Ej 1

SEBASTIÁN ANDRÉS 1026/22 hoja 1

Ej 1)

(a) OBSERVACIONES:

- a recibe una secuencia de tuplas <char x char> (b) y una secuencia de CHAR (c), y devuelve otra secuencia de char.
- requiere x: DICE QUE las tuplas en (B) son distintas entre sí en la primer componente.
- requiere y: DICE QUE las tuplas en (B) son distintas entre sí en la segunda componente.
- requiere Z: EXIGE QUE TODOS LOS CHARS DE (C) ESTÁN COMO PRIMER COMPONENTE DE ALGUNA TUPLA EN (B).
- ASEGURA M: la longitud del resultado y de (C) es la misma.
- ASEGURA W: DICE QUE resultado[i] es la segunda componente de la tupla en (B) que tiene como primer componente a c[i].

~ LUEGO, PODEMOS REESCRIBIR la ESPECIFICACIÓN DE LA SIG. FORMA...

~~[illegible]~~

•)

problema <u>leerSegundaComponente</u> (in l: seq<char×char>, in ids: seq<char>) : seq<char> {

requiere <u>distintos1Componente</u> {

$(\forall i,j : \mathbb{Z})((0 \leq i < |l| \land 0 \leq j < |l| \land i \neq j) \rightarrow l[i]_0 \neq l[j]_0)$

}

requiere <u>distintos2Componente</u> {

$(\forall i,j : \mathbb{Z})((0 \leq i < |l| \land 0 \leq j < |l| \land i \neq j) \rightarrow l[i]_1 \neq l[j]_0)$

}

requiere <u>todoIdDeCenB</u> {

$(\forall i : \mathbb{Z})(0 \leq i < |ids| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |l| \land l[j]_0 = ids[i]))$

}

asegura <u>mismoTamaño</u> { $|resultado| = |ids|$ }

asegura <u>devuelve2Componente</u> {

$(\forall i : \mathbb{Z})(0 \leq i < |ids| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |l| \land l[j]_0 = ids[i] \land l[j]_1 = res[i]))$

}

}

• EN MI CASO, INTERPRETÉ QUE (B) era ...

$B = [(id_0, valor_0), \ldots, (id_N, valor_N)]$

LUEGO LLAMÉ A C = IDS y la función (a) TAMBIÉN PODRÍA LLAMARSE "LEERVALORES".

B

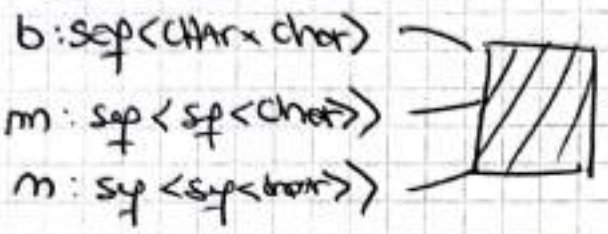
→ Punto B

Ej 4

SEBASTIÁN ANDRÉS 1028/22

HOJA 2

(b) ME PIDE ESPECIFICAR:



- true ⇔ n es el resultado de aplicar A a cada elemento de m.

- PROBLEMA sonLosValores (b: seq<char×char>, m, n: seq<seq<char>>): Bool {
  requiere {true} No estás teniendo en cuenta los requiere de leerSegundaComponente
  ASEGURA {
    res = true ⇔ |m| = |n| ∧ son2Componente(m,n) ✓
  }
}

- pred son2Componente (m, n: seq<seq<char>>) {
  (∀i: ℤ)(0 ≤ i < |m| ∧ n[i] = leerSegundaComponente(b, m[i])) ✓
}

- DONDE leerSegundaComponente(b, ids) es el (a) especificado en el punto anterior. ✓ R

Ej 2

SEBASTIAN ANDRÉS 1020/22

hoja 3

Ej 2) i- Programar en Haskell (a)

ii- Programar en Python (a) :

---

i)

```
a :: [(char,char)] -> [char] -> [char]
a VALORES [] = []                                   B-
a valores (x:xs) = segComp x valores : a valores xs

segComp :: char -> [(char,char)] -> char
segComp n [] = undefined -- ASUMO POR ESPECIFICACIÓN QUE ESTE CASO NUNCA SUCEDE
segComp n ((x,v):xs) | n==x = v
                     | otherwise = segComp n xs
```

ii)

```
DEF a( b:List[(char,char)] , c: List[char] ) -> List[char] :
    # Asumo el REQUIRE (∀ id ∈ C -> (∃ t: b) (t_0 == id))
    out:List[char] = list()
    for id in C:
        for (i, VALUE) in b:                        B-
            if i==id:
                out.append(VALUE)
                BREAK
        PASS X
    RETURN OUT
```

OBSERVACIONES

[(char,char)]

i) sgComp :: char → [(char,char)] → char es una auxiliar que devuelve la segunda componente de la tupla en la lista de tuplas dada, que tiene como primer componente al char dado.

Ej:

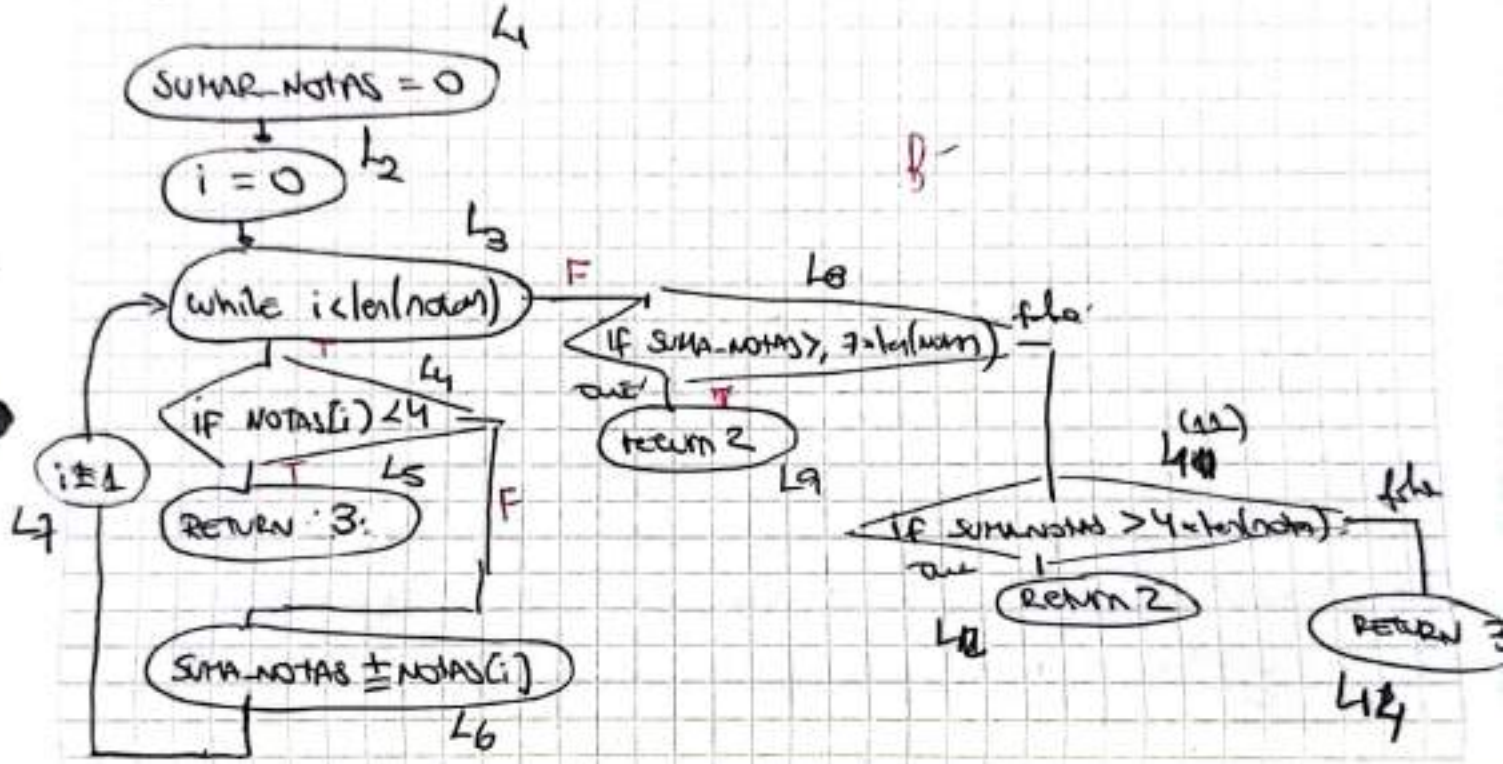
sgComp 1 [(2,3),(1,0)] → 0

ii) En Python aproveche la descomposición de tuplas iterando en la lista de tuplas pero transpilatoriamente podría haber usado un while (i < len(C)) ... i++ o con un for i in range(len(C)).

Otra observación es que Python admite escribir list en vez de List[int] de la librería Typing o hasta no escribirlo.

Ej 3 SEBASTIÁN ANDRÉS 10/08/22 hoja 4

Ej 3) i) ARMAR CFG:



ii) ESCRIBIR TEST SUITE QUE PASE POR TODOS LOS LINEAS

TEST SUITE:

- ASSERT APROBADO ([3]) == 3; || líneas: {L1, L2, L3, L4, L5} ✓
- ASSERT APROBADO ([10,10]) == 1; || LÍNEAS {L1, L2, L3, L4, L6, L7, L8, L9} ✓
- ASSERT APROBADO ([6,6]) == 2; || LÍNEAS: {L1, L2, L3, L4, L6, L7, L8, L10, L11, L12} ✓
- ASSERT APROBADO ([4,4]) == 2; || líneas: {L1, L2, L3, L4, L6, L7, L8 L10, L11, L13, L14} ✓

∴ LUEGO, ESTOS 4 TEST CASE PASAN POR TODOS LOS LINEAS

iii) Escribir un test suite que cubra al menos 50% de las Branches. B

- En particular, el test suite dado en (ii) cubre el 100% de las branches.

$$\text{cobertura} = \frac{\#\{\text{Branches cubiertas}\}}{2 \#\{\text{Branches en el programa}\}}$$

TEST SUITE

```
ASSERT APROBADO ([3]) == 3
ASSERT APROBADO ([2,0]) == 1
ASSERT APROBADO ([6,6]) == 2
ASSERT APROBADO ([4,4]) == 2
```

---

iv) Explicar el error en la implementación:

- LA LÍNEA 9 DEBERÍA DEVOLVER 1 (RETURN 1)
- LA LÍNEA 11 DEBERÍA SER (IF SUMA_NOTAS >= 4 * len(notas))

Si, el test suite del punto (A) detecta ambos errores... B

- ASSERT APROBADO ([6,10]) == 1 falla y devuelve 2.
- ASSERT APROBADO ([4,4]) == 2 falla y devuelve 3.

Ej 4 Sebastián Antonio González Hoja 5

Ej 4)

i) Dados P_1, P_2 ...

- Si A es más fuerte que B, luego (A ⇒ B) es tautología, por lo que cumplir el requerimiento P_1 implica cumplir el requerimiento de P_2 ...

(B (A))

Ej: pre ---- post
A: {x = 1} --- {res = 1}
B = {x ∈ N} --- {res = 1}

- No obstante, cualquier algoritmo que satisface P_1 no cumple necesariamente P_2. Porque sólo es válido la postcondición C para el "dominio" de P_1 (A) que es un subconjunto del "dominio" de P_2 (B).

B

- En mi ejemplo:

- Problema P_1(x: Int): Int {
  req {x = 1}
  asg {res = 1}
  }
- Problema P_2(x: Int): Int {
  req {x ∈ N}
  asg {res = 1}
  }

- Los algoritmos que satisfagan P_1 no tienen ninguna garantía para (∀x ∈ N)(x ≠ 1).

- El caso contrario ...

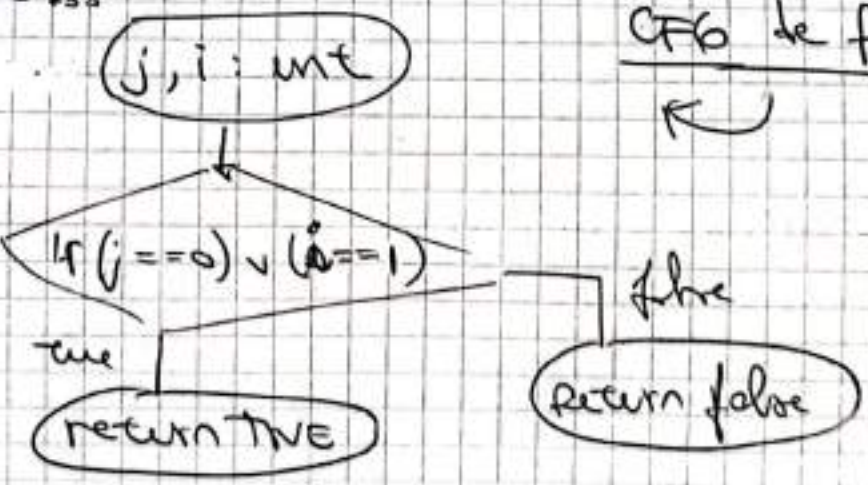
⟶

- El caso contrario sí vale, pues A es un caso particular de B. Entonces como P_2 garantiza que $\forall x \in B \rightarrow$ [illegible] y $A \subseteq B$ cumplir la especificación de P_2 implica cumplir la especificación de P_1. B
(OBS: tiene mínimo seguro C)

(ii) ¿Es posible un test suite con 100% cubrimiento de nodos y 0 fallos pero con 1 bug?

OBS. Cubrir todos los nodos NO implica cubrir todas las DECISIONES BÁSICAS. ✓

por lo tanto, sí es factible. A continuación 1 ejemplo...



Test Suite (todos los nodos): || ASSERT f(0,3) == TRUE
ASSERT f(4,1) == FALSE
antes TRUE

- CON LA ESPECIFICACIÓN PROBLEMA ALGUNOS O (j, i: ~~Int~~) : Bool {
req {true}
ens {res = TRUE ⇔ (j=0 v i=0)}
}

MÁS PRODIJO →

Ej 4 — Sebastián Andres — 10/08/22 — hoja 6

ES DECIR, SI SE PUEDE.
El test suite cubre todos los nodos del programa que implementa la función f(j,i) y no falla.
Sin embargo f(x,0) ≠ true, luego no cumple la especificación algunoEsO(x,y).
Por lo tanto hay uno falla que el test suite posee (cubrir todos los ~~Nodos~~) no encontró.

Programa en Python:

```
DEF f(j:INT, i:INT) -> BOOL:
  IF (j==0) or (i==1):
      return true
  ELSE:
      return false
```

Test Suite (nodos)

```
ASSERT f(0,3) == TRUE
ASSERT f(1,1) == false
```

(Test agregado)

```
ASSERT f(1,0) == true
```

falla

Especificación

```
Aridad algunoEsO(x,y: ~~INT~~) -> Bool {
  requiere {true}
  asegura {res = true <-> (x=0 v y=0)}
}
```

Este no el ejemplo pero dice la idea

B-