CS 569 Tester 1 Report

Enhanced Random Test Generation in TSTL APIs

Yu-Chun Tseng

May 3, 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

I would like to propose a test generation algorithm based on random testing in TSTL APIs. My intuitive thought is to modify random testing. Subsequent random operations are chosen related to SUT states being observed to cover and exploring new statements as many as possible at the same time. My major idea is to give a time budget ***t*** and a depth ***d***, the length

of random operations being executed, then the frequency assignment of all statements in the program being browsing should be equal.

In order to achieve this goal, I give $\frac{t}{k}$ time budget of the whole time budget to execute pure random testing at Phase 1 first. This operation browses initial statements which are all covered easily. d continuous random accesses are executed in this condition to search for new statements. In the case of finding new statements, I place the information of new statements in two individual arrays. After d continuous random accesses, I measure the frequency of browsing each statement that is covered.

After then, I compute the mean μ and the standard deviation σ of the frequency of all statements to decide the outsiders of statements.

$$\mu = \frac{\sum_{i=1}^{N} f_i}{N}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (f_i - \mu)^2}{N}}$$

where f_i is the frequency of the statement s_i that is covered.

I also label the statement outsiders as o based on the threshold τ, where $\tau = \mu - (0.67) * \sigma$. Hence, $o = \{s_i \mid f_i \leq \tau\}$ is the set of statements.

After o is decided, I choose the states of SUT that can search for the particular statements at Phase 1 ordered depending on minimum coverage statement. Later, I run random testing on these states in order to find new statements. I give the time budget for each state related to the statement that it lays over at Phase 1. For example, states that search for minimum statements would be given most time budget and so on. For each state has been utilized, I also name d as consecutive random operations and measure the browsing frequency of all statements continuously at Phase 1. If there is

a new statement found, the state that finds this new statement would be added in the array. The point is to utilize the state that discover new statements and to keep the time budget for those unimportant states.

3. References

[1]. Alex Groce, Gerard Holzmann, and Rajeev. Randomized Different Testing As a Prelude to Formal Verification. In Software Engineering, 2007. ICSE 2007. 29th International Conference on, pages 621-631. IEEE, 2007.

[2]. Richard Hamlet. Random Testing. Encyclopedia of Software Engineering, 1994