System Testing and Program Verification

Stephan Arlt Sergio Feo-Arenis Andreas Podelski Martin Wehrle

University of Freiburg University of Basel

{arlt,arenis,podelski}@informatik.uni-freiburg.de martin.wehrle@unibas.ch

Abstract: The effectiveness of black-box system testing can be increased by automatic program verification techniques. For example, the redundancy of a test case can be detected by static analysis; the analysis must be applied to a program in the 'white-box' layer of the system under test (e.g., in the setting of GUI testing, to the program which defines the event handlers). We will investigate the question of how automatic program verification techniques can be used to reduce the cost of testing and at the same time provide a guarantee for test coverage.

Overview. Testing and verification are two sides of the same coin. The corresponding techniques can complement each other. In particular, automatic program verification techniques can be used in system testing for a more informed selection of test cases. We have instantiated the idea for GUI testing, where a system is tested through a sequence of events which are triggered via a graphical user interface (GUI). As described in [APW14, AEFAP14, ABSP12, APB+12], we apply a static analysis to the event handler programs of the system. The static analysis infers verifiable statements about the runtime behavior of the programs and thus helps to determine the relevance resp. the redundancy of a test case (where relevance and redundancy have a precise, formal sense). We will next explain the context and the approach in greater detail.

The main challenge in GUI testing is to generate test cases selectively. So-called iterative approaches as in Gross et al. [GFZ12] and Mariani et al. [MPRS12] generate test cases on-the-fly (i.e., while executing the system under test). Here, the size of the test suite is *a priori* unbounded (in practice, timeouts are used for the overall test).

In contrast, non-iterative approaches generate test cases *offline* (before executing the system under test). Here, the size of the test suite can be bounded *a priori* (e.g., by fixing the length of event sequences). However, many event sequences in the test suite will not be *relevant* for detecting bugs (e.g., because the events in the sequence do not relate to each other), and many event sequences will not be *feasible* (e.g., because the button to trigger an event in the sequence is not visible). So-called black-box techniques avoid many infeasible test cases by using a black-box model such as an Event Flow Graph (EFG); the model is constructed by observing the order of events in sample executions of the system [Mem07].

The above-mentioned approach of [APW14, AEFAP14, ABSP12, APB+12] integrates automatic verification techniques with black-box techniques. In a first step, a static analysis on the event handler programs is used to compute sequences of events that are interconnected by def-use relationships. The second step uses the black-box model to turn each sequence into one that is feasible according to the black-box model. The approach thus integrates the information of what is relevant with the information of what is feasible.

The original approach in [APB+12] only considers sequences that contain a pair of two matching def-use events. Even with that restriction, the number of event sequences can

be prohibitively large (i.e., the test suites cannot be executed in reasonable time). Moreover, some complex bugs call for event sequences with more than just two inter-connected def-use events. This leads to the extension of the approach in [APW14] which uses static analysis based on a variant of program slicing to further reduce the resulting number of test cases. The approach is scalable even though it generates event sequences with more than two inter-connected def-use events as test cases. The approach uses a formal notion of redundancy. It can be proven that eliminating a redundant event sequence from a test suite does not affect the effectiveness of the test (wrt. the set of errors found by the test).

Static analysis based on slicing can identify redundant sequences and thus lead to a drastic test suite reduction. Experiments on real-world GUI applications indicate the potential of the approach. In particular, experiments show that redundant event sequences occur in a huge number of GUI applications. Moreover, the reduced number of test cases can strongly reduce the overall execution time of a GUI test suite without affecting test effectiveness.

Open Questions. The approach opens a broad range of general questions. What are the possibilities and the limitations of approaches to increase the effectiveness of black-box testing by automatic verification techniques (techniques based on static analysis and decision procedures for handling data)? We know that a high rate of warnings emitted by a static analysis indicates a high rate of faults, but we can also ask about the consequences of the absence of warnings: Can formally verified statements about the absence of 'low-level' runtime errors (program crashes) increase the effectiveness of 'high-level' system testing requirements (i.e., geared towards detecting 'serious' violations of functional requirements)? More generally, how can we make the results of a static analysis (i.e., statements about the behavior according to a model or to a programming language semantics) more conclusive about the behavior of the actual, running system? Can we use full-fledged automatic program verification techniques which compute contracts for the function calls in the system under test (e.g., for the functions of the GUI toolkit)? Can one exploit the modularity of these techniques to obtain sufficient scalability? Can these techniques help to further reduce the cost of testing? Can they help to provide guarantees for strong notions of coverage or, in some cases, even notions of exhaustiveness? The above questions indicate that research on system testing with program verification techniques is just at its beginning.

References

[ABSP12] S. Arlt, P. Borromeo, M. Schäf, and A. Podelski. Parameterized GUI Tests. In *ICTSS*, pages 247–262. Springer, 2012.

[AEFAP14] S. Arlt, E. Ermis, S. Feo-Arenis, and A. Podelski. Verification of GUI Applications: A Black-Box Approach. In *ISoLA*, pages 236–252. Springer, 2014.

[APB+12] S. Arlt, A. Podelski, C. Bertolini, M. Schäf, I. Banerjee, and A.M. Memon. Lightweight Static Analysis for GUI Testing. In *ISSRE*, pages 301–310. IEEE, 2012.

[APW14] S. Arlt, A. Podelski, and M. Wehrle. Reducing GUI test suites via program slicing. In *ISSTA*, pages 270–281. ACM, 2014.

[GFZ12] F. Gross, G. Fraser, and A. Zeller. Search-based system testing: high coverage, no false alarms. In *ISSTA*, pages 67–77. ACM, 2012.

[Mem07] A.M. Memon. An event-flow model of GUI-based applications for testing. *Softw. Test., Verif. Reliab.*, 17(3):137–157, 2007.

[MPRS12] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro. AutoBlackTest: Automatic Black-Box Testing of Interactive Applications. In *ICST*, pages 81–90. IEEE, 2012.