Name : Run Zeng

Student ID : 20794087

Email : r24zeng@uwaterloo.ca

Question 1

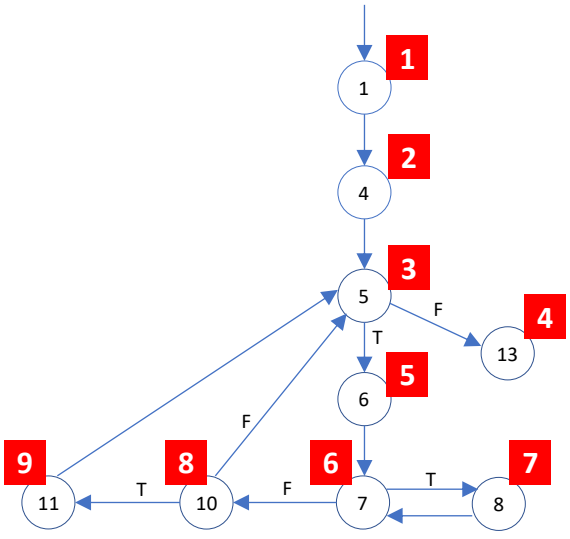
a) The null value for arr will make the state stop at the first " for " loop state as range(0,0).
Input : arr=null
Expected output : 0
Actual output : 0

b) Once the fault statement is executed, which means len(arr)>=2. It must result in less loops for j than expectation. Therefore, the error must happen.

c) If the last element is negative, which means the expected result does not depend on the last number in arr, the error state happens but no failure.
Input : arr = [1, 2, 7, 12, -6, -3]
Expected output : 22
Actual output : 22

d) The length of array x is 4. When i = 1, the expectation for j loop is to sum the first element in x. Because the error of this statement, j should be assigned 0 but is not assigned value. So this is the first error happen.
Input : arr=[4, 0, -2, 3]
Expected output : 5
Actual output : 4
First error state:
— arr=[4, 0, -2, 3]
— i= 0
— j= undefined
— res = 0
— temp = 0
— PC = if temp > res

e)



Question 2

(a)

```
class RepeatUntilStmt (object):
    def _init_ (self, cond, rept_stmt, inv = none):
        self.cond = cond
        self.rept_stmt = rept_stmt
        self.inv = inv
```

(b)

The formalized semantics are as follows:

$$\text{If } B[b]s=ff, \quad \frac{\langle S, q\rangle \Downarrow s' \qquad \langle S; \text{repeat S until b}, s'\rangle \Downarrow s''}{\langle \text{repeat S until b}, q\rangle \Downarrow s''}$$

$$\text{If } B[b]s=tt, \quad \frac{\langle S, q\rangle \Downarrow s'}{\langle \text{repeat S until b}, s'\rangle \Downarrow s'}$$

(c)

The formula can be divided as follow steps:

$$\frac{\dfrac{\langle x:=2, [\,]\rangle \Downarrow x:=2 \qquad \langle b, x:=2\rangle \Downarrow \text{false}}{\langle x:= x-1; \text{repeat x:=x-1until x<=0}, x:=2\rangle \Downarrow x:=0} \qquad \langle b, x:=0\rangle \Downarrow \text{true}}{\langle \text{repeat x:=x-1 until x<=0}, x:=0\rangle \Downarrow x:=0}$$

Therefore, the judgement is valid.

(d)

If B[b]=ff, according to (b), $\langle \text{repeat S until b}, q\rangle \Downarrow s''$

$$\frac{\langle S, s\rangle \Downarrow s' \qquad \langle \text{repeat S until b}, s'\rangle \Downarrow s''}{\langle \text{repeat S until b}, q\rangle \Downarrow s''}$$

We assume $\langle$ repeat S until b, s' $\rangle$ equals to $\langle$ S; while not b do S, s' $\rangle$, T1 = $\langle$ S, q $\rangle \Downarrow$ s', T2 = $\langle$ repeat S until b, s' $\rangle \Downarrow$ s''. Therefore, T2 = $\langle$ S; while not b do S, s' $\rangle$. T2 can be divided to "$\langle$ S, s' $\rangle \Downarrow$ s1', $\langle$ while not b do S, s1' $\rangle \Downarrow$S''. Then merge T1 and T2 to be $\langle$ S, s$\rangle \Downarrow$s1'. At last, merge it with while. It will be $\langle$ S; while not b do S, s $\rangle \Downarrow$s''. Therefore $\langle$ S; while not b do s$\rangle$ is equal to $\langle$ repeat S until b $\rangle$.

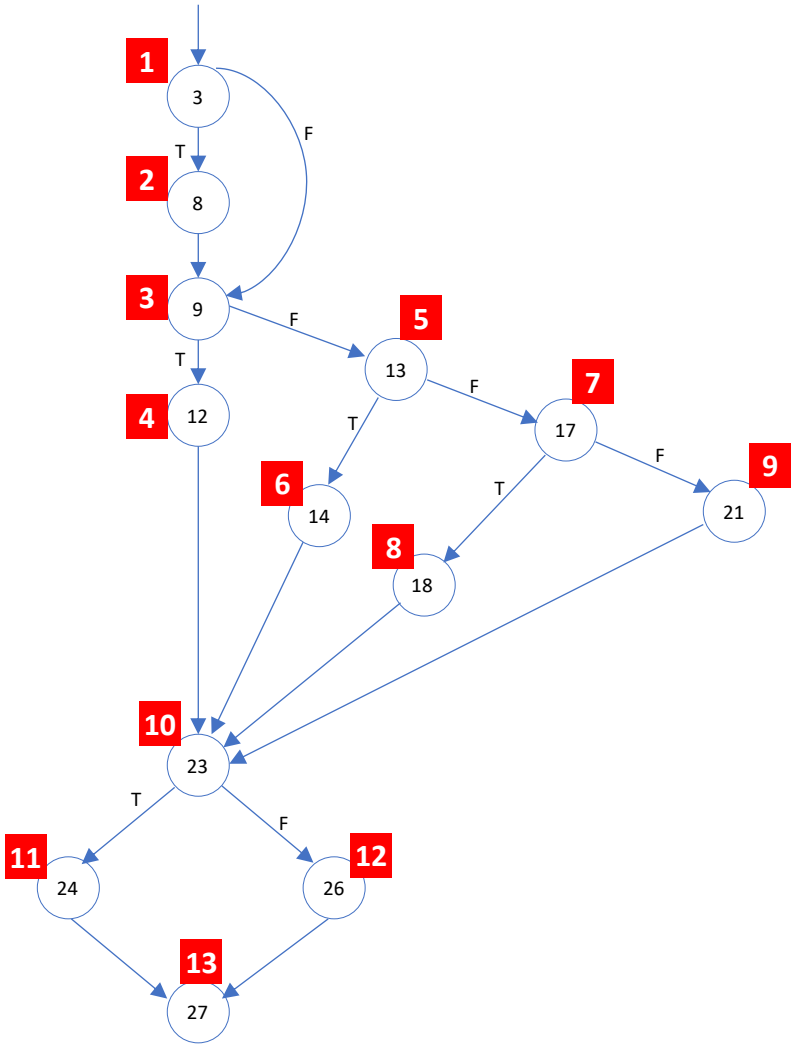
If B[b]=tt, according to (b), $\langle \text{repeat S until b}, q\rangle \Downarrow s'$

$$\frac{\langle S, q\rangle \Downarrow s'}{\langle S; \text{skip}, s\rangle \Downarrow s'}$$

Obviously, < S; skip, s > equals to < S; while not b do s, s >. Therefore when B[b]=tt, these two statements are same semantically.

Conclusion: the two statements are semantics equivalent.

3.(a)



(b)

TR_{NC}:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}

TR_{EC}:

{[1, 2], [2, 3], [1, 3], [3, 4], [3, 5], [5, 6], [5, 7], [7, 8], [7, 9], [4, 10], [6, 10], [8, 10], [9, 10], [10, 11], [10, 12], [11, 13], [12, 13]}

TR_{EPC}:

{[1, 2, 3], [1, 3, 4], [1, 3, 5], [2, 3, 4], [2, 3, 5], [3, 4, 10], [3, 5, 6], [3, 5, 7], [4, 10, 11], [4, 10, 12], [5, 6, 10], [5, 7, 8], [5, 7, 9], [6, 10, 11], [6, 10, 12], [7, 8, 10], [7, 9, 10], [8, 10, 11], [8, 10, 12], [9, 10, 11], [9, 10, 12], [10, 11, 13], [10, 12, 13] }

TR_{PPC}:
{[1, 2, 3, 4, 10, 11, 13], [1, 2, 3, 4, 10, 12, 13], [1, 2, 3, 5, 6, 10, 11, 13], [1, 2, 3, 5, 6, 10, 12, 13], [1, 2, 3, 5, 7, 8, 10, 11, 13], [1, 2, 3, 5, 7, 8, 10, 12, 13], [1, 2, 3, 5, 7, 9, 10, 11, 13], [1,2, 3, 5, 7, 9, 10, 12, 13], [1, 2, 3, 4, 10, 12, 13], [1, 3, 4, 10, 12, 13], [1, 3, 5, 6, 10, 11, 13], [1, 3, 5, 6, 10, 12, 13], [1, 3, 5, 7, 8, 10, 11, 13], [1, 3, 5, 7, 8, 10, 12, 13], [1, 3, 5, 7, 9, 10, 11, 13], [1, 3, 5, 7, 9, 10, 12, 13]}

Because in semantically, when n=0, it can only execute path 4-10-12, other values of n can't execute note 4 but can only execute note 11. Therefore those nodes coverage are infeasible. The infeasible TR are as follows:
In **TR_{EPC}**: {[4, 10, 11], [6, 10, 12], [8, 10, 12], [9, 10, 12]}
In **TR_{PPC}**:
{[1, 2, 3, 4, 10, 11,13], [1, 2, 3, 5, 6, 10, 11, 13], [1, 2, 3, 5, 7, 8, 10, 11, 13], [1, 2, 3, 5, 7, 9, 10, 11, 13], [1, 3, 4, 10, 12, 13], [1, 3, 5, 6, 10, 12, 13], [1, 3, 5, 7, 8, 10, 12, 13], [1, 3, 5, 7, 9, 10, 12, 13]}

(c)
According to (b), TR_{NC} but not TR_{EC} (never go through [1, 3]) :
[1, 2, 3, 4, 10, 12, 13], [1, 2, 3, 5, 6, 10, 11, 13], [1, 2, 3, 5, 7, 8, 10, 11, 13], [1, 2, 3, 5, 7, 9, 10, 11, 13]

TR_{EC} but not TR_{EPC}(never go through [2, 3, 5]):
[1, 3, 5, 6, 10, 11, 13], [1, 2, 3, 4, 10, 12, 13], [1, 3, 5, 7, 8, 10, 11, 13], [1, 3, 5, 7, 9, 10, 11, 13]

TR_{EPC} but not TR_{PPC}(never go through[2, 3, 5, 6]):
[1, 2, 3, 4, 10, 12, 13], [1, 3, 5, 6, 10, 11, 13], [1, 3, 4, 10, 12, 13], [1, 2, 3, 5, 7, 8, 10, 11, 13], [1, 2, 3, 5, 7, 9, 10, 11, 13]

Test Path of Prime Coverage is :
[1, 2, 3, 4, 10, 12, 13], [1, 3, 4, 10, 12, 13], [1, 2, 3, 5, 6, 10, 11, 13], [1, 3, 5, 6, 10, 11, 13], [1, 2, 3, 5, 7, 8, 10, 11, 13], [1, 3, 5, 7, 8, 10, 11, 13], [1, 2, 3, 5, 7, 9, 10, 11, 13], [1, 3, 5, 7, 9, 10, 11, 13]

4.(b)
can't coverage:
int.py : 35
because function _repr_( ) doesn't have return value

int.py : 58
Because relation expression doesn't have other operations. Once use other illegal notations, this node can't be recognized as RelExpression, so "assert False" can not be executed.

int.py : 172-189
Because these three blocks represent how to accept visitor. Only inside of the module can visit those blocks. So it's impossible to cover those.

paser.py : 114
Because function __stmt__( ) contains all feasible statements. When given other illegal statements, interpreter recognized it as an error. Then this execution will fail. self._error step will never not be executed.

paser.py : 168, 169
Because "while do " doesn't have "condition is false then execute other option", therefore there is no more bool expression.

paser.py : 264
Because "bfactor" only can be "not". Any other character will not be recognized as legal operation. So the fault statement will result to failure directly but not execute "self_error".

paser.py : 448, 449
When parser encounters newline character, it can only make statements separately but not start a new line.

parser.py : 556-573
Because these two blocks represent how to accept visitor. Only inside of the module can visit those blocks. So it's impossible to cover those.