Milestone Report

Name: Zixuan Zhao

ID: 932-282-628

My algorithm is based on the Feedback Test Generation algorithm. Here is the pseudocode:

```
receive and produce the parsing arguments
initialize the sut class
com_pool <- all actions related to the defined variable in pool
bad_pool, exc_pool, nice_pool <- []
while execution time is still in TIMEOUT, do
        randomly pickup two sequences A and B
        create a new sequence S by appending B to A
        execute sequence S
        classify S as poor test and put into bad_pool or exc_pool by checking current test
property, error, exception
        filter S if there is an existed same test sequence in nice_pool
        if pass the test, save current test sequence S into nice_pool
```

To complete this pseudocode, I split it into three processes, incremental generation, error checking and filter, and more update details.

Incremental Generation

In this process, my algorithm will randomly pick up multiple sequences and simply append the sequence into another sequence. To make sure I can build up sequences.

To randomly pick up a new test function and incrementally append to the old sequence, I will do these steps:

1. Create a list named after com_pool. This list com_pool will store all the sequences used for test generation, which means good tests. And it will be initialized by including all the actions' functions in sut.py which are related to the variable in pool defined in the TSTL file. For example, in TSTL file avlefficient.tstl, it defines

   pool: <avl> 1 REF

   pool: <val> 4 CONST

   In this case, all the actions function in sut.py related with "avl" and "val", such as act0(), act1(), will be put into com_pool. This means we will initialize the variable that we need in testing at first. Since these variables will be used to store or pass value in testing, it is better to be put into com_pool so that they can be picked up firstly.
2. In while loop:
   1) Algorithm will pick up two sequence randomly from com_pool by random generator.

2) Create a new sequence S by appending one sequence to another. To make it simple, I will generally append the second picked up one to the first one.
3) Execute sequence S
4) Add sequence S to com_pool first.

Error checking and Filter

In this process, algorithm will classify the sequence whether they are good testing or bad testing. This can be judged by these aspects:

1. Exceptions
   Exceptions normally are caused by unappropriated sequentially testing. They are not produced by tested operations themselves. In this case, it is better to classify and store these kinds of sequences into list exc_pool.
2. Errors
   Errors means the faults caused by operations themselves. These sequence producing error will be stored into bad_pool. Finally, these also will be reported.
3. Same Testing
   Some new sequences may result in the same testing with the ones in nice_pool. We will filter out those redundant sequence. We will not store those redundant sequence since it is not necessary.

More Details

In this process, I would like to modify some details.

1. Simplify execution

Since com_pool stores lots of sequences which are already tested, it is better to avoid retesting these tested ones when we execute the new sequence created by ones from com_pool. My idea is trying to store the status of test by some way and we can continue test based on these status.

2. Predict and filter unnecessary or bad sequence before execution

In this algorithm, to make it simply, I will execute the sequence and output some status or results at first, then judge the sequence. If I can reduce some redundant sequence before execution, it would save total time to run test.

Currently, I finished first process and I am still doing second process. One more thing is that, in my algorithm, I will not use all the parsing arguments. Since I do not work on more on the parameters setting part, probably “width” parameter will not be considered in my algorithm.