Name: Huipu Xu

Email: xuhui@oregonstate.edu

Class: ST/STATIC AN/MOD CHECK SFT ENG

Professor: Groce, Alex

Final Report

Introduction

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests. Random testing is useful even if it doesn't find as many defects per time interval, since it can be performed without manual intervention. An hour of computer time can be much less expensive than an hour of human time. Even if the random test can help user a lot in detecting errors and failures, but no one test is perfect for debugging, random test cannot cover all statements and branches in SUT. This mean, sometime you cannot detect all fails in all branches in your programs. The problem and drawback of the test generation is not only in algorithm, but also in machine technique issue. Recently, Adaptive Random Testing (ART) was proposed as an effective alternative to random testing. A growing body of research has examined the concept of Adaptive Random Testing (ART), which is an attempt to improve the failure-detection effectiveness of random testing. ART is based on various empirical observations showing that many program faults result in failures in contiguous areas of the input domain, known as failure patterns. Hence, right now, we can write a good test algorithm to improve the efficient of test generation. At present, I only need to fix some algorithm issue and improve its efficient.

Algorithm

In this report, you should have a basic implementation of a novel test generation algorithm using the TSTL API. And our code must include: timeout, seed, depth, width, faults, coverage and running. Here is some code provided by professor below.

My idea is just to improve the test branch's coverage. Only we do the test detection as many branches as possible, the algorithm will be more and more efficient. So I will handle the test method for SUT in black box. First of all, in test generation, collecting all the tests state in a list, saving the state of current test and do 40% backtrack on the save test. Then I collect the branch coverage and find the least coverage in current branches or not.

Next step the test will generate random tests with random actions by running the pure random tests. Then, I will check any failure while generated by test cases. When find failures, which means the actions in TSTL have error or are not ok. As we know, TSTL is a Template Scripting Language, which will check on the actions for SUT and Properties. Therefore, when you running the tester, and finding any bugs, the tester will save it into a file called failure “j”.test in current directory. Then, collect all of name of actions and sort them by their action count. In addition, once user set the specific running time, and when the running time was reached, the tester will stop and show the result of test to users.

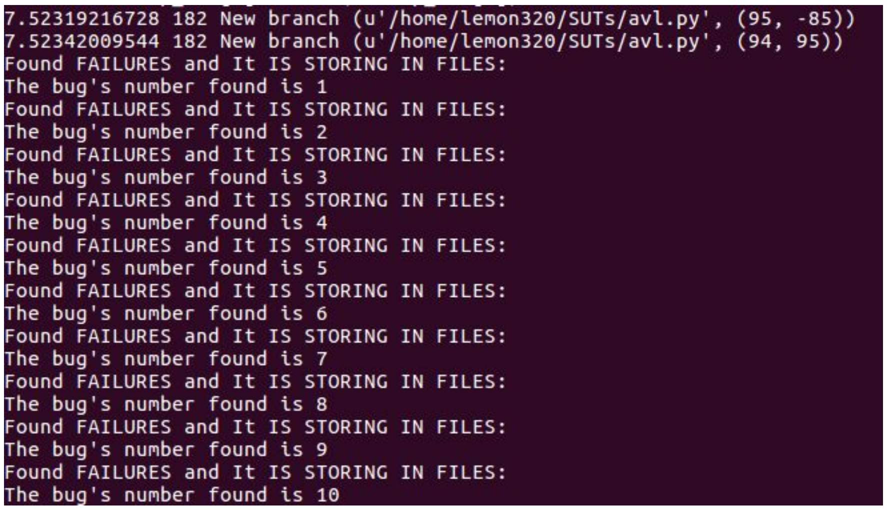
Finally, put the result in another pool and do statistics for them. In this code, there is a def loopBug(), which is for generating random tests and find bugs if faults = 1.

Part of functional code shows here:

```
def loopBug():
   global Stest,rgen,sut,depth,LCov,actCount,running,faults,j,bugs
   if Stest != None:
      if rgen.random() > 0.4:
         sut.backtrack(Stest)
   test = False
   for s in xrange(0,depth):
         act = sut.randomEnabled(rgen)
         ok = sut.safely(act)

         if len(sut.newBranches()) > 0:
            Stest = sut.state()
            test = True

         if (not test):
                                          if (LCov!= None):
                                                   if (LCov in sut.currBranches()):
                                                            Stest = sut.state()
                                                            test = True

         actCount += 1
         if running == 1:
            if sut.newBranches() != set([]):
               print "ACTION:",act[0]
```

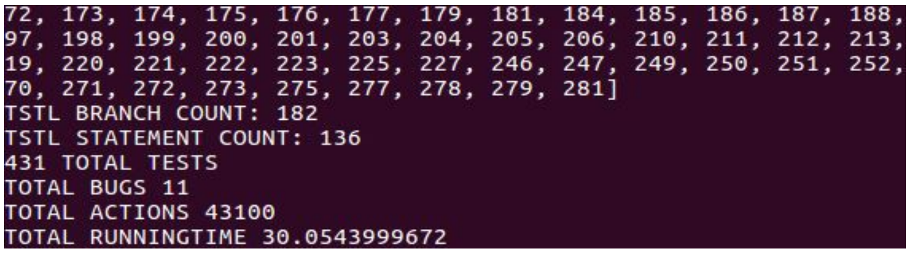
```
        for b in sut.newBranches():
            print time.time() - start,len(sut.allBranches()),"New branch",b


if faults == 1:
    if not ok:
        print "Found FAILURES and It IS STORING IN FILES:"
        j += 1
        bugs += 1
        fault = sut.failure()
        failurename = 'failure' + str(bugs) + '.test'
        sut.saveTest(sut.test(), failurename)
        print "The bug's number found is" ,j
        sut.restart()
```

Running result and evaluate the algorithm:



```
7.52319216728 182 New branch (u'/home/lemon320/SUTs/avl.py', (95, -85))
7.52342009544 182 New branch (u'/home/lemon320/SUTs/avl.py', (94, 95))
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 1
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 2
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 3
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 4
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 5
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 6
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 7
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 8
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 9
Found FAILURES and It IS STORING IN FILES:
The bug's number found is 10
```

Picture 1



```
72, 173, 174, 175, 176, 177, 179, 181, 184, 185, 186, 187, 188,
97, 198, 199, 200, 201, 203, 204, 205, 206, 210, 211, 212, 213,
19, 220, 221, 222, 223, 225, 227, 246, 247, 249, 250, 251, 252,
70, 271, 272, 273, 275, 277, 278, 279, 281]
TSTL BRANCH COUNT: 182
TSTL STATEMENT COUNT: 136
431 TOTAL TESTS
TOTAL BUGS 11
TOTAL ACTIONS 43100
TOTAL RUNNINGTIME 30.0543999672
```

Picture 2

In order to test my algorithm, I am going to run command: Python finaltester.py 30 1 100 1 0 1 1

The picture 1 shows: the bugs is founding…. And waiting for a short time, we can get the result above (Picture 2), which shows that after 30s running time, we do 43100 actions, get 431 tests, 182 branch and 136 statements, and finally find 11 bugs.

TEST Generation:

When user want to run the test, he or she needs to run in command line that allow us to input specific different parameters --- timeout, seed, depth, width, faults, coverage and running. Here is an example:

```
Python finaltester.py 30 1 100 1 0 1 1
```

30 is for the timeout: test generation will stop when the runtime reaches that number;

1 is for seeding which generates a random number of seeds;

100 is for the depth;

1 is for width;

0 is for faults: when it is 1 that mean the test generation will start to search for bugs or fails; when it is 0 that means test generation stop to detect bugs or fails.

1 is for printing TSTL internal coverage report to know the statement and branches were covered during the test to guide it. This will work when we specify 1 and 0 for do not print it.

1 is for running which will print the elapsed time, total branch count and new branch. It works when we put 1, 0 will not do anything.

Reference:

[1] Alex Groce coverTester.py. Retrived May 5, 2016, from

https://github.com/agroce/cs569sp16/blob/master/SUTs/coverTester.py

[2] Z. Zhou, “Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing,” in 34th Annual IEEE Computer Software and Applications Conference Workshops, 2010.

[3] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, “Adaptive random test case prioritization,” in Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society Press, November 2009, pp. 233–244.

[4] A. Groce; J. Holmes; J. Pinto; J. O’Brien; K. Kellar; P. Mottal, “TSTL: The Template Scripting Language,” in International Journal on Software Tools for Technology Transfer.

[5] T. Yueh Chena, K. Fei-Ching, “Adaptive Random Testing: the ART of Test Case Diversity,” in Journal of Systems and Software.