Course: CS 569
Instructor: Alex Groce
Name: Ruizhi Guo
Date: 05.01.2016
Email: guoru@oregonstate.edu

Part 2 Project Milestone 1 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

GenerateSequences(*classes*, *contracts*, *filters*, *timeLimit*)

```
1  errorSeqs ← {} // Their execution violates a contract.
2  nonErrorSeqs ← {} // Their execution violates no contract.
3  while timeLimit not reached do
4     // Create new sequence.
5     m(T1 ... Tk) ← randomPublicMethod(classes)
6     ⟨seqs, vals⟩ ← randomSeqsAndVals(nonErrorSeqs, T1 ... Tk)
7     newSeq ← extend(m, seqs, vals)
8     // Discard duplicates.
9     if newSeq ∈ nonErrorSeqs ∪ errorSeqs then
10       continue
11    end if
12    // Execute new sequence and check contracts.
13    ⟨o⃗, violated⟩ ← execute(newSeq, contracts)
14    // Classify new sequence and outputs.
15    if violated = true then
16       errorSeqs ← errorSeqs ∪ {newSeq}
17    else
18       nonErrorSeqs ← nonErrorSeqs ∪ {newSeq}
19       setExtensibleFlags(newSeq, filters, o⃗) // Apply filters.
20    end if
21 end while
22 return ⟨nonErrorSeqs, errorSeqs⟩
```

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 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.

Part 3 Project Milestone 2 Random Test Generation in TSTL

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 improve from 180 to 187. I already finish the 85% of the Feedback-directed Random Test Generation.

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 final version.

4. 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.