Name: Devansh Kansara
StudentId: 21112551
Email: dpkansar@uwaterloo.ca

Q1)

a) **Test case: a = 2 and b = 2.** Here the input is invalid, therefore the program will crash before executing the line 8. Hence, this does not execute the fault.

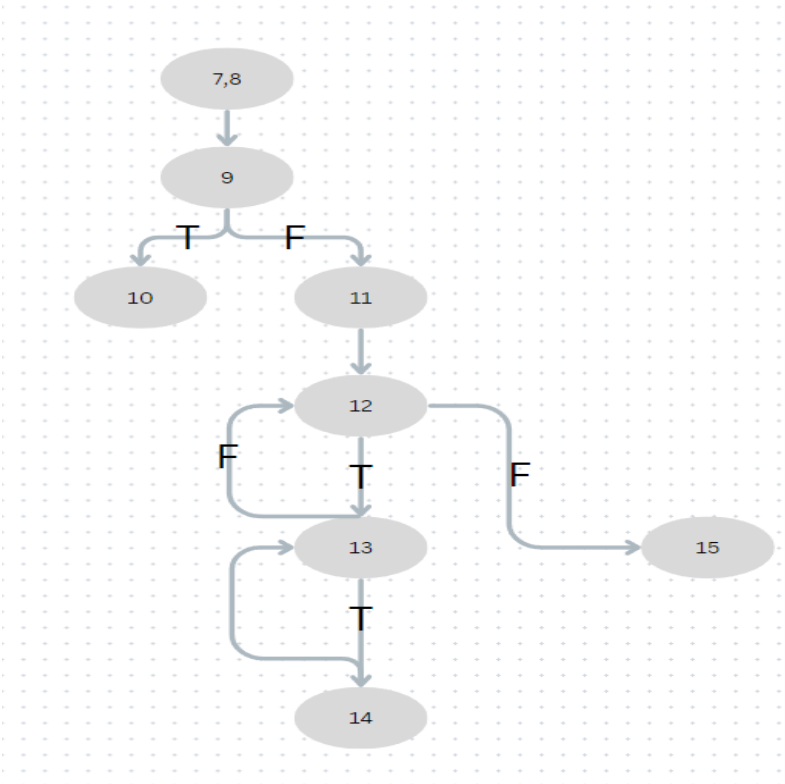
b) **Test case: a = [[1,2], [3,4]] and b = [[2,3], [5,7]].** In short, when a is a m x n matrix and b can be n x n matrix. That is when b is a square matrix of dimension n.

c) **Test case: a = [[1,2], [3,4]] and b = [[1], [2], [3]].** Here, we should get an error for incompatible dimensions but this will not result in a failure as we will get n=2, p=2, q=3, p1=1. However, in correct program, we will get q=1 and p1=3 which will also result in incompatible dimensions.

d) **State where the first error occurs:**

a = [[5,7], [8,21]]
b = [[8], [4]]
n=2
p=2
q=2
p1=1
c=undefined
i=undefined
j=undefined
PC → At line 10

e)



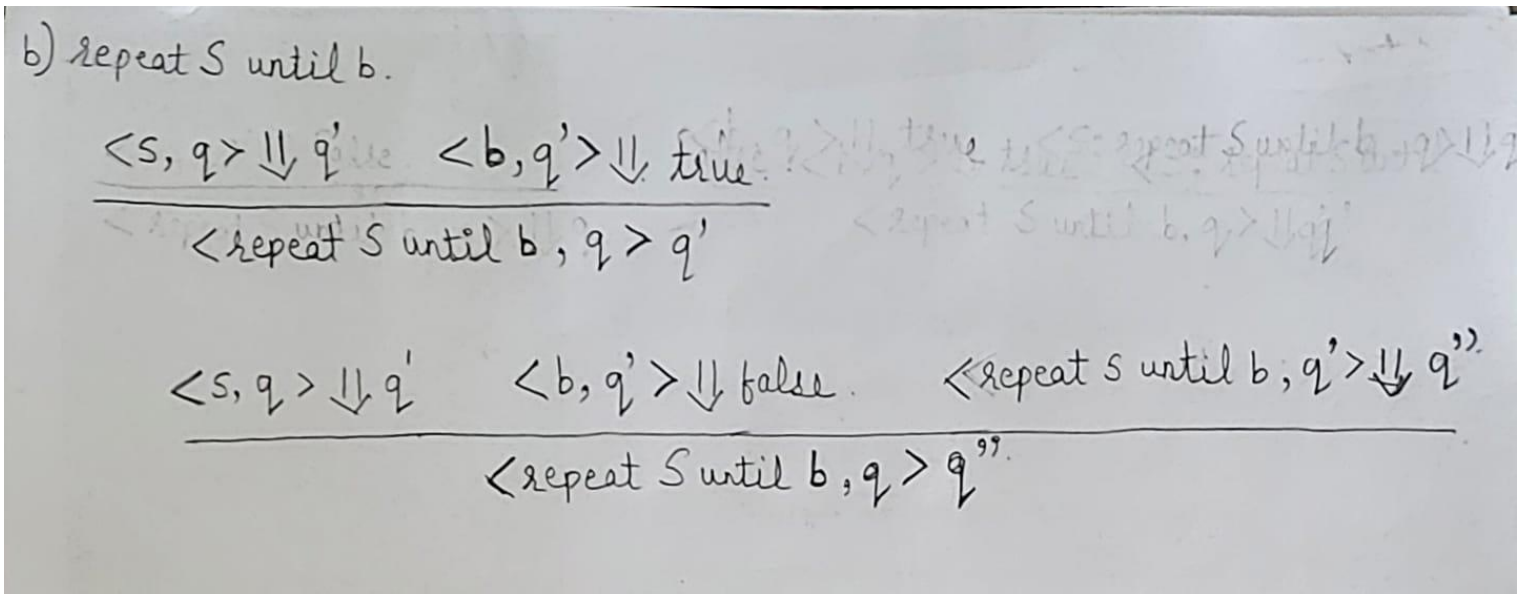
Assume that T means true and F means false.

Q2)

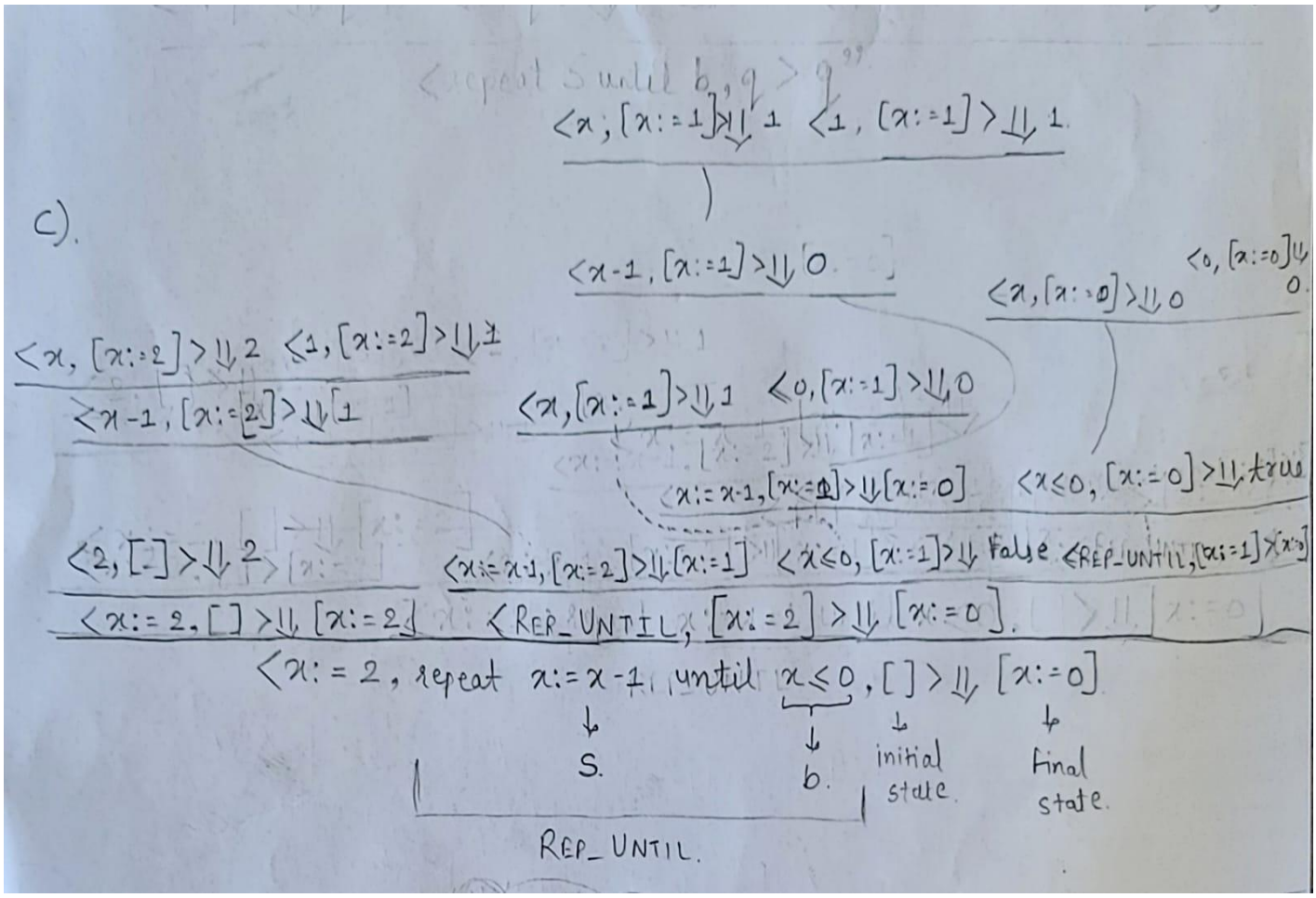
a)

```
class Ast(object):
  """Base Class of AST heirarchy"""
  pass

class Stmt(Ast):
  """A single statement"""
  pass

class AsgnStmt(Stmt):
  """An assignment statement"""
  def __init__(self,lhs,rhs):
    self.lhs=lhs
    self.rhs=rhs

class IfStmt(Stmt):
  """If-else statement"""
  def __init__(self,cond,then_stmt,else_stmt=None):
    self.cond = cond
    self.then_stmt=then_stmt
    self.else_stmt=else_stmt

class RepeatUntil(Stmt):
  """Repeat-until statement"""
  def __init__(self,cond,repeat_stmt,until_stmt=None):
    self.cond=cond
    self.repeat_stmt=repeat_stmt
    self.until_stmt=until_stmt
```

b)



c)


c).
⟨repeat S until b, q⟩
⟨x, [x:=1]⟩ ⇓ 1 ⟨1, [x:=1]⟩ ⇓ 1
⟨x-1, [x:=1]⟩ ⇓ 0
⟨x, [x:=0]⟩ ⇓ 0
⟨0, [x:=0]⟩ ⇓ 0
⟨x, [x:=2]⟩ ⇓ 2 ⟨1, [x:=2]⟩ ⇓ 1
⟨x-1, [x:=2]⟩ ⇓ 1
⟨x, [x:=1]⟩ ⇓ 1 ⟨0, [x:=1]⟩ ⇓ 0
⟨x:=x-1, [x:=1]⟩ ⇓ [x:=0]
⟨x≤0, [x:=0]⟩ ⇓ true
⟨2, [-]⟩ ⇓ 2
⟨x:=x-1, [x:=2]⟩ ⇓ [x:=1]
⟨x≤0, [x:=1]⟩ ⇓ false
⟨REP_UNTIL, [x:=1]⟩ ⇓ [x:=0]
⟨x:=2, []⟩ ⇓ [x:=2]
⟨REP_UNTIL, [x:=2]⟩ ⇓ [x:=0]
⟨x:=2, repeat x:=x-1, until x≤0, []⟩ ⇓ [x:=0]
S.
b.
initial state.
final state.
REP_UNTIL.

d)

(d) We have:

$\langle$repeat S until b, q$\rangle \Downarrow q'$ .... (i)

$\langle$S; if b the Skip else (repeat S until b), q$\rangle \Downarrow q'$ ....(ii)

For (i), we assume that we get state q'' after executing S at state q, and so there exists a derivation tree D for it.

Case I: If $\langle b, q''\rangle \Downarrow$ false,

$$\frac{D_1}{\langle \text{repeat S until b}, q\rangle \Downarrow q'}$$ where $D_1 = \langle$repeat S until b, q$\rangle \Downarrow q'$

Thus,

$$\frac{\langle S, q\rangle q'' \quad \langle b, q''\rangle \Downarrow \text{false} \quad D_1.}{\langle S; \text{if b then skip else (repeat S until b)}, q''\rangle \Downarrow q'}$$

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

Thus (i) holds.. Now for $\langle b, q''\rangle \Downarrow$ true, the we can say $q' = q''$:

$\langle$repeat S until b, q$\rangle q''$ (as $q' = q''$).

So,

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

Thus, we've completed the Ist part of the proof: We also need to prove other way around:.

if $\langle$S; if b then skip else (Repeat S until b), q$\rangle \Downarrow q'$ ... (iii)

then. $\langle$repeat S until b, q$\rangle \Downarrow q'$ ..... (iv).

Again assume that (3) holds and we get q'' after executing S at state q.

Case 1: If $\langle b, q''\rangle \Downarrow$ false, then

$$\frac{Q_1.}{\langle S; \text{if b then skip else (repeat S until b)}, q\rangle \Downarrow q'}$$

where $Q_1 = \langle$repeat S until b, q''$\rangle \Downarrow q'$

Using the above,

$$\frac{\langle S, q\rangle \Downarrow q'' \quad \langle b, q''\rangle \Downarrow \text{false} \quad Q_1}{\langle \text{repeat } S \text{ until } b, q\rangle \Downarrow q'}$$

. Thus (iii) holds.

Another Case: if $\langle b, q''\rangle \Downarrow$ true, then according to rule $\langle \text{skip}, q''\rangle \Downarrow q''$, we've $q' = q''$. $\Rightarrow$ $\langle S;$ if b then skip else (repeat S until b), $q\rangle \Downarrow q''$

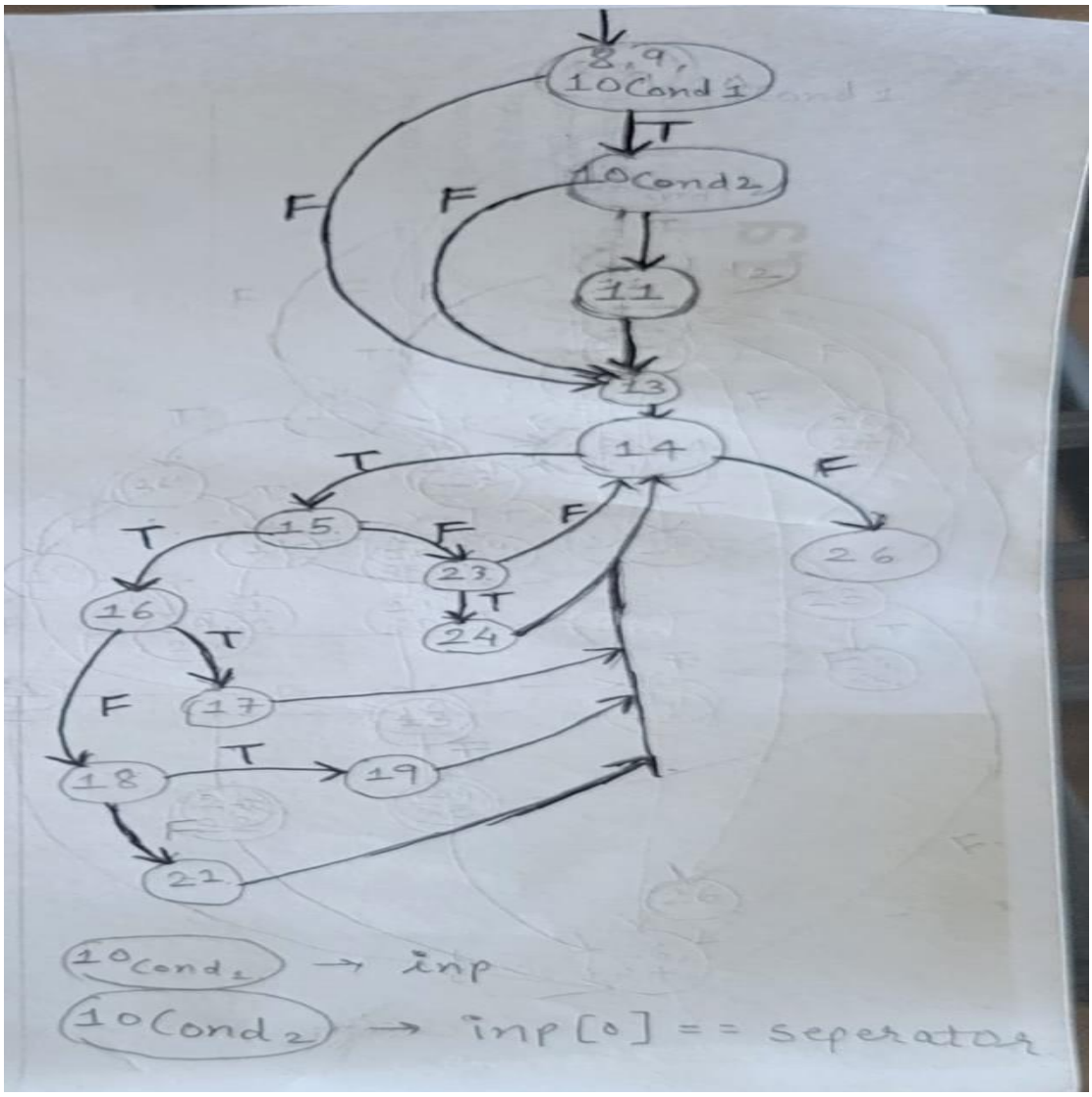
As $q' = q''$, we have:

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

Thus, the proof has been completed.

Q3)

a)



b) Node Coverage:

$\mathbf{TR_{NC}}$ = {{8,9,10_{Con1}}, 10_{Cond2}, 11, 13, 14, 15, 16, 17, 18, 19, 21, 23, 24, 26}

Edge Coverage:

$\mathbf{TR_{EC}}$ = {({8,9,10_{Con1}}, 10_{Cond2}), ({8,9,10_{Con1}}, 13), ($10_{Cond2,}$ 11), (10_{Cond2},13), (11,13), (13,14), (14,15), (14,26), (15,23), (15,16), (23,24), (23,14), (24,14), (16,17), (16,18), (17,14), (18,19), (19,14), (18,21), (21,14)}

Here, the edge (23,14) is infeasible as the variable 'state' has only two values, 0 or 1, so the program at node 23 will always be true and go to 24 and will never be false.

Edge-pair Coverage:

$\mathbf{TR_{EPC}}$ = {({8,9,10_{Cond1}},10_{Cond2},11), ({8,9,10_{Cond1}},13,14), (10_{Cond2},11,13), (10_{Cond2},13,14), (11,13,14), (13,14,26), (13,14,15), (14,15,16), (14,15,23), (15,23,24), (15,16,17), (15,16,18), (16,17,14), (16,18,19), (16,18,21), (18,19,14), (18,21,14),(23,14,15),(23,14,26),(23,24,14), (24,14,15), (24,14,26), (17,14,15), (17,14,26), (19,14,15), (19,14,26), (21,14,15), (21,14,26)}

Here, the pairs (15,23,14), (23,14,15) and (23,14,26) are infeasible nodes as we cannot reach the edge (23,14)

c)

1) **Node coverage but not edge coverage:** see coverage__tests.py
2) **Edge coverage but not edge pair coverage:** This is impossible to achieve as the edge ({8,9,10_{Cond1}},13) and ({8,9,10_{Cond1}},10_{Cond2}) cannot be executed at the same time. Similarly, (10_{Cond2},13) and (10_{Cond2},11) cannot be executed at the same time because for the first pair, either an input exists or an input does not exist. Similarly, for the send pair, either an input has a separator at the first(inp[0]) position or it does not have a separator.
3) **Edge-pair coverage but not prime-path coverage:** This is also impossible as the edge pairs ({8,9,10_{Cond1}},13,14) and ({8,9,10_{Cond1}},10_{Cond2},11) OR ({8,9,10_{Cond1}},10_{Cond2},11) and ({8,9,10_{Cond1}},10_{Cond2},13) are impossible to execute at the same time as a n input can have a separator at the first position or it can't have a separator at the first position but both of that cannot happen at the same time.

Q4)

a) **int.py:75** is not covered because expression only has '<=' | '<' | '=' | '>=' | '>' relational operators.
int.py: 179 – 197 are not covered because we did not execute the int.py as the main program.
parser.py: 590-609 are not covered because we did not execute the parser.py as the main program.
int.py:35 is not covered because the function does not return anything.
parser.py: 481 – 482 Newline character is not a valid syntax.

b) **int.py**: 72 – 75 is not covered because expression only has '<=' | '<' | '=' | '>=' | '>' relational operators. So, assert False is never executed
int.py: 90 – 94 is not covered because expression only has 'not' | 'and' | 'or' logical operators.
int.py: 111 – 114 is not covered because expression only has '+' | ' −' | ' ∗' | '/' arithmetic operators.
int.py: 196 – 197 is not covered because we are not executing int.py as the main function.
parser.**py**: 603 – 604 is not covered because we are not executing parser.py as the main function.

Conclusion about the interpreter: The interpreter is able to check all the input paths' validity. Hence, it is sound and complete.