Guiding Random Test Generation with Program Analysis

Lei Ma[1], Cyrille Artho[2], Cheng Zhang[3], Hiroyuki Sato[4], Johannes Gmeiner[5] and Rudolf Ramler[6]

Abstract: Random test generation is effective in creating method sequences for exercising the software under test. However, black-box approaches for random testing are known to suffer from low code coverage and limited defect detection ability. Analyzing the software under test and using the extracted knowledge to guide test generation can help to overcome these limitations. We developed a random test case generator augmented by a combination of six static and dynamic program analysis techniques. Our tool GRT (Guided Random Testing) has been evaluated on real-world software systems as well as Defects4J benchmarks. It outperformed related approaches in terms of code coverage, mutation score and detected faults. The results show a considerable improvement potential of random test generation when combined with advanced analysis techniques.

Keywords: Random testing, program analysis, static and dynamic analysis.

Random approaches for testing object-oriented programs can effectively generate sequences of method calls to execute the objects of the system under test (SUT). The test data for input parameters are either constant values in case of primitive data types or objects retuned by already generated method sequences, which can be used as inputs for further test generation. The generation process incrementally builds more and longer test sequences by randomly selecting methods and reusing previously generated method sequences (that return objects) as input until a time limit is reached.

While being highly automated and easy to use, random testing may suffer from low code coverage and limited defect detection ability when applied to real-world applications. It is considered unlikely that random approaches are able to exercise all "deeper" features of a reasonably-sized program by mere chance. These limitations are due to the adoption of a black-box approach without using application-/implementation-specific knowledge. Mining and leveraging information about the SUT can provide a valuable aid to guide random testing and, thus, to overcome such limitations.

We developed an approach for random test generation, Guided Random Testing (GRT) [Ma15], which has been augmented by an ensemble of six static and dynamic program analysis techniques. They are used to extract and incorporate information on program

[1] University of Tokyo, Japan, malei@satolab.itc.u-tokyo.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST), Japan, c.artho@aist.go.jp
[3] University of Waterloo, Canada, c16zhang@uwaterloo.ca
[4] University of Tokyo, Japan, schuko@satolab.itc.u-tokyo.ac.jp
[5] Software Competence Center Hagenberg (SCCH), Austria, johannes.gmeiner@scch.at
[6] Software Competence Center Hagenberg (SCCH), Austria, rudolf.ramler@scch.at

types, data, and dependencies in the various stages of the test generation process. The overall effectiveness of GRT results not only from applying each of the individual techniques, but also from their combination and orchestration. Program information is extracted by some components at specific steps and passed to others to facilitate their tasks.

Fig. 1 shows the different techniques and how they are interacting. First, the SUT is statically analyzed. *Constant mining* extracts constant values to create a diverse yet application-specific input data set for test generation. The diversity is further increased by applying input fuzzing and by favoring methods that change the state of input objects as a side effect of their execution, which is determined by *Impurity analysis*. Information about dependencies between methods is used by the technique *Detective* for constructing method sequences returning input objects that are not in main object pool. *Elephant brain* manages all the objects stored in the main object pool including exact type information. Coverage information is recorded throughout test generation and is used by *Bloodhound* to select methods not well covered so far. *Orienteering* estimates the execution time of each method sequence to accelerate the overall generation process.

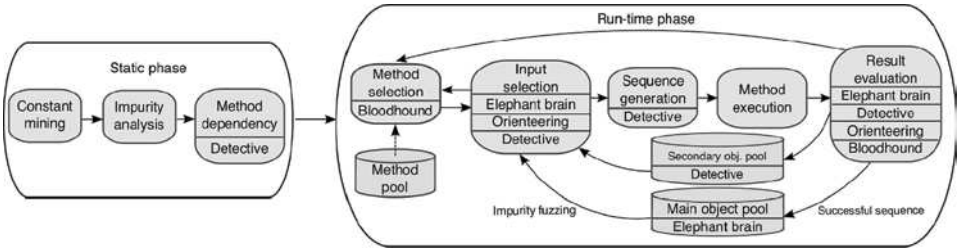

Fig. 1: Static and dynamic program analysis techniques included in GRT.

GRT has been evaluated on 32 real-world projects and, when compared to other tools (Randoop and EvoSuite), outperformed major peer techniques in terms of code coverage (by 13%) and mutation score (by 9 %). On the four studied benchmarks from Defects4J, which contain 224 real faults, GRT also showed better fault detection capability, finding 147 faults (66 %). Furthermore, in an in-depth evaluation on the latest versions of ten popular open source projects, GRT successfully detected over 20 previously unknown defects that were confirmed by the developers.

The results indicate that random testing has not yet reached its limits. There is still a considerable potential for further improving random test generation approaches by incorporating advanced analysis techniques – a path we plan to follow in our future work.

References

[Ma15] Ma, L.; Artho, C.; Zhang, C.; Sato, H.; Gmeiner, J.; Ramler, R.: GRT: Program-Analysis-Guided Random Testing. Proc. 30th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2015), Lincoln, Nebraska, USA, November 2015.