Assignment 1 Report

Hao Dong, 20757585, h45dong@uwaterloo.ca

Question 1

(a) If possible, identify a test case that does not execute the fault.

The null value for arr will result in a NullPointerException before the loop test is evaluated, hence no execution of the fault. Also, if arr is empty, the fault will not be executed either.

- Input: arr = null;
- Expected Output: TypeError: object of type 'NoneType' has no len()
- Actual Output: TypeError: object of type 'NoneType' has no len()

(b) If possible, identify a test case that executes the fault, but does not result in an error state.

There doesn't exist the test case which does not result in an error state when executing the fault. Because once line(5) is executed, it must result in an error state.

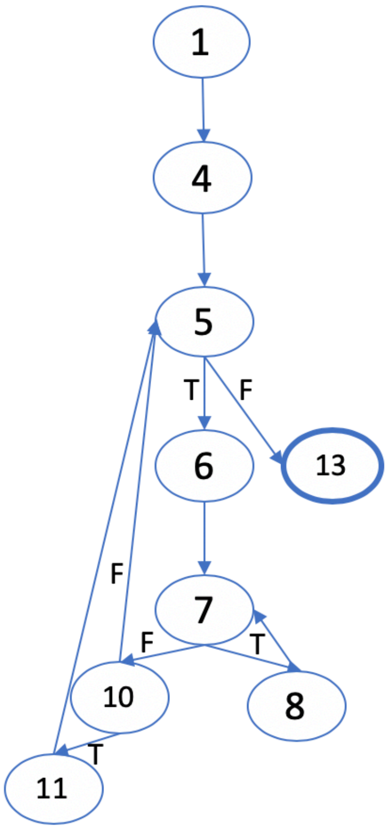
(c) If possible, identify a test case that results in an error, but not a failure.

- Input: arr = [0,0,0];
- Expected Output: 0
- Actual Output: 0

(d) For the test case x = [4, 0, -2, 3] the expected output is 5. Identify the first error state. Describe the complete state.

- Input: x = [4, 0, -2, 3]
- Expected Output: 5
- Actual Output: 4
- First Error State:
  - x = [4, 0, -2, 3]
  - i = 0
  - j = 0
  - PC = if temp > res:

(e) CFG



Question 2

(a)

```python
class RepeatUntilStmt(Stmt):
    """Repeat-until statement"""
    def __init__(self):
        self.cond = cond
        self.stmt = stmt
```

(b)

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

$$\frac{<S, q> \Downarrow q' \quad <b, q'> \Downarrow true}{<repeat\ S\ until\ b, q> \Downarrow q'}$$

(c)

$$\cfrac{\cfrac{<2, [\,]> \Downarrow 2}{<x:=2, [\,]> \Downarrow [x:=2]} \quad \cfrac{<x:=x-1, [x:=2]> \Downarrow [x:=1) \quad <x \leq 0, [x:=1]> \Downarrow false \quad <repeat\ x:=x-1\ until\ x \leq 0, [x:=1]> \Downarrow [x:=0]}{<repeat\ x:=x-1\ until\ x \leq 0, [x:=2]> \Downarrow [x:=0]}}{<x:=2;\ repeat\ x:=x-1\ until\ x \leq 0, [\,]> \Downarrow [x:=0]}$$

(d) We need to prove: $< repeat\ S\ until\ b,\ q > \rightarrow q' \quad \text{iff} \quad < S; while\ \neg b\ do\ S,\ q > \rightarrow q'$

We can prove by **induction** that:

$\text{if} \quad < repeat\ S\ until\ b,\ q > \rightarrow q' \quad \text{then} \quad < S; while\ \neg b\ do\ S,\ q > \rightarrow q'$ (and vice versa)

,showing that each step preserves the property:

Base case:

- if $B(b)$ = false, which means $B(\neg b)$ = true

  $$\frac{<S, q> \Downarrow q' \quad <b, q'> \Downarrow false \quad <repeat\ S\ until\ b, q'> \Downarrow q''}{<repeat\ S\ until\ b, q> \Downarrow q''}$$

  By the induction hypothesis, $< repeat\ S\ until\ b,\ q' > \rightarrow q''$ is equivalent to $< S;\ while\ \neg b\ do\ S,\ q' > \rightarrow q''$

  Therefore, we can deduce like:

  $$\cfrac{<S, q> \Downarrow q' \quad \cfrac{\overset{<\neg b, q'> \Downarrow true}{\boxed{<b, q'> \Downarrow false}} \quad \overset{<S; while\ \neg b\ do\ S, q'> \Downarrow q''}{\boxed{<repeat\ S\ until\ b, q'> \Downarrow q''}}}{<while\ \neg b\ do\ S, q'> \Downarrow q''}}{<S;\ while\ \neg b\ do\ S,\ q> \Downarrow q''}$$

- Similarly, if $B(b)$ = true, which means $B(\neg b)$ = true

$$\frac{<S, q> \Downarrow q' \quad <b, q'> \Downarrow true}{<repeat\ S\ until\ b, q> \Downarrow q'}$$

By the induction hypothesis, $< repeat\ S\ until\ b,\ q > \rightarrow q'$ is equivalent to $< S;\ while\ \neg b\ do\ S,\ q > \rightarrow q'$

$$\frac{<S, q> \Downarrow q' \qquad \dfrac{<\neg b, q'> \Downarrow false}{<while\ \neg b\ do\ S, q'> \Downarrow q'}}{<S;\ while\ \neg b\ do\ S, q> \Downarrow q'}$$

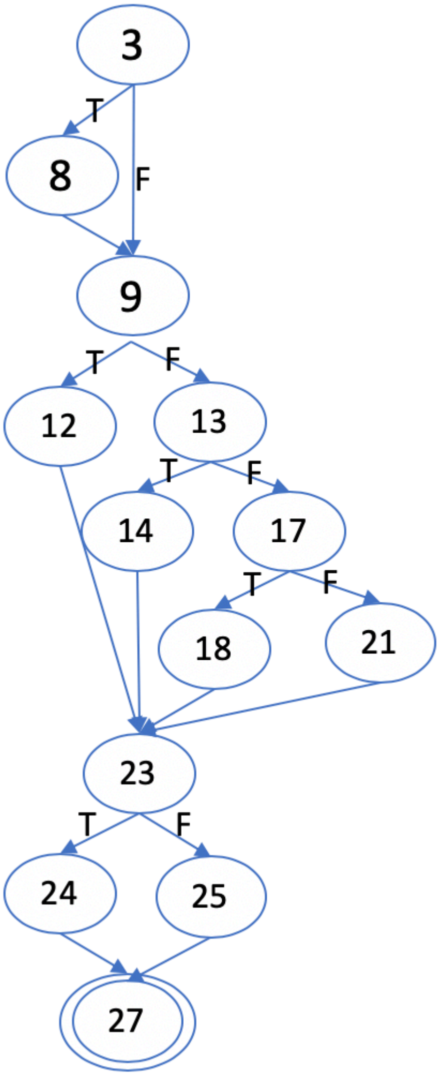
So in both cases,

$\text{if} \quad < repeat\ S\ until\ b,\ q > \rightarrow q' \quad \text{then} \quad < S; while\ \neg b\ do\ S,\ q > \rightarrow q'$ (and vice versa)

The induction step for compositional trees works is now trivial, we assume that for n applications of the repeat-rules, we can use n-applications of the while-rule plus the additional rule for the composition to gain the same result

Question 3

(1)



(2)

The Test Requirements for the Basic Block CFG is listed below.

TR_{NC} = {[3, 8, 9, 12, 13, 14, 17, 18, 21, 23, 24, 25, 27]}

TR_{EC} = {[3,8], [3,9], [8,9], [9,13], [9,12], [13,17], [13,14], [17,21], [17,18], [12,23], [14,23], [18,23], [21,23], [23,25], [23,24], [25,27], [24,27]}

TR_{EPC} = {[3,8,9], [3,9,13], [3,9,12], [8,9,12], [8,9,13], [9,13,17], [9,13,14], [9,12,23], [12,23,25], [13,17,21], [13,17,18], [13,17,21], [13,14,23], [14,23,24], [17,21,23], [17,18,23], [18,23,24], [21,23,24], [23,25,27], [23,24,27]}

$InfeasibleEdgePair$ = {[12,23,24],[14,23,25], [18,23,25], [21,23,25]}

Because node 12(which indicates n = 0) and 24(which indicates n > 0) can not happen at the same path, containing them both is inifeasible. Besides, since n is the length of arr, n >= 0, which means that if node 12 doesn't appear, 24 must be contained.

TR_{PPC}= { [3,9,12,23,25,27], [3,8,9,12,23,25,27], [3,8,9,13,14,23,24,27], [3,9,13,14,23,24,27], [3,8,9,13,17,18,23,24,27], [3,9,13,17,18,23,24,27], [3,8,9,13,17,21,23,24,27], [3,9,13,17,21,23,24,27] }

$InfeasiblePrimePaths$ = { [3,9,13,17,21,23,25,27], [3,8,9,13,17,21,23,25,27], [3,8,9,13,14,23,25,27], [3,9,13,14,23,25,27], [3,8,9,13,17,18,23,25,27], [3,9,13,17,18,23,25,27], [3,9,12,23,24,27], [3,8,9,12,23,24,27] }

Question 4

Explanation for each line that was not covered:

Int.py:68

For RelExp, there is no possible to turn to assert False

parse.py:114

For _stmt_, there is no possible to turn to self._error

parse.py:264

For _bfactor_, there is no possible to turn to self._error

parse.py:448

It's difficult to create a newline