $(esPrimo(y) \wedge_L x \bmod y = 0) \wedge \neg(\exists i: \mathbb{Z}) (esPrimo(i) \wedge_L (x \bmod i = 0 \wedge i > y))$
}

1.3. Ejercicio 3

Nombre los siguientes predicados auxiliares sobre secuencias de enteros:

- a) `pred todoPositivos` ($s: seq(\mathbb{Z})$) {
 $(\forall i: \mathbb{Z}) ((0 \leq i < |s|) \rightarrow_L s[i] \geq 0)$
}
- b) `pred todosDistintos` ($s: seq(\mathbb{Z})$) {
 $(\forall i: \mathbb{Z}) ((0 \leq i < |s|) \rightarrow_L (\forall j: \mathbb{Z}) ((0 \leq j < |s| \wedge i \neq j) \rightarrow_L (s[i] \neq s[j])))$
}

1.4. Ejercicio 4

Escriba los siguientes predicados auxiliares sobre secuencias de enteros, aclarando los tipos de los parámetros que recibe:

- a) *esPrefijo*, que determina si una secuencia es prefijo de otra.
`pred esPrefijo` ($s1, s2: seq(\mathbb{Z})$) {
 $(|s1| \leq |s2|) \wedge_L (\forall i: \mathbb{Z}) ((0 \leq i < |s1|) \rightarrow_L (s1[i] = s2[i]))$
}
- b) *estáOrdenada*, que determina si la secuencia está ordenada de menor a mayor.
`pred estáOrdenada` ($s: seq(\mathbb{Z})$) {
 $(\forall i: \mathbb{Z}) ((0 \leq i < |s| - 1) \rightarrow_L (s[i] \leq s[i + 1]))$
}

}

- c) *hayUnoParQueDivideAlResto*, que determina si hay un elemento par en la secuencia que divide a todos los otros elementos de la secuencia.

```

pred divideA (d, n:  $\mathbb{Z}$ ) {
     $(d \neq 0) \wedge_L n \bmod d = 0$ 
}

pred hayUnoParQueDivideAlResto (s:  $seq\langle\mathbb{Z}\rangle$ ) {
     $(\exists i : \mathbb{Z}) ((0 \leq i < |s|) \wedge_L esPar(s[i]) \wedge (\forall j : \mathbb{Z}) ((0 \leq j < |s|) \wedge_L divideA(s[i], s[j])))$ 
}

```

- d) *enTresPartes*, que determina si en la secuencia aparecen (de izquierda a derecha) primero 0s, después 1s y por último 2s. Por ejemplo $\langle 0, 0, 1, 1, 1, 2 \rangle$ cumple, pero $\langle 0, 1, 3, 0 \rangle$ o $\langle 0, 0, 0, 1, 1 \rangle$ no. ¿Cómo modificaría la expresión para que se admitan cero apariciones de 0s, 1s y 2s (es decir, para que por ejemplo $\langle 0, 0, 0, 1, 1 \rangle$ o $\langle \rangle$ sí cumplan)?

```

pred tieneSoloCeroUnoYDos (s:  $seq\langle\mathbb{Z}\rangle$ ) {
     $(\forall i : \mathbb{Z}) ((0 \leq i < |s|) \rightarrow_L (s[i] = 0 \vee s[i] = 1 \vee s[i] = 2))$ 
}

pred enTresPartes (s:  $seq\langle\mathbb{Z}\rangle$ ) {
     $tieneSoloCeroUnoYDos(s) \wedge estaOrdenada(s)$ 
}

```

1.5. Ejercicio 5

Sea s una secuencia de elementos de tipo \mathbb{Z} . Escribir una expresión (utilizando sumatoria y productoria) tal que:

- a) Cuente la cantidad de veces que aparece el elemento e de tipo \mathbb{Z} en la secuencia s .

$$\sum_{i=0}^{|s|-1} \text{if } s[i] = e \text{ then } 1 \text{ else } 0 \text{ fi}$$

- b) Sume los elementos en las posiciones impares de la secuencia s .

$$\sum_{i=0}^{|s|-1} \text{if } \neg esPar(s[i]) \text{ then } s[i] \text{ else } 0 \text{ fi}$$

- c) Sume los elementos mayores a 0 contenidos en la secuencia s .

$$\sum_{i=0}^{|s|-1} \text{if } s[i] > 0 \text{ then } s[i] \text{ else } 0 \text{ fi}$$

- d) Sume los inverso multiplicativos ($\frac{1}{x}$) de los elementos contenidos en la secuencia s distintos a 0.

$$\sum_{i=0}^{|s|-1} \text{if } s[i] \neq 0 \text{ then } \frac{1}{s[i]} \text{ else } 0 \text{ fi}$$

1.6. Ejercicio 6

Las siguientes especificaciones no son correctas. Indicar por qué y corregirlas para que describan correctamente el problema.

- a) *progresionGeometricaFactor2*: Indica si la secuencia l representa una progresión geométrica factor 2. Es decir, si cada elemento de la secuencia es el doble del elemento anterior.

```

proc progresionGeometricaFactor2 (in l:  $seq\langle\mathbb{Z}\rangle$ ) : Bool
    requiere {True}

```

$\text{asegura } \{res = True \leftrightarrow ((\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L l[i] = 2 * l[i - 1]))\}$

El asegura se indefin con $i = 0$ pues trata de acceder a $l[-1]$.

Solucion:

```
proc progresionGeometricaFactor2 (in l: seq⟨ℤ⟩) : Bool
  requiere {True}
  asegura {res = True ↔ ((∀ i : ℤ) (0 < i < |l| →L l[i] = 2 * l[i - 1]))}
```

b) minimo: Devuelve en res el menor elemento de l .

```
proc minimo (in l: seq⟨ℤ⟩) : ℤ
  requiere {True}
  asegura {(∀ y : ℤ) ((y ∈ l ∧ y ≠ x) → y > res)}
```

En el asegura se hace referencia a x que no está definida. La lista no puede estar vacía y res tiene que estar en la lista.

Solución:

```
proc minimo (in l: seq⟨ℤ⟩) : ℤ
  requiere {|l| > 0}
  asegura {res ∈ l ∧ (∀ y : ℤ) ((y ∈ l ∧ y ≠ res) → y > res)}
```

1.7. Ejercicio 7

Para los siguientes problemas, dar todas las soluciones posibles a las entradas dadas:

a) $\text{proc indiceDelMaximo (in l: seq⟨ℝ⟩) : ℤ}$
 requiere $\{|l| > 0\}$
 asegura $\{0 \leq res < |l| \wedge_L ((\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L l[i] \leq l[res]))\}$

I) $\langle 1, 2, 3, 4 \rangle \text{ } res = 3$

II) $\langle 15.5, -18, 4.215, 15.5, -1 \rangle \text{ } res = 0 \vee res = 3$

III) $\langle 0, 0, 0, 0, 0, 0 \rangle \text{ } res \in [0, |l|)$

b) $\text{proc indiceDelPrimerMaximo (in l: seq⟨ℝ⟩) : ℤ}$
 requiere $\{|l| > 0\}$
 asegura $\{0 \leq res < |l| \wedge_L ((\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L (l[i] \leq l[res] \vee (l[i] = l[res] \wedge i > res))))\}$

I) $\langle 1, 2, 3, 4 \rangle \text{ } res = 3$

II) $\langle 15.5, -18, 4.215, 15.5, -1 \rangle \text{ } res = 0$

III) $\langle 0, 0, 0, 0, 0, 0 \rangle \text{ } res = 0$

c) ¿Para qué valores de entrada $\text{indiceDelPrimerMaximo}$ y indiceDelMaximo tienen necesariamente la misma salida?

Tienen la misma salida sii el máximo de la lista no está repetido.

1.8. Ejercicio 8

Sea $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ definida como:

$$f(a, b) = \begin{cases} 2b & \text{si } a < 0 \\ b - 1 & \text{en otro caso} \end{cases}$$

Indicar cuáles de las siguientes especificaciones son correctas para el problema de calcular $f(a, b)$. Para aquellas que no lo son, indicar por qué.

a) `proc f (in a, b: \mathbb{R}) : \mathbb{R}`
 `requiere {True}`
 `asegura {(a < 0 \wedge res = 2 * b) \wedge (a \geq 0 \wedge res = b - 1)}`

Falla pues el asegura es siempre falso ya que, por la asociatividad, se tiene que $(a < 0 \wedge a \geq 0 \wedge \dots)$ lo cual es siempre falso.

b) `proc f (in a, b: \mathbb{R}) : \mathbb{R}`
 `requiere {True}`
 `asegura {(a < 0 \wedge res = 2 * b) \vee (a \geq 0 \wedge res = b - 1)}`

Es correcta.

c) `proc f (in a, b: \mathbb{R}) : \mathbb{R}`
 `requiere {True}`
 `asegura {(a < 0 \rightarrow res = 2 * b) \vee (a \geq 0 \rightarrow res = b - 1)}`

Falla pues siempre es verdadera.

d) `proc f (in a, b: \mathbb{R}) : \mathbb{R}`
 `requiere {True}`
 `asegura {res = IfThenElse(a < 0, 2 * b, b - 1)}`

Es correcta.

1.9. Ejercicio 9

Considerar la siguiente especificación, junto con un algoritmo que dado x devuelve x^2 .

```
proc unoMasGrande (in x:  $\mathbb{R}$ ) :  $\mathbb{R}$ 
  requiere {True}
  asegura {res > x}
```

a) ¿Qué devuelve el algoritmo si recibe $x = 3$? ¿El resultado hace verdadera la posrtcondición de unoMasGrande?

El algoritmo devuelve 9 y hace verdadera la postcondición pues $9 \not> 3$.

b) ¿Qué sucede para las entradas $x = 0.5$, $x = 1$, $x = -0.2$ y $x = -7$?

Para los primeros 2 falla y para los últimos 2 cumple.

c) Teniendo en cuenta lo respondido en los puntos anteriores, escribir una **precondición** para unoMasGrande, de manera tal que el algoritmo cumpla con la especificación.

`requiere {|x| > 1}`

1.10. Ejercicio 10

Sean x y r variables de tipo \mathbb{R} . Considerar los siguientes predicados:

- P1: $\{x \leq 0\}$
- P2: $\{x \leq 10\}$
- P3: $\{x \leq -10\}$
- Q1: $\{r \geq x^2\}$
- Q2: $\{r \geq 0\}$
- Q3: $\{r = x^2\}$

a) Indicar la relación de fuerza entre P1, P2 y P3

P1 es más fuerte que P2

P3 es más fuerte que P1 y P2

b) Indicar la relación de fuerza entre Q1, Q2 y Q3

Q1 es más fuerte que Q2

Q3 es más fuerte que Q2 y Q1

c) Escribir 2 programas que cumplan con la siguiente especificación:

```
proc hagoAlgo (in x:  $\mathbb{R}$ ) :  $\mathbb{R}$ 
  requiere  $\{x \leq 0\}$ 
  asegura  $\{res \geq x^2\}$ 
```

$S1 \equiv res := x^2$

$S2 \equiv res := x^2 + 1$

d) Sea A un algoritmo que cumple con la especificación del ítem anterior. Decidir si necesariamente cumple las siguientes especificaciones:

a) requiere $\{x \leq -10\}$, asegura $\{r \geq x^2\}$ Cumple.

b) requiere $\{x \leq 10\}$, asegura $\{r \geq x^2\}$ No Cumple.

c) requiere $\{x \leq 0\}$, asegura $\{r \geq 0\}$ Cumple.

d) requiere $\{x \leq 0\}$, asegura $\{r = 0\}$ No Cumple.

e) requiere $\{x \leq -10\}$, asegura $\{r \geq 0\}$ Cumple.

f) requiere $\{x \leq 10\}$, asegura $\{r = 0\}$ No Cumple.

e) ¿Qué conclusión pueden sacar? ¿Qué debe cumplirse con respecto a las precondiciones y postcondiciones para que sea seguro **reemplazar la especificación**?

Se concluye que para poder reemplazar una especificación por otra debe ocurrir que la nueva precondición sea igual o más fuerte que la anterior, y la postcondición sea igual o más débil que la anterior.

1.11. Ejercicio 11

Considerar las siguientes dos especificaciones, junto con un algoritmo a que satisface la especificación de p2.

```
proc p1 (in x:  $\mathbb{R}$ , in n:  $\mathbb{Z}$ ) :  $\mathbb{Z}$ 
  requiere  $\{x \neq 0\}$ 
  asegura  $\{x^n - 1 < res \leq x^n\}$ 
```

```
proc p2 (in x:  $\mathbb{R}$ , in n:  $\mathbb{Z}$ ) :  $\mathbb{Z}$ 
  requiere  $\{n \leq 0 \rightarrow x \neq 0\}$ 
  asegura  $\{res = \lfloor x^n \rfloor\}$ 
```

a) Dados valores de x y n que hacen verdadera la precondición de p1, demostrar que hacen también verdadera la precondición de p2.

Basta probar que $x \neq 0 \rightarrow (n \leq 0 \rightarrow x \neq 0)$

$$\begin{aligned}
x \neq 0 &\rightarrow (n \leq 0 \rightarrow x \neq 0) \\
x = 0 &\vee (n > 0 \vee x \neq 0) \\
(x = 0 \vee x \neq 0) &\vee n > 0 \\
True &\vee n > 0 \\
True
\end{aligned}$$

Por lo tanto, los valores que cumplen la precondition de p1 cumplen la precondition de p2.

- b) Ahora, dados estos valores de x y n , supongamos que se ejecuta a : llegamos a un valor de res que hace verdadera la postcondición de p2. ¿Será también verdadera la postcondición de p1 con este valor de res ?

Si res satisface la postcondición de p2 esto es que $res = \lfloor x^n \rfloor$. Para ver si satisface la precondition de p1 hay que ver si es verdadero que $x^n - 1 < \lfloor x^n \rfloor \leq x^n$, lo cuál es trivial. Por lo tanto satisface ambas postcondiciones.

- c) ¿Podemos concluir que a satisface la especificación de p1?

Si. Como cualquier valor que cumple la precondition de p1, cumple a su vez la precondition de p2, entonces como a satisface la especificación de p2, ejecutar a con cualquier valor que satisfaga la precondition de p1 devolverá un valor que satisface ambas postcondiciones, en particular la de p1, por lo tanto, a satisface la especificación de p1.

1.12. Ejercicio 12

Especificar los siguientes problemas:

- a Dado un entero, decidir si es par

```

proc esPar (in n:  $\mathbb{Z}$ ) : Bool
    requiere {True}
    asegura {res = true  $\leftrightarrow$  n mód 2 = 0}

```

- b Dado un entero n y otro m , decidir si n es un múltiplo de m .

```

proc esMultiplo (in n,m:  $\mathbb{Z}$ ) : Bool
    requiere {True}
    asegura {res = true  $\leftrightarrow$  ( $\exists i : \mathbb{Z}$ ) ( $n = i * m$ )}

```

- c Dado un entero, listar todos sus divisores positivos (sin duplicados)

```

proc divisoresPos (in n:  $\mathbb{Z}$ ) : seq( $\mathbb{Z}$ )
    requiere {True}
    asegura {sinDuplicados(res)  $\wedge$  sonDivisoresPos(n, res)  $\wedge$  noHayOtroDivisorPos(n, res)}
    pred sonDivisoresPos (n:  $\mathbb{Z}$ , divsPos: seq( $\mathbb{Z}$ )) {
        ( $\forall i : \mathbb{Z}$ ) ( $i \in divsPos \rightarrow_L i > 0 \wedge divideA(i, n)$ )
    }
    pred noHayOtroDivisorPos (n:  $\mathbb{Z}$ , divsPos: seq( $\mathbb{Z}$ )) {
         $\neg(\exists i : \mathbb{Z}) (i > 0 \wedge i \notin divsPos \wedge divideA(i, n))$ 
    }
    pred sinDuplicados (list: seq( $\mathbb{Z}$ )) {
        ( $\forall i : \mathbb{Z}$ ) ( $i \in list \rightarrow_L cantApariciones(i, list) = 1$ )
    }

```



```

}
pred divideA (d, n:  $\mathbb{Z}$ ) {
  ( $\exists k : \mathbb{Z}$ ) ( $n = d * k$ )
}

aux cantApariciones (e:  $\mathbb{Z}$ , list:  $\text{seq}\langle\mathbb{Z}\rangle$ ) :  $\mathbb{Z} = \sum_{i=0}^{|list|-1} \text{IfThenElse}(list[i] = e, 1, 0);$ 

```

- d Dado un entero positivo, obtener su descomposición en factores primos. Devolver una secuencia de tuplas (p, e) , donde p es un primo y e es su exponente, ordenada en forma creciente con respecto a p .

```

proc descomposicionEnPrimos (in n:  $\mathbb{Z}$ ) :  $\text{seq}\langle\mathbb{Z} \times \mathbb{N}\rangle$ 
  requiere { $n \geq 0$ }
  asegura { $\text{estaOrdenada}(\text{res}) \wedge$ 
    ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |\text{res}| \rightarrow_L \text{esPrimo}(\text{res}[i]_0)$ )  $\wedge$ 
    ( $\prod_{i=0}^{|\text{res}|-1} \text{res}[i]_0^{\text{res}[i]_1} = n$ )}
  pred estaOrdenada (list:  $\text{seq}\langle\mathbb{Z} \times \mathbb{N}\rangle$ ) {
    ( $\forall i : \mathbb{Z}$ ) ( $0 \leq i < |\text{list}| - 1 \rightarrow_L \text{list}[i]_0 \leq \text{list}[i+1]_0$ )
  }
  pred esPrimo (n:  $\mathbb{Z}$ ) {
    ( $\forall i : \mathbb{Z}$ ) (( $i > 0 \wedge_L n \bmod i = 0$ )  $\rightarrow_L (i = 1 \vee i = n)$ )
  }
}

```

1.13. Ejercicio 13

Especificar los siguientes problemas sobre secuencias:

- a) Dados dos secuencias s y t , decidir si s está *incluida* en t , es decir, si todos los elementos de s aparecen en t en igual o mayor cantidad.

```

proc estaIncluida (in s, t:  $\text{seq}\langle\mathbb{Z}\rangle$ ) : Bool
  requiere {True}
  asegura { $\text{res} = \text{true} \leftrightarrow (\forall i : \mathbb{Z}) (i \in s \rightarrow \text{cantApariciones}(i, s) \leq \text{cantApariciones}(i, t))$ }

  aux cantApariciones (e:  $\mathbb{Z}$ , s:  $\text{seq}\langle\mathbb{Z}\rangle$ ) :  $\mathbb{Z} = \sum_{i=0}^{|s|-1} \text{IfThenElse}(s[i] = e, 1, 0);$ 

```

- b) Dadas dos secuencias s y t , devolver su *intersección*, es decir, una secuencia con todos los elementos que aparecen en ambas. Si un mismo elemento tiene repetidos, la secuencia retornada debe contener la cantidad mínima de apariciones del elemento en s y t .

```

proc interseccion (in s, t:  $\text{seq}\langle\mathbb{Z}\rangle$ ) :  $\text{seq}\langle\mathbb{Z}\rangle$ 
  requiere {True}
  asegura { $\text{todoElementoPerteneceAAmbas}(\text{res}, s, t) \wedge$ 
     $\text{todoElementoConLaMinimaAparicion}(\text{res}, s, t)$ }
  pred todoElementoPerteneceAAmbas (list, s, t:  $\text{seq}\langle\mathbb{Z}\rangle$ ) {
    ( $\forall n : \mathbb{Z}$ ) ( $n \in \text{list} \leftrightarrow (n \in s \wedge n \in t)$ )
  }
  pred todoElementoConLaMinimaAparicion (list, s, t:  $\text{seq}\langle\mathbb{Z}\rangle$ ) {

```

```

    (∀n : ℤ) (
      n ∈ list → cantApariciones(n, list) = min(cantApariciones(n, s), cantApariciones(n, t))
    )
  }

  aux cantApariciones (e: ℤ, s: seq⟨ℤ⟩) : ℤ = ∑i=0|s|-1 IfThenElse(s[i] = e, 1, 0);

```

- c) Dada una secuencia de números enteros, devolver aquel que divida a más elementos de la secuencia. El elemento tiene que pertenecer a la secuencia original. Si existe más de un elemento que cumple esta propiedad, devolver alguno de ellos.

```

proc elQueMasDivide (in s: seq⟨ℤ⟩) : ℤ
  requiere {True}
  asegura {res ∈ s ∧ (∀n : ℤ) (
    n ∈ s → cantidadDeMultiplosEnLista(n, s) ≤ cantidadDeMultiplosEnLista(res, s)
  )}
  pred divideA (d, n: ℤ) {
    (∃k : ℤ) (n = d * k)
  }

  aux cantidadDeMultiplosEnLista (e: ℤ, s: seq⟨ℤ⟩) : ℤ = ∑i=0|s|-1 IfThenElse(divideA(e, s[i]), 1, 0);

```

- d) Dada una secuencia de secuencias de enteros l , devolver una secuencia de l que contenga el máximo valor. Por ejemplo, si $l = \langle \langle 2, 3, 5 \rangle, \langle 8, 1 \rangle, \langle 2, 8, 4, 3 \rangle \rangle$, devolver $\langle 8, 1 \rangle$ o $\langle 2, 8, 4, 3 \rangle$

```

proc secuenciaConMax (in s: seq⟨seq⟨ℤ⟩⟩) : seq⟨ℤ⟩
  requiere {True}
  asegura {res ∈ s ∧
    (∃max : ℤ) (max ∈ res ∧ (∀i : ℤ) (0 ≤ i < |s| →L (∀n : ℤ) (n ∈ s[i] → n ≤ max)))}

```

- e) Dada una secuencia l con todos sus elementos distintos, devolver la secuencia de *partes*, es decir, la secuencia de todas las secuencias incluidas en l , cada una con sus elementos en el mismo orden en que aparecen en l .

```

proc partes (in l: seq⟨ℤ⟩) : seq⟨seq⟨ℤ⟩⟩
  requiere {todosDistintos(l)}
  asegura {}
  HACER!
  pred todosDistintos (l: seq⟨ℤ⟩) {
    (∀n : ℤ) (n ∈ l → cantApariciones(n, l) = 1)
  }

  aux cantApariciones (e: ℤ, s: seq⟨ℤ⟩) : ℤ = ∑i=0|s|-1 IfThenElse(s[i] = e, 1, 0);

```

1.14. Ejercicio 14

Dados dos enteros a y b , se necesita calcular la suma y retornarla en un entero c . ¿Cuáles de las siguientes especificaciones son correctas para ese problema? Para las que no lo son, indicar por qué.

- a) `proc sumar (inout a, b, c: \mathbb{Z})`
 `requiere $\{True\}$`
 `asegura $\{a + b = c\}$`

Esta especificación está mal porque los valores de a y b pueden modificarse durante la ejecución, lo cual produciría un resultado no deseado.

- b) `proc sumar (in a, b: \mathbb{Z} , inout c: \mathbb{Z})`
 `requiere $\{True\}$`
 `asegura $\{c = a + b\}$`

Esta especificación es correcta.

- c) `proc sumar (inout a, b, c: \mathbb{Z})`
 `requiere $\{a = A_0 \wedge b = B_0\}$`
 `asegura $\{a = A_0 \wedge b = B_0 \wedge c = a + b\}$`

Esta especificación es correcta.

1.15. Ejercicio 15

Dada una secuencia l , se desea sacar su primer elemento y devolverlo. Decidir cuáles de estas especificaciones son correctas. Para las que no lo son, indicar por qué y justificar con ejemplos.

- a) `proc tomarPrimero (inout l: $seq(\mathbb{R})$) : \mathbb{R}`
 `requiere $\{|l| > 0\}$`
 `asegura $\{res = head(l)\}$`

Está mal porque el asegura referencia al estado final de l , por lo que $head(l)$ sería el segundo elemento de la lista. Además no pone condiciones sobre que hay que sacar el primer elemento de la lista.

- b) `proc tomarPrimero (inout l: $seq(\mathbb{R})$) : \mathbb{R}`
 `requiere $\{|l| > 0 \wedge l = L_0\}$`
 `asegura $\{res = head(L_0)\}$`

Está mal pues no especifica que se debe sacar el primer elemento de la lista.

- c) `proc tomarPrimero (inout l: $seq(\mathbb{R})$) : \mathbb{R}`
 `requiere $\{|l| > 0\}$`
 `asegura $\{res = head(L_0) \wedge |l| = |L_0| - 1\}$`

Está mal pues L_0 no está definida previamente en el requiere. Que la longitud sea 1 menor no necesariamente implica que se haya eliminado el primer elemento.

- d) `proc tomarPrimero (inout l: $seq(\mathbb{R})$) : \mathbb{R}`
 `requiere $\{|l| > 0 \wedge l = L_0\}$`
 `asegura $\{res = head(L_0) \wedge l = tail(L_0)\}$`

Esta especificación es correcta.

1.16. Ejercicio 16

Dada una secuencia de enteros, se requiere multiplicar por 2 aquéllos valores que se encuentran en posiciones pares. Indicar por qué son incorrectas las siguientes especificaciones y proponer una alternativa correcta.

a) **proc duplicarPares** (inout l: $seq\langle\mathbb{Z}\rangle$)
 requiere $\{l = L_0\}$
 asegura $\{|l| = |L_0| \wedge$
 $(\forall i : \mathbb{Z}) ((0 \leq i < |l| \wedge i \bmod 2 = 0) \rightarrow_L l[i] = 2 * L_0[i])\}$

Está mal pues no especifica que los elementos en posiciones impares se mantienen igual

b) **proc duplicarPares** (inout l: $seq\langle\mathbb{Z}\rangle$)
 requiere $\{l = L_0\}$
 asegura $\{(\forall i : \mathbb{Z}) ((0 \leq i < |l| \wedge i \bmod 2 \neq 0) \rightarrow_L l[i] = L_0[i]) \wedge$
 $(\forall i : \mathbb{Z}) ((0 \leq i < |l| \wedge i \bmod 2 = 0) \rightarrow_L l[i] = 2 * L_0[i])\}$

Está mal pues no especifica que la secuencia tenga la misma cantidad de elementos.

c) **proc duplicarPares** (inout l: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$
 requiere $\{True\}$
 asegura $\{|l| = |res| \wedge$
 $(\forall i : \mathbb{Z}) ((0 \leq i < |l| \wedge i \bmod 2 \neq 0) \rightarrow_L res[i] = l[i]) \wedge$
 $(\forall i : \mathbb{Z}) ((0 \leq i < |l| \wedge i \bmod 2 = 0) \rightarrow_L res[i] = 2 * l[i])\}$

Está mal pues no especifica que tiene que modificar la secuencia original.

Solución propuesta:

proc duplicarPares (inout l: $seq\langle\mathbb{Z}\rangle$)
 requiere $\{l = L_0\}$
 asegura $\{|l| = |L_0| \wedge_L$
 $(\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L (i \bmod 2 = 0 \wedge l[i] = 2 * L_0[i]) \vee (i \bmod 2 \neq 0 \wedge l[i] = L_0[i]))\}$

1.17. Ejercicio 17

Especificar los siguientes problemas de modificación de secuencias:

a) Dada una secuencia de enteros mayores a dos, reemplaza dichos valores por el número primo menor más cercano. Por ejemplo, si $l = \langle 6, 5, 9, 14 \rangle$, luego de aplicar $primosHermanos(l)$, $l = \langle 5, 3, 7, 13 \rangle$

proc primosHermanos (inout l: $seq\langle\mathbb{Z}\rangle$)
 requiere $\{(\forall n : \mathbb{Z}) (n \in l \rightarrow_L n > 2) \wedge l = L_0\}$
 asegura $\{|l| = |L_0| \wedge_L$
 $(\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L esElPrimoMasCercano(l[i], L_0[i]))\}$
 pred esElPrimoMasCercano (p, n: \mathbb{Z}) {
 $esPrimo(p) \wedge p < n \wedge (\forall q : \mathbb{Z}) ((esPrimo(q) \wedge q < n) \rightarrow_L q \leq p)$
 }

b) Reemplaza todas las apariciones de a en l por b .

proc reemplazar (inout l: $seq\langle\text{Char}\rangle$, in a,b: Char)
 requiere $\{l = L_0\}$
 asegura $\{|l| = |L_0| \wedge$
 $(\forall i : \mathbb{Z}) (0 \leq i < |l| \rightarrow_L l[i] = \text{IfThenElse}(L_0[i] = a, b, L_0[i]))\}$

- c) Elimina los elementos duplicados de l dejando sólo su primera aparición (en el orden original). Devuelva además una secuencia con todas las apariciones eliminadas (en cualquier orden)

```
proc limpiarDuplicados (inout l: seq<Char>) : seq<Char>
    requiere {l = L0}
    asegura {}
```

HACER!

1.18. Ejercicio 18

HACER!