Boccio Sebastián

ord 29

Ingeniería del Software II

Parcial #1 – Generación Automática de Tests

Ej1	Ej2	Ej3	Ej4	Nota
R	B	B	B	A

El exámen es a **libro abierto**. Cada ejercicio se evaluará como **Bien**, **Regular** o **Mal**. Para aprobar el examen es necesario tener al menos 2 ejercicios Bien y al menos 1 ejercicio Regular.

Bien = 2,5p, Regular = 1,25p, Mal = 0p.

Ejercicio 1

Sea el siguiente programa:

```
1  def test_me(j: int) -> int:
2      i:int=0
3      r:int=0+0
4      while i<2: # C1
5          if i==j: # C2
6              r=r+1
7          i=i+1
8
9      return r
```

Usando el operador de mutación AOR[1], exhibir dos mutantes (indicando la línea que cambió) tales que:

a. Un mutante con un test que lo mate. Indique cual es el test case que lo detecta.

b. Un mutante equivalente tal que no exista test que lo detecte.

Ejercicio 2

Sea el programa `test_me` del ejercicio #1.

a. Completar la siguiente tabla con la ejecución simbólica dinámica del programa de forma manual, indicando para cada iteración:

- El input concreto utilizado
- La condición de ruta (i.e. "path condition") que se produce al ejecutar el input concreto, asumiendo que el valor simbólico inicial es $j = j_0$.
- La fórmula lógica (no es necesario escribirla en SMTLib) que se envía al demostrador de teoremas de acuerdo al algoritmo de ejecución simbólica dinámica.
- El resultado posible que podría producir un demostrador de teoremas (ej: Z3).

b. Describir el árbol de cómputo del programa explorado durante la ejecución simbólica dinámica del programa

Iteración	Input Concreto	Condición de Ruta	Fórmula enviada al demostrador	Resultado posible
1	j=0	...	...	...
2	...	...	...	...
...	...	...	...	...

[1] modifica una operador artimético reemplazándolo por $+, -, *, /$ y $\%$

Ejercicio 3

Sea el programa **test_me** del ejercicio #1 y el siguiente test suite:

```
class TestSuite(unittest.TestCase):
  def test_1(self):
    self.assertEqual(0, test_me(10))

  def test_2(self):
    self.assertEqual(0, test_me(5))

  def test_3(self):
    self.assertEqual(0, test_me(7))
```

a. Sea K=5, ¿cuál es el valor de la distancia de branch no normalizada para cada decisión si ejecutamos el test suite?

branch	distancia true	distancia false
4: `while i<2:`		
5: `if i==j:`		

b. ¿Cuál es el cubrimiento de branches que logra el test suite?

Ejercicio 4

Sea el siguiente programa:

```
1  def parse_url_manual(url):
2      scheme = ""
3      netloc = ""
4      path = ""
5      params = ""
6      query = ""
7      fragment = ""
8
9      # Split the URL into parts
10     if "://" in url:
11         scheme, url = url.split("://", 1)
12     if "/" in url:
13         netloc, url = url.split("/", 1)
14     if "?" in url:
15         path, query = url.split("?", 1)
16     elif "#" in url:
17         path, fragment = url.split("#", 1)
18     else:
19         path = url
20
21     # Split query into parameters
22     query_params = {}
23     if query:
24         for param in query.split("&"):
25             key, value = param.split("=") if "=" in param else (param, "")
26             query_params[key] = value
27
28     return [scheme, netloc, path, params, query, fragment]
```

Asumiendo que tenemos un boosted greybox fuzzer con exponente a=3. Sea el siguiente conjunto inicial de inputs:

a. • `https://www.example.com/path/to/resource`
b. • `http://example.com/page?p1=v1&p2=v2`
c. • `ftp://ftp.example.com/resource#section1`
d. • `https://www.example.com/path?p1=v1`
e. • `https://www.example.com/path?p2=v2`
f. • `https://www.example.com/path?p3=v3`

¿Cuál es la probabilidad que el fuzzer elija el input `"https://www.example.com/path?p3=v3"` para mutar?

BOCACCIO SEBASTIÁN 1
287/18
29

Ejercicio 1:

b) Un mutante equivalente se obtiene realizando una mutación en la línea 3 como

__ r: int = 0 + 0__ cambiando el operador "+" por "*"

Para este caso, dado que 0 = 0 + 0 = 0 * 0 el comportamiento es idéntico y no existe Test posible que pueda matarlo. ✓

a) Para este caso, podemos mutar el "<" de la línea 4 por un ">"
Quedando: ✗ se pide mutar AOR

__ while i > 2 __

Notemos que como i empieza con valor 0 esta condición ~~nunca~~ no es verdadera y no va a ejecutarse el body del while.

Un test que lo mate sería:

```
Def test1(self):
    self.assertequals(1, testme(1)
```

En este caso, el mutante devuelve 0 ya que no crece el valor de r al no entrar nunca en el while. En el código original esto no sucede. y cuando i = 1 se cumple que i == j y r va a pasar a valer 1. Por esto, el assert va a fallar haciendo que falle el test

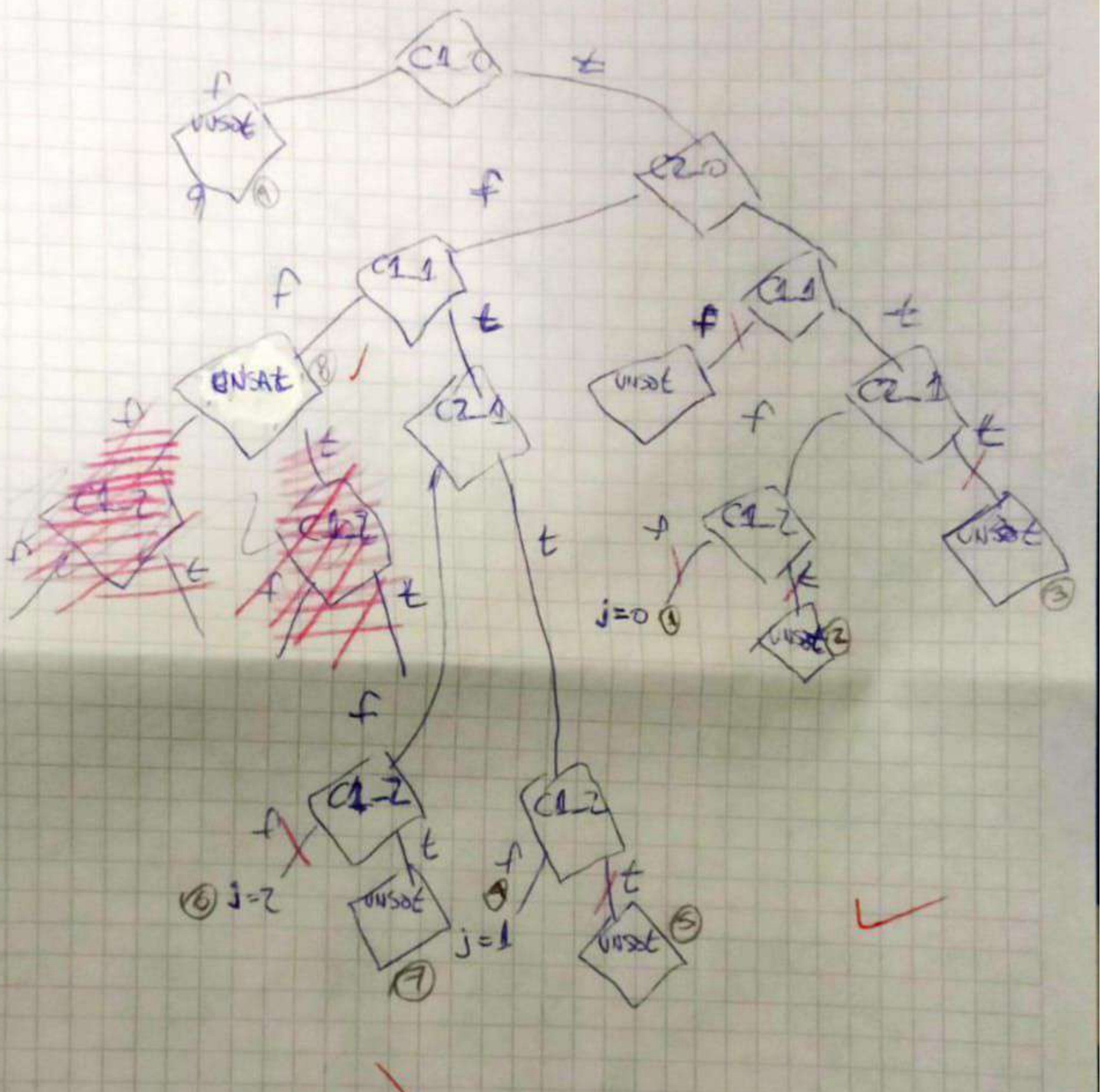
Ejercicio 2:

a)

I	Input concreto	Condición de ruta	Fórmula para Z3	Resultados posibles	
1	j=0	$c1_0 \neg c2_0 c1_1 \neg c2_1 \neg c1_2$	$c1_0 \neg c2_0 c1_1 \neg c2_1 c1_2$	UNSAT	
			$c1_0 \neg c2_0 c1_1 c2_1$	j=1	
2	j=1	$c1_0 \neg c2_0 c1_1 c2_1 \neg c1_2$	$c1_0 \neg c2_0 c1_1 c2_1 \neg c1_2$	UNSAT	
			$c1_0 \neg c2_0 \neg c1_1$	UNSAT	
3	j=2	$c1_0 \neg c2_0 \neg c1_1$	$c1_0 c2_0$		
1	j=0	$c1_0 c2_0 c1_1 \neg c2_1 \neg c1_2$	$c1_0 c2_0 c1_1 \neg c2_1 c1_2$	UNSAT	①
			$c1_0 c2_0 c1_1 c2_1$	UNSAT	②
			$c1_0 c2_0 \neg c1_1$	UNSAT	③
2	j		$c1_0 \neg c2_0$	j=1	④
2	j=1	$c1_0 \neg c2_0 c1_1 c2_1 \neg c1_2$	$c1_0 \neg c2_0 c1_1 c2_1 c1_2$	UNSAT	⑤
			$c1_0 \neg c2_0 c1_1 \neg c2_1$	~~UNSAT~~ j=2	⑥
3	j=2	$c1_0 \neg c2_0 c1_1 \neg c2_1 \neg c1_2$	$c1_0 \neg c2_0 c1_1 c2_1 c1_2$	UNSAT	⑦
			$c1_0 \neg c2_0 \neg c1_1$	UNSAT	⑧
			$\neg c1_0$	UNSAT	⑨
	FIN	FIN	FIN		

Bocacci[?]
[?]
237/18
29
Ejercicio 2

b)



Bocacio Sebastián
28718
29

Ejercicio 3:

a)

Branch	distance true	Distance false
4: while i<2	0	0
5: if i==j	4	0

✓

Justifiquemos un poco, 1) Para el caso del while es 0 ya que en el primer test ya se evaluam como en false como en true. Esto es así ya que para i=0 entra y para i=2 no entra.

2) Para el caso de if i==j, en ningún test se va a cumplir la guarda porque i solo puede valer 1 o 0. Por eso la distancia false es 0. Por otro lado, en el test_2 cuando i=1 es cuando se minimiza el valor de la distancia de true ya que j=5 i=1 j= 4= ABS(5-1)

b) ~~Todos que se cubrieran~~ ✓

El cubrimiento de branches que se logra es de 3/4. ✓

Ejercicio 4:

Nota: A/D Llamo al conjunto de inputs iniciales a, b, c, d, e, f ~~Para~~ respectivamente para facilitar la ~~estructura~~ escritura

Ahora, voy a agrupar los inputs según los caminos que recorren

I a — $I = \{a\}$ — I

II b — $II = \{b\}$ — II

III c — $III = \{c\}$ — III

IV d e f — $IV = \{d, e, f\}$ — IV

Notemos que me están pidiendo la probabilidad de que fuzzer elija el input f

Para calcular la probabilidad de f tenemos que calcular primero la energía de cada input. Usando la definición:

$$e(s) = \frac{1}{f(p(s))^a}, \text{ con } a = 3$$

Ahora bien

$e(a) = 1$
$e(B) = 1$
$e(C) = 1$
$x \in IV$, $e(x) = \frac{1}{3^3} = \frac{1}{27}$

Con esto, ya podemos hacer los cálculos.

- sea $S = I \cup II \cup III \cup IV$
- sea P el evento = "el fuzzer elige el input f para mutar"

$$P(P) = \frac{e(f)}{\sum_{s \in S} e(s)} = \frac{\frac{1}{9}}{1+1+1+3\cdot\frac{1}{27}} = \frac{\frac{1}{9}}{3+\frac{1}{3}} = \frac{\left(\frac{1}{9}\right)}{\left(\frac{10}{3}\right)} = \frac{3}{90} = \frac{1}{30}$$

1/84