Zhengkang Chen
20789905
z585chen@uwaterloo.ca

Zhengkang Chen
20789905
z585chen@uwaterloo.ca

Q1:

a) Input a = None, b = [[1], [1]]. Expected output: TypeError. Actual output: TypeError.
It will throw TypeError at line 7 before the faulty line 8.

b) a = [[1, 1]], b = [[1, 1, 1]] Expected output: "Incompatible dimensions"
Actual output: "Incompatible dimensions"

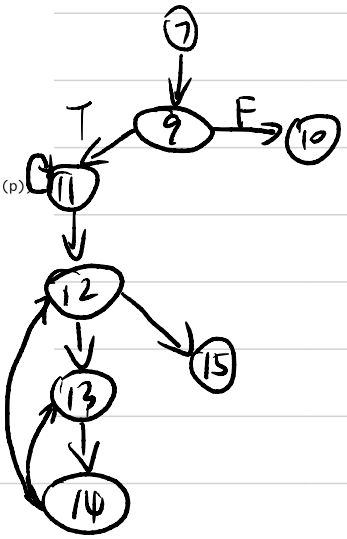
c) a = [[1]], b = [[1]]. Expected output: [[1]], Actual output: [[1]]

d) a = [[5, 7], [8, 21]], b = [[8], [4]] the expected output: [[68], [148]]

a = [[5, 7], [8, 21]] n = 2, p = 2 pc = if(p != p1);
b = [[8], [4]] q = 2, p1 = 1

e)

```
1  def matmul(a, b):
2      """
3      Returns the result of multiply two input matrices.
4
5      Raises an exception when the input is not valid.
6      """
7      n, p = len(a), len(a[0])
8      q, p1 = len(b), len(b[0])
9      if p != p1:
10         raise ValueError("Incompatible dimensions")
11     c = [[0] * q for i in range(n)]
12     for i in range(n):
13         for j in range(q):
14             c[i][j] = sum(a[i][k] * b[k][j] for k in range(p))
15     return c
16
17 # >>> a = [[5, 7], [8, 21]]
18 # >>> b = [[8], [4]]
19 # >>> matmul(a, b)
20 # [[68], [148]]
```



Q2: a)

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

b)

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

c)

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

d)

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

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

Comparing with b), they are semetically equivalent.

Q3

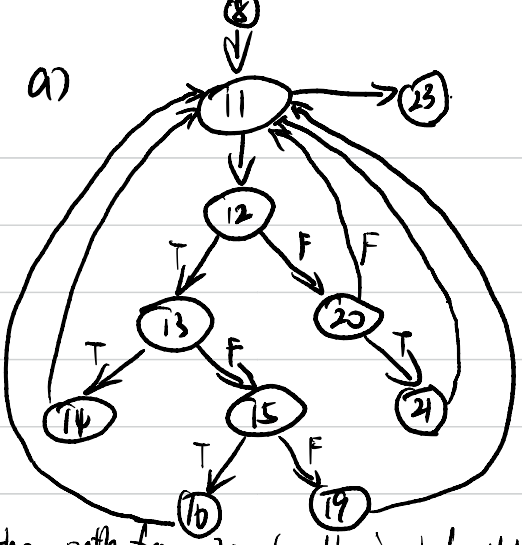
Question 3 (15 points)

Consider the following program from http://www.rosettacode.org/wiki/Tokenize.a.string.with.escaping:

```
def token_with_escape(inpt, escape="^", separator="|"):
    """
    Issue python -m doctest thisfile.py to run the doctests.

    >>> print(token_with_escape('one^|uno||three^^^^|four^^^|cuatro|'))
    ['one|uno', '', 'three^^', 'four^|cuatro', '']
    """
    result = []
    token = ""
    state = 0
    for c in inpt:
        if state == 0:
            if c == escape:
                state = 1
            elif c == separator:
                result.append(token)
                token = ""
            else:
                token += c
        elif state == 1:
            token += c
            state = 0
    result.append(token)
    return result
```

a)



b)

Since the state is either 0 or 1, so the path from 20 to 11 is infeasible

$TR_{NC} = \{8, 11, 12, 13, 14, 15, 16, 19, 20, 21, 23\}$

Test paths: [8, 11, 12, 13, 14, 11, 12, 20, 21, 11, 23] [8, 11, 12, 13, 15, 16, 11, 23]
[8, 11, 12, 13, 15, 19, 11, 23]

$TR_{EC} = \{[8, 11], [11, 12], [12, 13], [13, 14], [14, 11], [11, 23], [13, 15], [15, 16], [16, 11], [15, 19], [19, 11], [12, 20], [20, 21], [21, 11]\}$

Infeasible TR: [20, 11]

Test Paths: [8, 11, 12, 13, 14, 11, 12, 20, 21, 11, 23]
[8, 11, 12, 13, 15, 19, 11, 23] [8, 11, 12, 13, 15, 16, 11, 23]

$TR_{EPC} = \{[8, 11, 12], [11, 12, 13], [11, 12, 20], [12, 13, 14], [12, 13, 15], [13, 14, 11], [14, 11, 23], [13, 15, 16], [13, 15, 19], [16, 11, 23], [15, 19, 11], [19, 11, 23], [20, 21, 11], [21, 11, 23]$ [14, 11, 12], [16, 11, 12], [19, 11, 12], [8, 11, 23]$\}$

Infeasible TR: [20, 11, 12], [12, 20, 11], [20, 11, 23],

Test paths: [8, 11, 12, 13, 14, 11, 12, 20, 21, 11, 12, 13, 14, 11, 23]
[8, 11, 12, 13, 15, 19, 11, 12, 13, 15, 19, 11, 23]
[8, 11, 12, 13, 15, 16, 11, 12, 13, 15, 16, 11, 23]
[8, 11, 12, 13, 14, 11, 12, 20, 21, 11, 23]
[8 11, 23]

d) It is not possible to satisfy (1). because in order to cover all nodes all edges will be covered as well except for the feasible edge.

Q4 a: For int.py, I achieved 90% coverage. The line I cannot cover are the lines that will raise error due to the incorrect semantics of RelExp(line 75) and it will never reach. Also I cannot cover the main function and the _parse_args() because I am not calling the main function and the _parse_args() function is called in the main function(line 179-197).
For parse.py, I achieved 96% coverage. I cannot cover are line 268 which raise the error that I cannot get into. Line 481 and 482 I cannot cover the NEWLINE function as I couldn't get the correct pattern. Line 590-609 cannot cover due to I didn't call the main function.
For ast.py, I achieved 84% coverage. The line I cannot cover are related to the class AstVisitor and PrintVisitor that couldn't resolve.

b: For int.py I achieved 91% branch coverage. For ast.py, I achieved 81% branch coverage. For parser.py, I achieved 96% branch coverage. The line couldn't cover have the similar reasons as part a. Due to some statements are not covered in part a which leads to the missing branch coverage in this part. Compared to the result from part a, after covering all statements in part a, most branches are covered.

c: I achieved 88% coverage for stats_visitor.py. The line I cannot cover are the lines related to main function (line 105-116)

d: I achieved 100% coverage for undef_visitor.py.