CS 569 Tester 2 Report

Improved Random Test Generation in TSTL APIs

Yu-Chun Tseng

May 18, 2016

1. Background

Random testing is a black-box software testing, also known as monkey testing, where programs are tested by generating random and independent inputs. Results of the output are compared against software criterions to verify that the test output is success or failure. In case of lacking of criterions, the expectations of the language are used that means if an expectation appears during test execution then it shows there is a bug in the program. However, random testing only finds basic bugs (e.g. Null pointer dereferencing) and is as precise as the criterion and criterions are basically imprecise. Moreover, according to the in-class experiment, we can find that utilizing pure random testing, such as BFS, in TSTL to test Python programs often causes unrelated operations that cannot lay over every code line of the programs. Such portion of the code that we cannot cover could have bugs.

2. Approach

In tester1, my approach did not work well. Program crush happened. So I revise my algorithm. The basic idea is that an object-oriented unit consists of a sequence of method calls that set up state, such as creating and mutating objects, and an assertion about the result of the final call.

The algorithm is as follows.

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

```
errorSeqs ← {} // Their execution violates a contract.
nonErrorSeqs ← {} // Their execution violates no contract.
While timeLimit not reached do
        // Create new sequence.
        m( $T_1$ ... $T_k$ ) ← randomPublicMethod(classes)
        <seqs, vals> ← randomSeqsAndVals(nonErrorSeqs, $T_1$ ... $T_k$ )
        newSeq ← extend(m, seqs, vals)
        // Discard duplicates.
        if newSeq ∈ nonErrorSeqs ∪ errorSeqs then
            continue
        end if
        // Execute new sequence and check contracts.
        <$\underset{o}{\rightarrow}$, violated> ← execute(newSeq, contracts)
        // Classify new sequence and outputs
        if violated = true then
            errorSeqs ← errorSeqs ∪ {newSeq}
        else
            nonErrorSeqs ← nonErrorSeqs ∪ {newSeq}
            setExtensibleFlags(newSeq, filters, <$\underset{o}{\rightarrow}$ ) // Apply filters.
        end if
end while
return <nonErrorSeqs, errorSeqs>
```

3. Usage

There are 7 arguments required sequentially in the experiment.

<timeout>: time in seconds for testing

<seed>: seed for Python Random.random object used for random number generation in the code, if it is stochastic

<depth>: maximum length of a test generated by the algorithm

<width>: maximum memory/BFS queue/other parameter that is basically a search width

<faults>: either 0 or 1 depending on whether the tester should check for faults in the SUTs; if true, save a test case for each discovered failure in the current directory

<coverage>: either 0 or 1 depending on whether a final coverage report should be produced, using TSTL's internalReport() function.

<running>: either 0 or 1 depending on whether running info on branch coverage should be produced

4. Experiment Result

```
python tester2.py 20 1 100 0 1 1 1
Total Actions: 24531
Total Runtime: 20.0025942326
TSTL BRANCH COUNT: 187
TSTL STATEMENT COUNT: 140
```

5. References

[1]. Groce, Alex, el al. "TSTL: a language and tool for testing." Proceedings of 2015 International Symposium on Software Testing and Analysis. ACM, 2015.

[2]. Pacheco, Carlos, el al. “Feedback-directed random test generation.” Software Engineering, 2007. ICSE 2007. 29th International Conference. IEEE, 2007.