Milestone Report 2

Name:Zixuan Zhao
Student ID: 932-282-628

In this process, I almost modified my whole algorithm to suit TSTL. Here are the changes.

1. Reorganize Data Structure

Last time, I created three pools to store different state, one is for nice sequence, one is for error sequence, and another one is for component sequence. The pool for nice sequence is similar with the pool for component sequence. The only difference is that component sequence pool contains initialized variables. Thus, I remove the component sequence pool. And the pool for nice sequence will contains the initialized variables at first.

What's more, each pool stores the sut.test() information in previous version for each test. This affects how to generate the sequence. In this time, to improve my algorithm, I store the sut.state() information into pools. I will explain why I store it in next part.

2. Generate Sequences and Run Sequence

Last time, my algorithm has two ways to increase sequences, one is to randomly pick up two sequences in component pool and concatenate them together, then run this sequences by sut.replay() function. Another one is to randomly pick up one sequence and run it by sut.replay() function. Also, randomly pick up one action by sut.randomEnabled() and run it by sut.safely(). This method a kind of exactly follows the feedback test generator algorithm. However, it is not efficient in TSTL.

The modified version is as following:

1) Randomly pick up one sequence in nice sequence pool.
2) Call sut.backtrack() function to set the test by the picked sequence.
3) Randomly pick up several actions by sut.randomEnableds().
4) Run these actions one by one and check the error at the same time.
5) When there is no error after running all actions or the depth reached to the specific number, append this test's sut.state() into nice sequence pool. Otherwise, the test's sut.state() will be input into error sequence pool.

Since I store the sut.state() of a test into pool, I can easily and fast to replay the test by step 2. This really saves the time without running all actions again. As I said before, I remove the component sequence pool, since the actions returned by sut.enabled() is like the function of component sequence pool. Also, it is easily to control the depth for each sequence.

Currently, there is still one problem that I still cannot break through, which is the filter function. Filter function is used to reduce the duplicated test. My original idea is to reduce the sequence while generating sequence. In that case, we can easily to compare new sequence with old sequence and avoid duplications. However, it is hard to achieve it right now. It takes really long time to reduce sequence.

Good thing is the new tester can find bugs in shorter time than before. But it is not so stable and too many duplicated failures. What I want to share is that when I wrote my first version algorithm, I only considered how to exactly achieve the algorithm. That made my test generator work worse. In this time, I change my mind and try to suit my code with TSTL. I believe my new tester improves a lot than before. This is a really good experience.