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

Part 3
Competitive Milestone 2
Feedback-directed Random Test Generation* in *TSTL

Introduction
In this document, I would like to explain how my *tester2.py* works and how efficiency it was to find bug for *sut.py* created by *avl.py* and *avl.tstl* under *cs569sp16/SUTs* on github repository. Same with *tester1.py*, I implemented *tester2.py* based on *GenerateSequences* in Figure 3 in the paper, *Feedback-directed random test generation* [1].

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

1 *errorSeqs* $\leftarrow \{\}$ *// Their execution violates a contract.*
2 *nonErrorSeqs* $\leftarrow \{\}$ *// 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.

How my *tester2.py* works

The following snippet codes are the main part in my *tester2.py*.

```
01 while time.time() - start < timeout:
02     seq = rgen.choice(nseqs)[:]
03     sut.replay(seq)
04     if rgen.randint(0, 9) == 0:
05           n = rgen.randint(2, 100)
06           ok, propok, classTable, timeover =
           genAndExeSeq(n, seq, eseqs, nseqs)
07     else:
08           ok, propok, classTable, timeover =
           genAndExeSeq(1, seq, eseqs, nseqs)
09     if filters(seq, ok, propok, classTable):
10           nseqs.append(seq)
```

NOTE:

seq: list of actions
nseqs: list of non-error seqs
eseqs: list of error seqs

The above codes work as:

1. Repeat STEP 2 to STEP 4 until timeout (line 01).
2. Randomly pick seq from nseqs and replay it (line 02 - 03).
3. Generate and execute sequence by (line 04 - 08):
    a. Repeat STEP 2-b and STEP 2-c 1 time in 90% of probability.
    Repeat STEP 2-b and STEP 2-c n times in 10% of probability where n is between 2 and 100 in equally likely.
    b. Pick up an enable action randomly and execute it.
    c. If it is executed without error, append it into seq.
    Otherwise, append it into seq, and then, append seq into eseqs.
4. Check seq in STEP 2 is whether we should add into nseqs or not by (line 09 - 10):
    a. Length of seq is whether less and equal than depth or not.
    b. The maximum number of actionclass of actions that we append to seq in STEP 2 is whether less and equal than width or not.
    c. Whether actions that we append to seq in STEP 2 are executed without error or not.

How efficiency it was to find bug

I evaluated how efficiency my *tester2.py* by following conditions:

- System under test:
  Created *sut.py* by using *avl.py* and *avl.tstl* in *cs569sp16/SUTs* on class repository.
  In *avl.py*, there is bug such that it crashes when we insert 1, 2, 3, and 4 into AVL.
- Command line:
  python tester2.py 300 1 100 10 1 1 1
- How many times to run the above command line:
  100 times

So, I ran my *tester2.py* with 300 seconds 100 times to see how efficiency it was to find the bug of inserting 1, 2, 3 and 4 into AVL.

Here is the statistic results:

- 81 times found the bug
- 264 seconds to find the bug in average with 26 standard division
  231 seconds to find the bug in best case
  299 seconds to find the bug in worst case
- 190 branch and 141 statement coverage if it found the bug
  187 branch and 140 statement coverage otherwise
- Length of seq that detected the bug was 91

According to the statistic results, tester2.py can detect the bug of inserting 1, 2, 3, and 4 in high probability in 300 seconds with high coverage cover.

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.