CS 569
Milestone 2
05/18/2016
Hao Wang (wanghao@oregonstate.edu)

Implementation of Feedback-directed Random Test in TSTL

1. Background

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied to virtually every level of software testing, unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well[1]. For Black-box testing, testers do not need to know the process of implementation, or focus on source code. The high generality of method and low requirements of testers lead to a high concurrency and collaboration of the whole project process, which is the core idea of software engineering.

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail[2]. As an effective method of black-box testing, it is a complement of white-box testing to garantee the completeness of the test process. However, random testing is not a good enough method sometimes since its inherent shortage. Currently, there is a number of modifications that work with the shortage of random testing.

2. Algorithm and Implementation

I get the idea of algorithm from the paper "Feedback-directed random test generation"[3], and the code skeleton from the given randomtester.py and newCover.py[4]. My algorithm is a modification of random testing. Instead of just creating inputs randomly, this algorithm builds 3 method sequences to store the methods. These are error sequence, non-error sequence, and new sequence. Error sequence stores the cases that have been tested right. Non-error sequence stores the cases that have been tested wrong. New sequence stores the cases that are new random generated. Afterwards, do the following steps:

1. Randomly generate a test case and put it in the new sequence;
2. Check whether this case is in error or non-error sequence, if yes, go to step 1, otherwise go to step 3;
3. Test the case and put it in the error or non-error sequence based on its result;
4. Go to step 1 unless the time is out.

It is good to show a flow chart for a clear understanding as follows,

Comparing with the normal random testing, there are at least 2 advantage of this modification. First, it saves time since it stores the result of tested cases, so it will not

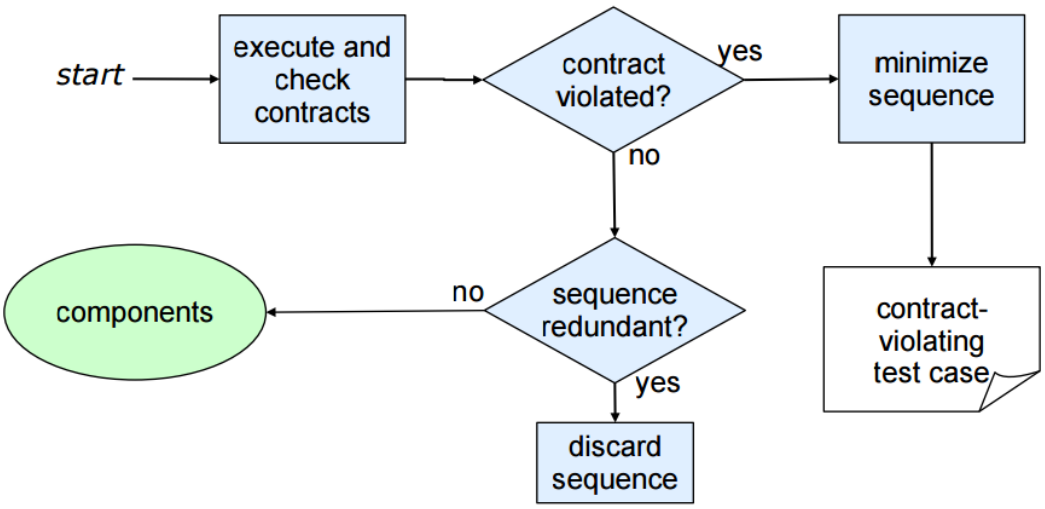
redundantly repeat former tested cases. Second, it will generate a higher coverage of all possible cases in a limited time, because random test would most probably repeat same cases multiple times.

Also, I implemented the parameter part by using the sys.argv.

3. Modification and Future Work

In milestone 1, I had some error while calling the non-error sequence. Thus I just left it a comment, and check the error sequence only. In the last days, I kept working on it and finally solved it. Now the feedback-directed algorithm is nearly 80% completed.

In the original paper [3], the whole algorithm can be simply described as the following flow chart:



I have completed almost all steps but discard the redundant sequence. I think it is a part that could not be implemented easily in TSTL. I will keep working on it in the following days.

Reference

[1]. S. Debnath. *"Mastering PowerCLI"*. Packt Publishing Ltd. 2005. pp.35.

[2]. R. Hamlet (1994). *"Random Testing"*. In J. J. Marciniak. *"Encyclopedia of Software Engineering"*. John Wiley and Sons. 2013.

[3]. C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball. *"Feedback-directed random test generation"*. In *"Proceedings of the 29th International Conference on Software Engineering"*. 2007. pp.75-84.

[4]. "https://github.com/agroce/cs569sp16"