CS 569

Instructor: Professor Alex Groce

Student Name: Yipeng Wang

ID: 931903609

Date: Jun 5, 2016

Forth Part

Final Submission

Feedback-directed Random Test Generation in TSTL

Introduction

In my project proposal, I choose the paper "Feedback-directed Random Test Generation", which was written by Pacheco, Carlos; Shuvendu K. Lahiri; Michael D. Ernst and Thomas Ball. In that paper, author pointed out a kind of method which could test program efficiently, which was called feedback-directed random test. This method is based on the random test generation, which will analyze the input data, if it has certain feature, it will do certain implementation. By using feedback-directed random test method, it could reduce the unnecessary data tested in many times efficiently. Therefore, it could improve the efficiency on program testing.

Algorithm from Feedback-directed Random Test Generation

For the feedback-directed random test generation, it will generate the incremental data sequences. Then it will take four types of input: classes, contracts, filters, and time limit. In time condition part, it will randomly choose the sequences to append with the data sequences which we test previous. After appending, classify the result, for each sequence, it has a Boolean flag to determine the set of contracts and filters: if it is

redundant or illegal, we will put it into error sequence and ignore it; if it has bugs, we will record the bugs in logs; if it is useful, we can use it again in future test appending.

What I did in tester1.py

In second part of the project, I took the newCover.py from the Professor Alex's website as the reference. In this part, there are seven important part need to pay more attention. The first is the timeout, we use it to limit the time consumption in this testing generation implementation, which means if the program runs for enough long period to reach the limitation, the testing generation will automatically halt. The second part is seed, for this parameter, we can use it to set the random numbers in random.random object in python. The third part is depth, for this parameter, we can use it to set the maximum length of the test generation implementation. The fourth part is width, for this parameter, we can use it to limit the maximum width of the testing generation implementation. The fifth part is faults, this part is very important. For this parameter, it will only have 0 or 1 which depends on the checking faulting mechanics we defined in sut file, if it returns true, it needs to store the faults result of this testing generation; if it returns false, it will not record the result in sut. The six part is coverage, for this parameter, it will also only contain 0 or 1, which depends on the final coverage report producing, it will use TSTL's internalReport() function, if it returns true, it will print the report, else, it will not. The last parameter is running, it is familiar with the previous two parameter, it only has 0 or 1, if it returns true, it will print; else, it will not. For the algorithm, I divided the test into two phases, each part will contain the half of the whole running time. For the benchmark part, I sort the cover times of the action, then get the median number of the list. If the cover time of the action is below than median, it will store the result; else, it will skip that. In the second part, I run the states randomly to get the new list which could improve the coverage ratio.
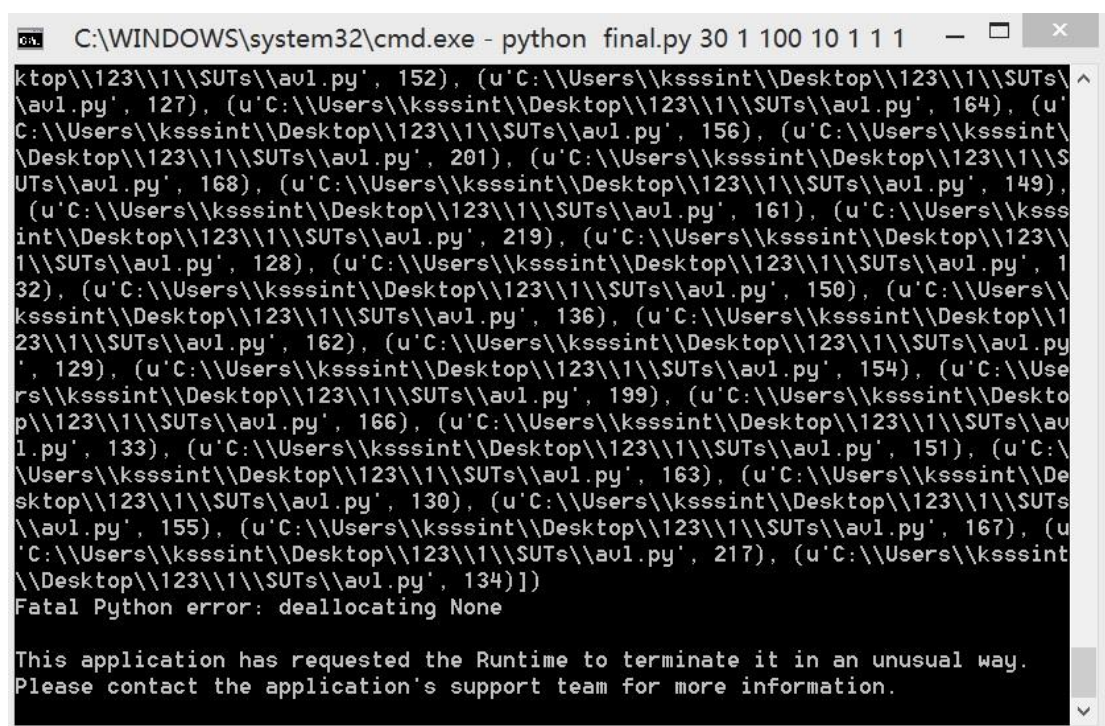
Improvement in Tester2.py

In milestone 2, I added a new function for my tester file. Since, I use the newcover.py

as the reference, which has quite a limit space to improve the performance, because in newcover.py, it runs the program twice then compare the difference, so I choose to add a new function in milestone 2. In this part, I add a function which could output the coverage report for users. We can not only check the internal report to see whether the bugs exist, but also check the coverage report to check the coverage rate like I did in last term.

Problem I meet

For doing the final tester file, I met a very annoying problem which cost me lots of time to do it. At last, I still cannot fix it. The problem is after I finished the final tester file, I want to implement the tester file to check the result whether is in expect. However, the python got crashed all the time, like this:

At first, I thought it was caused by my python, however, after I reinstall the python, this problem still exist. I also borrowed a mac computer from my friend, the problem also happened. Some of my classmates also met this problem, he also cannot fix this problem. But he tried to use ubuntu, after that, there is no such problem. I try to download the virtual machine and no such problem exist anymore.

```
X _ ☐   wang@ubuntu: ~/Desktop/123/1/SUTs
TSTL STATEMENT COUNT: 140
18 TESTS
1 10
3 13
3 15
2 17
3 58
5 63
5 65
7 70
5 74
13 104
4 112
6 113
7 115
6 123
3 136
12 137
7 138
7 139
0 FAILED
TOTAL ACTIONS 16815
TOTAL RUNTIME 15.1144928932
wang@ubuntu:~/Desktop/123/1/SUTs$ █
```

But I still curious why the problem will happen.

In Finaltester.py

In final tester file, I focus on solving all problem which I meet in previous version. Therefore, in final tester file, it could solve all parameter which I set. In addition, I add a new parameter which named factor, by using the factor, I could control the size of active pool. If the factor increase, the active pool will decrease, the branches will also decrease. If the factor decrease, the active pool will increase and the branches will increase. In addition, I add the print out for the branch and statement, so I could not only check the result from internal report, but also get from the final report.

In Mytester.py

In this file, it include all functions which I add in final tester file. In addition, I add a new function which called failure collection. By using this function, I could collect all the failure into a log file, so user could get the failure report much easier.

Future Work

From this class, I learned some useful method to debug the program. In my tester file, I try to use the feedback directed random test by using the TSTL tool. After the period of this term, I changed my tester file a lot, from the first version, which still have some bugs and sometimes might cause timeout, to the last version, the final tester file, I could debug the program by running every parameter without any timeouts or crashed. In addition, I added some interesting small functions to improve the tester file.

From the tester file, I could say that the algorithm have already implemented some results on improving the random test algorithm on coverage ratio and fail detected. However, there are still some space which could improve in the future. One of the point is the

Reference:

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