CS569
Instructor: Professor Alex Groce
Student: Kazuki Kaneoka
ID: 932-277-488
Email: kaneokak@oregonstate.edu

Part 2
Competitive Milestone 1
Feedback-directed Random Test Generation* in *TSTL

Introduction
In this document, I would like to explain the important concepts of *Feedback-directed Random Test Generation* and how I implemented *tester1.py*.

The Important Concepts of *Feedback-directed Random Test Generation*
Feedback-directed Random Test Generation (*FRTG*) is a type of Random Test Generations for automated software testing. It contains two unique concepts using runtime data: create sequence incrementally and checking contracts and filters.

- Create sequence incrementally:
  First of all, sequence in *FRTG* means a sequence of calling methods such that each method in sequence contains input arguments. *FRTG* is an algorithm for generating input test data automatically. So, we can say that sequence in *FRTG* is a generated input test data. *FRTG* creates sequence incrementally means:
  1. Randomly pick up one method or multiple methods with corresponding input arguments to generate sequence.
  2. Append this sequence to the previous sequence to create new sequence.
  3. Check the new sequence is whether we have already created before or not yet.

  Random Test Generation is well-used technique for automated software testing. However, since it is random, it often generates redundant. *FRTG* can avoid this issue by creating sequence incrementally.

- Checking contracts and filters
  Contract is the property that method or object should always hold. For example, Object *o* should always return *true* for *o equals o*.

  Filter is the evaluation for the result of executing sequence to determine whether we should expand the sequence or not. For instance, if the result of executing the current sequence is same with the result of executing the previous sequence, we do not need to expand the current sequence.

  FRTG uses contracts to find errors in the system under test and uses filter to avoid meaningless test.

How I Implemented ***tester1.py***

Basically, I implemented *tester1.py* based on *GenerateSequences* in Figure 3 in the paper, *Feedback-directed random test generation* [1].

```
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      $m(T_1 \ldots T_k) \leftarrow randomPublicMethod(classes)$
6      $\langle seqs, vals \rangle \leftarrow randomSeqsAndVals(nonErrorSeqs, T_1 \ldots T_k)$
7      $newSeq \leftarrow extend(m, seqs, vals)$
8      // Discard duplicates.
9      if $newSeq \in nonErrorSeqs \cup errorSeqs$ then
10         continue
11     end if
12     // Execute new sequence and check contracts.
13     $\langle \vec{\mathbf{o}}, violated \rangle \leftarrow execute(newSeq, contracts)$
14     // Classify new sequence and outputs.
15     if $violated = true$ then
16         $errorSeqs \leftarrow errorSeqs \cup \{newSeq\}$
17     else
18         $nonErrorSeqs \leftarrow nonErrorSeqs \cup \{newSeq\}$
19         $setExtensibleFlags(newSeq, filters, \vec{\mathbf{o}})$ // Apply filters.
20     end if
21 end while
22 return $\langle nonErrorSeqs, errorSeqs \rangle$
```

Figure 3. Feedback-directed generation algorithm for sequences.

GenerateSequences takes 4 arguments: *classes*, *contracts*, *filters*, and *timeLimit*. Instead of them, *tester1.py* takes 7 arguments: *timeout*, *seed*, *depth*, *width*, *faults*, *coverage*, and *running*.

- *timeout* is used as *timeLimit* at Line 3 in Figure 3.
- *seed* is used as the number of methods that we choose at Line 5 in Figure 3 when we choose multiple actions from *sut.py* instead of a single action.
- *depth* is used as the maximum length of sequence.
- *width* is not used in *tester1.py* even it takes width as its argument.
- *faults* is used for the discovered failure as mentioned in *projects.txt*.
- *coverage* is used for the coverage report as mentioned in *projects.txt*.
- *running* is used for the running information as mentioned in *projects.txt*.

I checked contracts by:

- Whether *sut.safely* returns *True* or not
- Whether *sut.check* returns *True* or not

I checked filters by:

- We can get type of class by *sut.actionClass*. We can also get all classes of actions by *sut.actionClasses*. So, I can count how many times that I execute for each action class. I checked that number for each executing sequence. If the sequence has too much execution times for one particular action class. I throw the sequence.

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*, ICSE'07, pages 75-84, 2007.