Avoid known bugs generated by TSTL Random Tester

CS 569 – Project Proposal

Swathi Sri Vishnu Priya Rayala (932-696-872)

Background

Random Testing (RT) 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. Thus, RT is used to assess software reliability as well as to detect software failures by simply selecting test cases in a random manner from the whole input domain. However, there is a lot of scope in improving the Random Testing to detect the software failures depending on failure-causing inputs.

Proposal

The existing Random Tester of TSTL, tries to identify all the possible bugs for a given timeout period. But, all the identified bugs are not necessarily the unknown bugs. Some bugs can be repetitive. So, my idea is to modify the existing random tester of TSTL in such a way that bugs identified should be distinct. Also, I'm curious to see what the branch coverage and code coverage would be.

There are few algorithms to implement this as mentioned in [1] and [2]. The first approach is to generalize each bug as it is found, to a 'bug pattern', and then adapt test case generation so that test cases matching an existing bug pattern are never generated again randomly [2]. The second approach is to randomly select a point by generating two random numbers. The point is tested to see whether it contains error or not. If it contains an error, all the neighbors are computed and the process is repeated for each neighbors. Otherwise another point is selected randomly [1]. The third approach is to randomly select a point by generating random numbers. The point is checked to see whether it contains error or not. If the selected point contains an error all its end points are computed and checked to see whether the point contains an error. Otherwise another point is selected randomly. All the tested and defect containing points are put into the arrays DT[] and DF[] respectively and by performing a boundary test, the pattern type is identified [1].

Plan

I would first review the algorithms mentioned in [1] and [2]. The existing random tester uses breadth first search and depth first search algorithms which is said to be more expensive. So, I would choose an algorithm that would be less expensive and more efficient to be implemented with TSTL. Later, I'll concentrate on implementing one of those algorithms that supports with TSTL API. I would prefer taking the existing AVLTree example and find bugs in it. If that was successful, I would try to improve the test case generation besides maximizing the code coverage.

References

[1] Algorithms to identify failure pattern, Bhuwan Krishna Som, Poudel

Url: http://www.diva-portal.org/smash/get/diva2:655645/FULLTEXT01.pdf

[2] Find more bugs with QuickCheck, John Hughes, Ulf Norell, Nicholas Smallbone

Url: http://publications.lib.chalmers.se/records/fulltext/232554/local_232554.pdf

[3] TSTL: A Language and Tool for Testing (Demo), Alex Groce, Jervis Pinto, Pooria Azimi, Pranjal Mittal

Url: http://www.cs.cmu.edu/~agroce/issta15.pdf