Assignment 1

Name: Zhen Gong
Student ID: 20673670
Email: z33gong@uwaterloo.ca

Q1

a)

Test case: a = 1, b = 2.

The inputs are not valid, thus the program will crash before executing line 8. Therefore, it does not execute the fault

b)

Test case: a = [[1,2], [3,4]], b = [[5,6], [7,8]]

The fault at line 8 will be executed. Since b is a 2 by 2 matrix, no error will occur when assigning q and p1.

c)

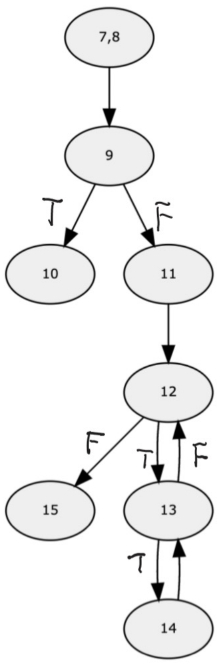
Test case: a = [[1,2], [3,4]], b = [[5], [6], [7]]

The program will have error since it should be q=1 p1=3 as intended, but got q=3, p1=1 due to the fault. However, there will be no failure because the program will raise ValueError and output message "Incompatible dimension" as we expected.

d)

n = 2
p = 2
q = 2
p_1 = 1
c = undefined
i = undefined
j = undefined
PC = at line 10

e)



Assume a for loop is one node, 'T' means enter the loop, 'F' means exit the loop.

Q2

a)

```python
class RepeatUntilStmt(Stmt):
    def __init__(self, cond, body, inv=None):
        self.cond = cond
        self.body = body
        self.inv = inv

    def __eq__(self, other):
        return (
            type(self) == type(other)
            and self.cond == other.cond
            and self.body == other.body
            and self.inv == other.inv
        )
```

b)

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

c)

$$\dfrac{\dfrac{\dfrac{}{\langle 2, [\,]\rangle \Downarrow 2}}{\langle x := 2, [\,]\rangle \Downarrow [x:=2]} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\langle x, [x:=2]\rangle \Downarrow 2} \quad \dfrac{}{\langle 1, [x:=2]\rangle \Downarrow 1}}{\langle x-1, [x:=2]\rangle \Downarrow 1}}{\langle x := x-1, [x:=2]\rangle \Downarrow [x:=1]} \quad \dfrac{\dfrac{}{\langle x, [x:=1]\rangle \Downarrow 1} \quad \dfrac{}{\langle 0, [x:=1]\rangle \Downarrow 0}}{\langle x \le 0, [x:=1]\rangle \Downarrow \text{false}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\langle x, [x:=1]\rangle \Downarrow 1} \quad \dfrac{}{\langle 1, [x:=1]\rangle \Downarrow 1}}{\langle x-1, [x:=1]\rangle \Downarrow 0}}{\langle x := x-1, [x:=1]\rangle \Downarrow [x:=0]} \quad \dfrac{\dfrac{}{\langle x, [x:=0]\rangle \Downarrow 0} \quad \dfrac{}{\langle 0, [x:=0]\rangle \Downarrow 0}}{\langle x \le 0, [x:=0]\rangle \Downarrow \text{true}}}{\langle \text{RPT\_UTL}, [x:=1]\rangle \Downarrow [x:=0]\rangle}}{\langle \text{RPT\_UTL}, [x:=2]\rangle \Downarrow [x:=0]}}{\langle x := 2;\ \underbrace{\text{repeat } x := x-1 \text{ until } x \le 0}_{\text{Let's call it RPT\_UTL}}, [\,]\rangle \Downarrow [x:=0]}$$

d)

Proof: The proof is in two parts, we shall first prove that if

$\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q''$ (*)

then

$\langle S; \text{if } b \text{ then skip else (repeat } S \text{ until } b), q\rangle \Downarrow q''$ (**)

Because (*) holds, we know that we have a derivation tree T for it, and it will have one of the two forms depend on the rule $[\text{repeat}^{tt}_{ns}]$ or $[\text{repeat}^{ff}_{ns}]$, where we defined that

(1)

$[\text{repeat}^{ff}_{ns}]$ when $\mathcal{B}[\![b]\!]q' = ff$

(2)

$[\text{repeat}^{tt}_{ns}]$ when $\mathcal{B}[\![b]\!]q' = tt$

In condition (1), the deviation tree has the form

$$\frac{\overset{T_1}{\langle S, q\rangle \Downarrow q'} \quad \overset{T_2}{\langle \text{repeat } S \text{ until } b, q'\rangle \Downarrow q''}}{\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q''}$$

Furthermore, $\mathcal{B}[\![b]\!]q' = ff$, T_1 and T_2 as the premises for rule $[\text{Comp}_{ns}]$ We can construct deviation tree

$$\frac{T_1 \qquad T_2}{\langle S; \text{repeat } S \text{ until } b, q\rangle \Downarrow q''}$$

Using $\mathcal{B}[b]q' = ff$, apply rule $[\text{if}^{ff}_{ns}]$ to construct deviation tree

$$\frac{\dfrac{T_1 \qquad T_2}{\langle S; \text{repeat } S \text{ until } b, q\rangle \Downarrow q''}}{\langle S; \text{if } b \text{ then skip else (repeat } S \text{ until } b), q\rangle \Downarrow q''}$$

Thereby showing that (**) holds

In condition (2), $\mathcal{B}[\![b]\!]q' = tt$, using the rule of $[\text{repeat}^{tt}_{ns}]$, we get derivation tree

$$\frac{\overset{T_1}{\searrow}\langle S, q\rangle \Downarrow q''}{\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q''} \qquad ③$$

Similarly, we use rule $[\text{if}^{tt}_{ns}]$ to construct the derivation tree by showing (✱✱) holds. ③ and ④ are equivalent since $[\text{skip}_{ns}]$ axiom guarantees $q' = q''$

$$\frac{\overset{T_1}{\searrow}\langle S, q\rangle \Downarrow q' \qquad \dfrac{\langle \text{skip}, q'\rangle \Downarrow q''}{\langle \text{if } b \text{ then skip else (repeat } S \text{ until } b), q'\rangle \Downarrow q''}}{\langle S; \text{if } b \text{ then skip else (repeat } S \text{ until } b), q\rangle \Downarrow q''} \qquad ④$$

This completes the first part of the proof.

For the second part of the proof, we assume (**) holds and prove (*) shall also hold. Similarly, there are two rules (1) $[\text{if}^{ff}_{ns}]$ and (2) $[\text{if}^{tt}_{ns}]$

In condition (1),

(*) can be written as $\langle S;\text{repeat } S \text{ until } b, q\rangle \Downarrow q''$

The statement has general form $S_1;S_2$ and use $[\text{comp}_{ns}]$ rule, we get the derivation trees T_2 and T_3

T_2: $\langle S, q\rangle \Downarrow q'$

T_3: $\langle \text{repeat } S \text{ until } b, q'\rangle \Downarrow q''$

It is straightforward to use $[\text{repeat}^{ff}_{ns}]$ rule to combine T_2 and T_3 into a derivation tree for (*)

In condition (2), $\mathcal{B}[\![b]\!]s = tt$ and T is constructed using rule $[\text{if}^{tt}_{ns}]$ we get

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

Use $[\text{comp}_{ns}]$ rule, it can be written as

T_2: $\langle S, q\rangle \Downarrow q'$

T_3: $\langle \text{skip}, q'\rangle \Downarrow q''$

Due to the axiom $[\text{skip}_{ns}]$, it must be the case $q' = q''$

Therefore the $[\text{repeat}^{tt}_{ns}]$ derivation tree can combine T_2 and T_3 such that

$$\frac{\langle S, q\rangle \Downarrow q' \quad \langle \text{skip}, q'\rangle \Downarrow q''}{\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q''}, \text{ where } q' = q''$$

and remove the $[\text{skip}_{ns}]$ part will not affect the equation, the derivation tree (*) also holds.

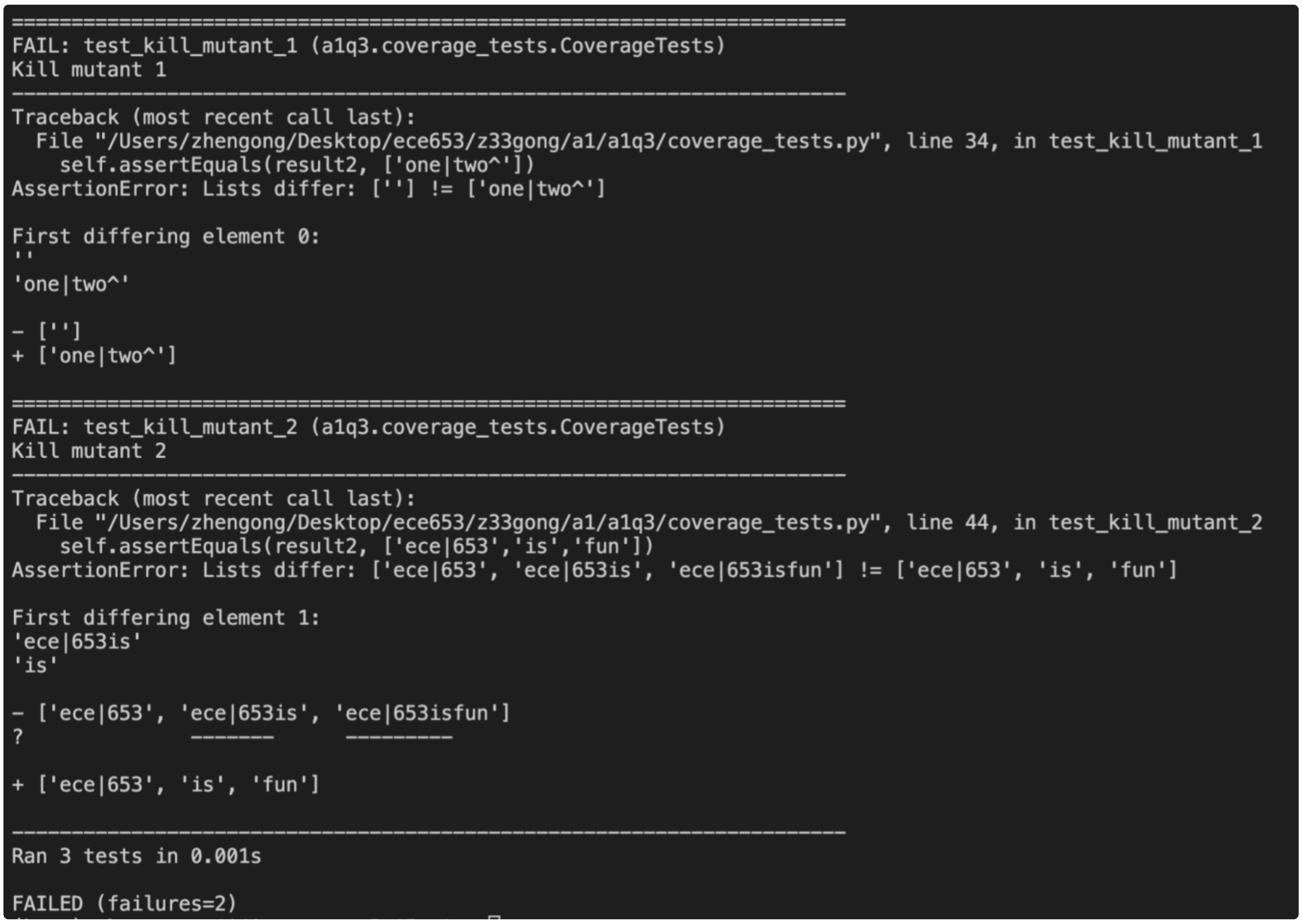
$$\frac{\langle S, q\rangle \Downarrow q''}{\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q''} \qquad (*)$$

This complete the proof.

Q3

c)

The picture shows the message print on the console when running the unit tests. It justifies that the unit tests kill each of the mutant and succeed on the original program.



```
======================================================================
FAIL: test_kill_mutant_1 (a1q3.coverage_tests.CoverageTests)
Kill mutant 1
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/zhengong/Desktop/ece653/z33gong/a1/a1q3/coverage_tests.py", line 34, in test_kill_mutant_1
    self.assertEquals(result2, ['one|two^'])
AssertionError: Lists differ: [''] != ['one|two^']

First differing element 0:
''
'one|two^'

- ['']
+ ['one|two^']

======================================================================
FAIL: test_kill_mutant_2 (a1q3.coverage_tests.CoverageTests)
Kill mutant 2
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/zhengong/Desktop/ece653/z33gong/a1/a1q3/coverage_tests.py", line 44, in test_kill_mutant_2
    self.assertEquals(result2, ['ece|653','is','fun'])
AssertionError: Lists differ: ['ece|653', 'ece|653is', 'ece|653isfun'] != ['ece|653', 'is', 'fun']

First differing element 1:
'ece|653is'
'is'

- ['ece|653', 'ece|653is', 'ece|653isfun']
?             -------      ---------

+ ['ece|653', 'is', 'fun']

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=2)
```

These test cases are useful. In part b) we have mutant program, and in part c), these additional unit tests give us more information about the testing performance on the original program and mutanted programs. Since the test results from the mutants to the original programs are different, which means the unit tests detected the faults in the mutants, then the mutants can be discarded, or killed.

Q4

a) Statement: Impossible/ Difficult to cover

ast.py:305-335
These functions cannot be covered by tests because they are extended from base class "Stmt". When we try to use AstVisitor to visit these classes, it will try to use getattr() to get the base class's name and thus trigger the error message 'AstVisitor' object has no attribute 'visit_Stmt' since "visit_Stmt" is not defined.

int.py:179-197
These lines includes the main function and function called in main. We don't need to write test to cover since a main method is a static method which makes unit testing without additional frameworks not possible.

parser.py: 452-453
Newline character is not a valid syntax to pass in, thus it is impossible to cover.

parser.py: 456-557
class WhileLangSemantics is never used in any part of the program in this assignment, I cannot cover these lines unless import parse in parser.py. In Piazza post @117, professor suggests it is better not import parse in practice so I choose to not cover these lines.

parser.py: 560-582
It is the same idea as **int.py:179-197**, main function cannot be cover using unittest only.

b) Branch: Impossible/ Difficult to cover

int.py: 90,111
The premise is that the input of the program must have valid operators, otherwise it will reach the line below it and raise an assertion error. Therefore, I choose not to cover them.

parser.py: 67
The "keywords" are set to None when the parser is initialized. Since I choose to not import Parser (reason as stated in part a), thus I cannot create an instance of WhilelangParser. Therefore, I did not cover this branch.

Conclusion: The interpreter can check the validation of the input by comparing it with the legal symbols in the dictionary, and I think the assert false in int.py is designed to prompt user when he passed in invalid input. Therefore, if we try to cover the branch by providing invalid input, we will get an assertionError.

Note:

Please notice that there will one assertion error when running the test cases because I want to cover int.py:155, which sugguests to try a false assertion.

int.py: 75
This might be a bug or design feature since assert false guarantees a failure, but I still provide a test case to cover it.