Ejercicios De Repaso Pre-Parcial

Especificación

Completar las siguientes especificaciones:

problema a (in b: $seq\langle\mathbb{Z}\rangle$, in c: $\mathbb{Z}$) : $\mathbb{Z}$ {
  requiere x: $\{c \in b\}$
  asegura y: $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i)\}$
}

En este ejercicio hay que sustituir a, x e y por nombres declarativos sobre que hace la especificación.

el requiere nos dice que c pertenece a la secuencia b.

el asegura nos dice que existe un resultado i el cual es la posición donde se encuentra el elemento c en la lista b.

entonces unos posibles nombres podrían ser:

problema IndiceDeC (in b: seq<Z>, in c: Z) : Z {

requiere perteneceC { c∈b }

asegura ExistePosiciónDeCenB { (∃i: Z) (0≤i<|b| ∧ b[i] = c ∧ res = i) }

}

Especificación (más difícil)

poner nombres declarativos a w, x, y, z

problema a (in b: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {

requiere: {True}

asegura w: $\{|resultado| \leq |b|\}$

asegura x: $\{(\forall i : \mathbb{Z})(0 \leq i < |resultado| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \wedge resultado[i] = b[j]))\}$

asegura y: $\{(\forall i : \mathbb{Z})(0 \leq i < |b| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |resultado| \wedge b[i] = resultado[j]))\}$

asegura z: $\{(\forall i, j : \mathbb{Z})((0 \leq i, j < |resultado| \wedge resultado[i] = resultado[j]) \rightarrow i = j)\}$

}

escribo semiformalmente que nos pide los asegura.

asegura w { la longitud de res es menor o igual a la longitud de b }

asegura x { todo elemento res es igual a algún elemento de b }

asegura y { todo elemento de b es igual a algún elemento de res }

asegura z { si dos elementos de res son iguales están en la misma posición }

esta especificación lo que hace es devolvernos una sequencia sin los elementos repetidos de b.

posibles nombres podrían ser:

- a: sinRepetidos
- w: longitudMenorOIgual
- x: elemsdeResenb
- y: elemsdebenRes
- z: NoRepetidosenRes

Como podemos extender esa especificación para que devuelva además una lista de pares de enteros que indique los elementos y cantidades eliminadas?

problema **eliminarYcontarRepetidos** (in $l : seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle \times seq\langle\mathbb{Z} \times \mathbb{Z}\rangle$ {
  requiere: {...}
  asegura: {...}
}

NOTA: la idea es reutilizar sinRepetidos

ejemplo: $b = [1, 2, 1, 1, 3, 2]$

$res = ([1,2,3], ([1,2], [2,1]))$

- res_0: un posible res
- res_1: b sin repetidos
- $res_{1,0}$: elemento eliminado
- $res_{1,1}$: cantidades eliminadas

la especificación que se me ocurrió fue la siguiente:

problema eliminarYcontarRepetidos (in l: $seq\langle\mathbb{Z}\rangle$): $seq\langle\mathbb{Z}\rangle \times seq\langle\mathbb{Z}\times\mathbb{Z}\rangle$ {

requiere {True}

asegura { $res_0 = sinRepetidos(l)$ }

asegura { $(\forall i : \mathbb{Z})(i \in l \wedge \#Apariciones(i, l) > 1 \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |res_1| \wedge res_1[j] = (i, \#Apariciones(i,l) - 1))$ }

asegura { $sinRepetidos(res_1)$ }

}

problema #Apariciones ($n : \mathbb{Z}$, $l : seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
  requiere {True}
  asegura { $res = \sum_{i=0}^{|l|-1} (\text{if } l[i] = n \text{ then } 1 \text{ else } 0 \text{ fi})$ }

Ahora vamos a programar!

- Implementar en Haskell la función `eliminarYcontarRepetidos :: [Int] -> ([Int], [(Int, Int)])`
- Implementar en Pyhton la función `def sacarRepetidos(l:list) -> list`

Implementacion en Haskell de eliminarYcontarRepetidos

```
eliminarYcontarRepetidos :: [Int] -> ([Int],[(Int,Int)])
eliminarYcontarRepetidos l = ( sinRepetidos l, contarRepetidos (sinRepetidos l) l )

contarRepetidos :: [Int] -> [Int] -> [(Int, Int)]
contarRepetidos [] l = []
contarRepetidos (x:xs) l | #apariciones x l > 1 = (x,(#apariciones x l)-1) : contarRepetidos xs l
                         | otherwise = contarRepetidos xs l

sinRepetidos :: [Int] -> [Int]
sinRepetidos [] = []
sinRepetidos (x:xs) | elem x xs = sinRepetidos xs
                    | otherwise = x : sinRepetidos xs

#apariciones :: Int -> [Int] -> Int
#apariciones n [] = 0
#apariciones n (x:xs) | n == x = #apariciones n xs + 1
                      | otherwise = #apariciones n xs
```

NOTA: la especificación no me pide que esten ordenados los elementos de re,

Implementación de sinRepetidos en Python:

```
def sinRepetidos ( l: list[Int]) -> list
    lsinRep : list = []
    for elem in l:
        if elem not in lsinRep:
            lsinRep.append (elem)

    return lsinRep
```

```
def f(s1: list[int], s2: list[int]):
L1   i:int = 0
L2   a:int = 0
L3   b:int = 0
L4   while i < len(s1):
L5       a = s1[i]
L6       if i >= len(s2):
L7           b = 0
         else:
L8           b = s2[i]
L9       s1[i] = a + b
L10      if i < len(s2):
L11          if a - b > 0:
L12              s2[i] = b - a
             else:
L13              s2[i] = a - b
L14      i += 1
```

Dada la siguiente especificación y programa

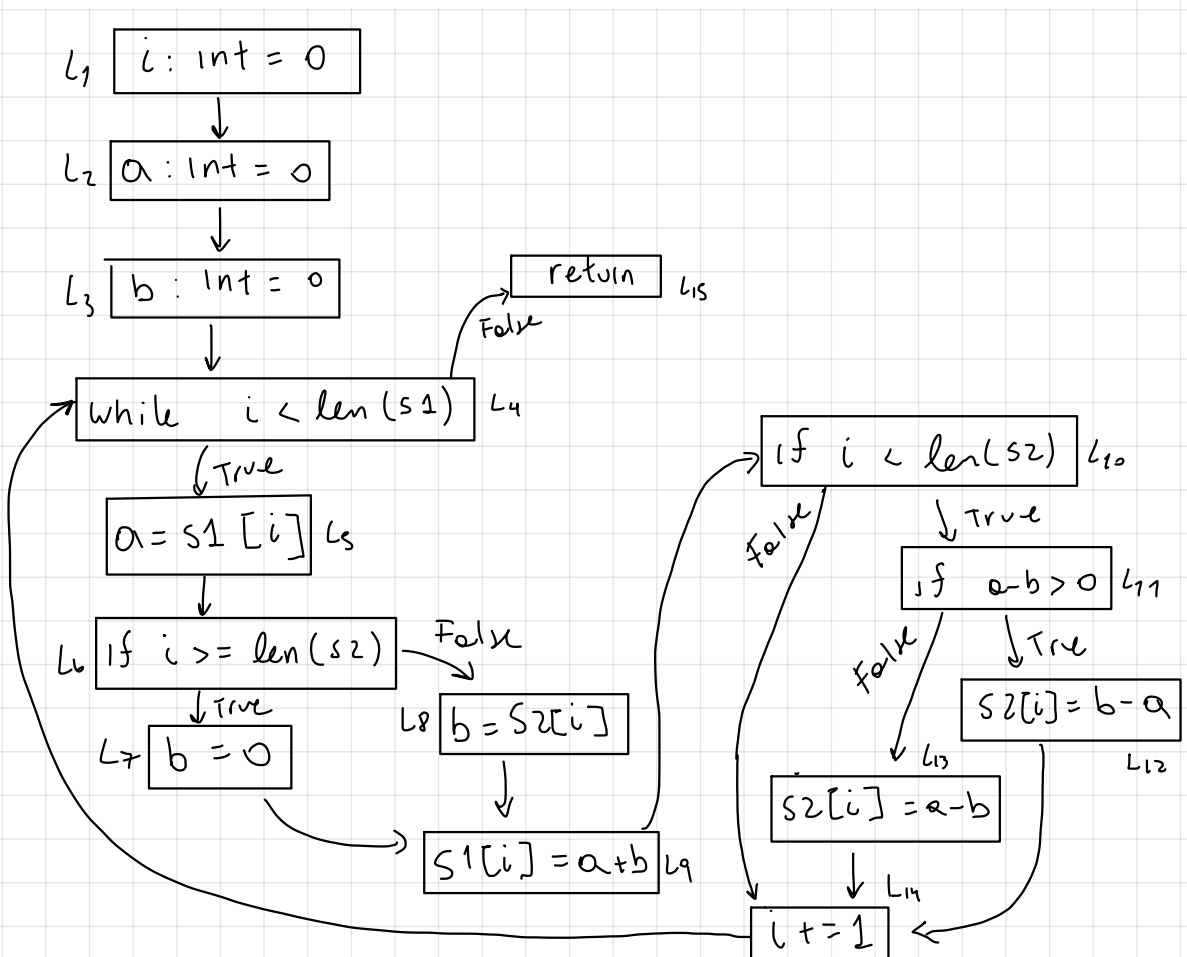
problema f (inout s1: $seq\langle\mathbb{Z}\rangle$, inout s2: $seq\langle\mathbb{Z}\rangle$) {
  requiere: {True}
  asegura: $\{(\forall i \in \mathbb{Z})((0 \leq i < |s1| \land 0 \leq i < |s2|) \rightarrow$
  $(s1[i] = s1@pre[i] + s2@pre[i]) \land$
  $(s2[i] = abs(s1@pre[i] - s2@pre[i]))) \land$
  $(\forall i \in \mathbb{Z})((i \geq |s1| \land 0 \leq i < |s2|) \rightarrow s2[i] = s2@pre[i]) \land$
  $(\forall i \in \mathbb{Z})((i \geq |s2| \land 0 \leq i < |s1|) \rightarrow s1[i] = s1@pre[i])\}$
}

@pre se refiere al valor original (in) de esa variable.

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

1. Describir el diagrama de control de flujo (*control-flow graph*) del programa.
2. 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).
3. 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).
4. ¿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.

1)



2) test suite:

- Test A
  - * entrada: $s_1 = [1,2]$ $s_2[4,1]$
  - * Salida esperada. $s_1 = [5,7]$ $s_2 = [3,3]$
- Test B
  - * entrada: $s_1 = [1]$ $s_2 = [4,2]$
  - * Salida esperada: $s_1 = [5]$ $s_2 = [3,2]$
- Test C
  - * entrada: $s_1 = [-1,16]$ $s_2 = [4]$
  - * Salida esperada: $s_1 = [3,16]$ $s_2 = [5]$

Cubrimento de sentencias:

test	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}	L_{11}	L_{12}	L_{13}	L_{14}	L_{15}
Test A	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Test B	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí
Test C	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí

$COV_{sentencias} = \frac{15}{15} = 100\%$

cubre todas las sentencias mi test suite ya que ejecuta todos los nodos del programa

Cubrimento de branches

test	L_4-True	L_4-False	L_6-True	L_6-False	L_{10}-True	L_{10}-False	L_{11}-True	L_{11}-False
Test A	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí
Test B	Sí	Sí	No	Sí	Sí	No	No	Sí
Test C	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí

$COV_{branches} = \frac{8}{8} = 100\%$

el test suite cubre todos los branches ya que ejecuta todas los posibles desiciones del programa.

3) el programa tiene un defecto que se puede encontrar con el test B, según la especificación la salida esperada es $s_1 = [5]$ $s_2 = [3,2]$ pero el programa devuelve $s_1 = [5]$ $s_2 = [-3,2]$, el defecto se debe a la lineas L_{11} L_{12} L_{13} en vez de aplicarle valor absoluto a la resta de $s_1[i] - s_2[i]$ cambian de lugar las variables de la resta generando que nos devuelva como resultado -3 en vez de 3.

4) No es posible ya que para que se ejecuten las branches L_{11}-True y L_{11}-False si o sí se ejecuta el defecto de las lineas L_{12} y L_{13}.

1. ¿Qué diferencia hay entre testing de caja blanca y de caja negra?
2. ¿Que es una variable InOut?

1) la diferencia entre el testing de caja blanca y el testing de caja negra es que en el primero se utiliza el codigo del programa al hacer testing mientras en el segundo solo se utiliza la especificación del problema.

2) una variable InOut es una variable la cual se toma como parametro al especificar, se modifica en la especificación o procedimiento y se le asigna un nuevo valor.