Course: CS 569
Instructor: Alex Groce
Name: Ruizhi Guo
Date: 06.06.2016
Email: guoru@oregonstate.edu

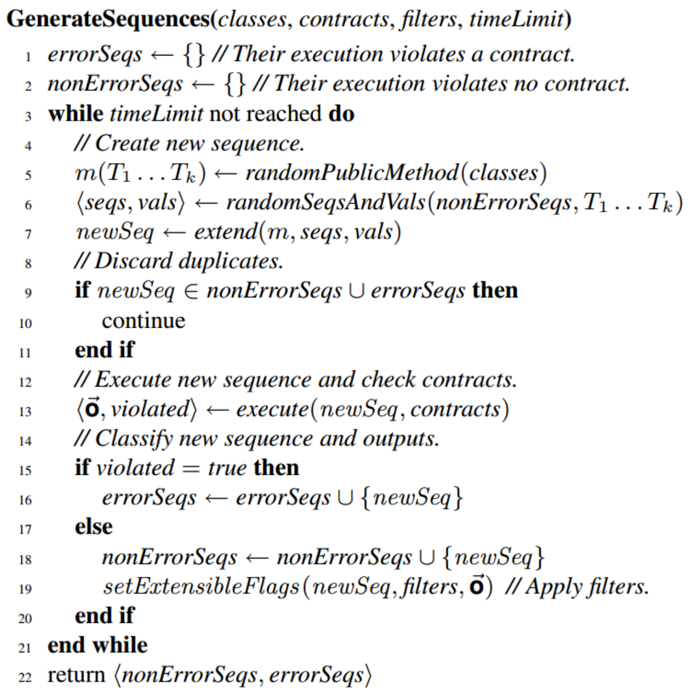
Random Test Generation in TSTL

1. Instruction

Nowadays, random test is a useful way in debugging software. But it still has some disadvantages, so some software experts spend a lot of their time on find some way to improve the random testing. As we know that random testing is very convenient to use and it could save a lot of time to people. But in some special cases, the random testing could not provide an efficient solution. So in my project, I could read some related paper in order to find a way to improve the random testing. In these papers, "Feedback-directed Random Test Generation" which write by Pacheco, Carlos; Michael D. Ernst; Shuvendu K. Lahiri; and Tomsa Ball provides some ingenious idea for me to do this project. In the authors' opinion, their technique would improve the efficacy of random testing. The authors improving the random test generation by look after the feedback obtained from the input test cases that the generator created before. In this situation, the generator could avoid some unuseful cases, then we make the random testing more efficacy.

2. Algorithm

This picture is from the article "Feedback-directed Random Test Generation", it clearly shows that how the algorithm worked. According to the paper, the authors presents a technique that improves random test generation by incorporating feedback obtained from executing test inputs as they created. The authors provide that their technique builds input incrementally by randomly selecting a method call to apply and finding arguments from among previously-constructed inputs. As soon as an input is built, it is executed and checked against a set of contracts



```
GenerateSequences(classes, contracts, filters, timeLimit)
errorSeqs ← {} // Their execution violates a contract.
nonErrorSeqs ← {} // Their execution violates no contract.
while timeLimit not reached do
    // Create new sequence.
    $m(T_1 \dots T_k) \leftarrow randomPublicMethod(classes)$
    $\langle seqs, vals \rangle \leftarrow randomSeqsAndVals(nonErrorSeqs, T_1 \dots T_k)$
    $newSeq \leftarrow extend(m, seqs, vals)$
    // Discard duplicates.
    if $newSeq \in nonErrorSeqs \cup errorSeqs$ then
        continue
    end if
    // Execute new sequence and check contracts.
    $\langle \vec{\mathbf{o}}, violated \rangle \leftarrow execute(newSeq, contracts)$
    // Classify new sequence and outputs.
    if $violated = true$ then
        $errorSeqs \leftarrow errorSeqs \cup \{newSeq\}$
    else
        $nonErrorSeqs \leftarrow nonErrorSeqs \cup \{newSeq\}$
        $setExtensibleFlags(newSeq, filters, \vec{\mathbf{o}})$ // Apply filters.
    end if
end while
return $\langle nonErrorSeqs, errorSeqs \rangle$
```

and filters. The author also believed that the result of the execution determines whether the input is redundant, illegal, contract-violating, or useful for generating more inputs. So this method generates more useful input for random test and it reduce the time of work. In the algorithm provided by the authors, there are three different sequences, they are non-error sequence, error sequence and the new sequence. This algorithm would improve the random testing by the following steps, the first step is the generator create a test case, then checking whether this case has been test, and check the feed back, if the feed back has error then put it in the error sequence other put it in the non error sequence. So this step improves the algorithm.

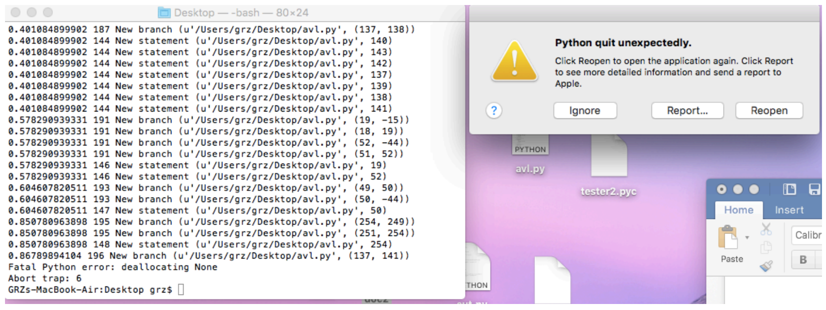
3. What I did in the tester1.py

In this part, I use the file written by Professor as an example and do my work to meet the requirement of this assignment. As the requirement by professor Agroce I make my file support the required command line. The first is "timeout", it provides the running time of the test in seconds. Second is "Seed", it provides the seed for Python Random. random object used for random number generation in my code. Third is "Depth", it provides the maximum length of a test generated by my algorithm. Forth is "Width", it provides the maximum memory/Brs queue/other parameter that is basically a search width. The algorithm is my tester1 is coming from the "Feedback-directed Random Test Generation" and it check whether the input is already test. So it would avoid the same input check again.

From the last milestone, I try to improve my project in several aspects. The first one is to find the same number of bug in shorter time. Second, I try to change the structure to improve the performance of the program. I set the time is 600 seconds, then the result would came out that it find one error at 4 second. This is different from the last version of the tester. The last version would find the error at 71 second. In this time, I set several functions and directed use them in the main. This step improves the performance, and the reason maybe because the parallel computation in the computer. The branch count improves from 180 to 187. I already finish the 90% of the Feedback-directed Random Test Generation. In the final work, I also add some functions. Such like if user enter a list of wrong parameters, the program would provides a feedback to remind that reenter the parameters.

4. Current problem I meet.

An interesting problem happen on my laptop. After I update the TSTL then I can not run the tester. It would show the problem on the picture. At first, I thought is my code has some problem. I spend a long time to fix my code. At last, I find that my code is ok, I can run it on my desktop. Then I think it is maybe something wrong



```
0.401084899902 187 New branch (u'/Users/grz/Desktop/avl.py', (137, 138))
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 140)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 143)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 142)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 137)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 139)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 138)
0.401084899902 144 New statement (u'/Users/grz/Desktop/avl.py', 141)
0.578290939331 191 New branch (u'/Users/grz/Desktop/avl.py', (19, -15))
0.578290939331 191 New branch (u'/Users/grz/Desktop/avl.py', (18, 19))
0.578290939331 191 New branch (u'/Users/grz/Desktop/avl.py', (52, -44))
0.578290939331 191 New branch (u'/Users/grz/Desktop/avl.py', (51, 52))
0.578290939331 146 New statement (u'/Users/grz/Desktop/avl.py', 19)
0.578290939331 146 New statement (u'/Users/grz/Desktop/avl.py', 52)
0.604607820511 193 New branch (u'/Users/grz/Desktop/avl.py', (49, 50))
0.604607820511 193 New branch (u'/Users/grz/Desktop/avl.py', (50, -44))
0.604607820511 147 New statement (u'/Users/grz/Desktop/avl.py', 50)
0.850780963898 195 New branch (u'/Users/grz/Desktop/avl.py', (254, 249))
0.850780963898 195 New branch (u'/Users/grz/Desktop/avl.py', (251, 254))
0.850780963898 148 New statement (u'/Users/grz/Desktop/avl.py', 254)
0.86789894104 196 New branch (u'/Users/grz/Desktop/avl.py', (137, 141))
Fatal Python error: deallocating None
Abort trap: 6
GRZs-MacBook-Air:Desktop grz$
```

Python quit unexpectedly.

Click Reopen to open the application again. Click Report to see more detailed information and send a report to Apple.

with my computer. I talk about this problem with my classmates. I find that some other classmates also has the same problem. And I find that if I set the runtime is 1s, then I can run the tester.py with no problem. If I set the running time greater than 5. Then it would happen the error show in the picture.

5. Future work

I try to set the test immediately stop when the tester found a bug. But at this time, this step is not success. I would try to implement it in the future. I would try to add some function to provides a more friendly interface to help user.

6. Conclusion

In my opinion, TSTL is very useful tool after I learn it this term. This term I learn to use Feedback directed Random Test Generation to improve the normal random test generation. Although the target is to avoid the unnecessary work. In fact, this way does not complete solve the problem. Though this class, I learn better about TSTL and learn how to write the random tester. It is worth to learn this thing and implement them. I believed this would provide some help to me in the future.

Reference

[1] J. W. Duran and S. C. Ntafos. An evaluation of random test. IEEE TSE, 10(4): 438-444, July. 1984.

[2] K. Yatoh, K. Sakamoto, F. Ishikawa, S, Honiden. Feedback-Controlled Random Test Genreation. In ISSTA 2015

[3] Pacheco, Carlos, et al. “Feedback-directed random test feneration.” Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007.

[4] R. Ferguson and B. Korel. The chaining approach for software test data generation. ACM TOSEM, 5(1): 63-86, Jan. 1996.