A Case for White-box Testing Using Declarative Specifications
Poster Abstract

Danhua Shao Sarfraz Khurshid Dewayne E. Perry
Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712
{dshao, khurshid, perry}@ece.utexas.edu

Software testing, the most commonly used technique for validating the quality of software, is a labor intensive process, and typically accounts for about half the total cost of software development and maintenance. Automating testing not only reduces the cost of producing software but also increases the reliability of modern software.

White-box testing and black-box testing are two commonly used techniques that have complementary strengths. White-box testing uses the internal structures (such as control flow or data flow) of programs. Black-box uses an external interface.

Automated approaches to black-box testing make extensive use of specifications, e.g., to specify test inputs or test oracles. In unit testing of object-oriented code, *preconditions,* which define constraints on legal method inputs, and *postconditions*, which define expected behavior and outputs, form an integral part of specifications.

Black-box testing using TestEra. TestEra [1] is a specification-based black-box testing for Java programs. Using a Java method's precondition, TestEra automatically generates all nonisomorphic test inputs up to the given bound, executes the method on each test input, and uses the method postconditions as an oracle to check the correctness of each output.

Although TestEra provides efficient enumeration of structurally complex data structures according to given constraints, it is not efficient in generating tests for code coverage. A precondition does not specify a method's implementation. For example, consider the `contains()` method for `LinkedList`, which implements doubly-linked circular lists. Four test cases are enough to cover all the branches. However, TestEra generates 148 test cases.

White-box testing using TestEra. To extend the existing specification-based testing to support white-box testing, we propose a novel framework, Whispec. Given a Java method's precondition as a declarative constraint, Whispec systematically integrates it with the control flow of the program and generates test inputs to maximize the code coverage.

Alloy as an enabling technology. In Whispec, the specifications are declared as first-order logic formulas. As an enabling technology, Whispec uses the SAT-based Alloy toolset [2]. Alloy is a first-order declarative language based on sets and relations. Alloy Analyzer is an automatic tool that finds *instances* of Alloy specifications, i.e., finds assignments of values to the sets and relations in the specification such that the specification formulas evaluate to true. Instances of the constraints are translated to concrete inputs to run on the program.

Combining preconditions with desired path conditions. The key idea of Whispec is the integration of both preconditions and path conditions [3] using relational logic so that they are solved together for test generation. Given the precondition of the method under test, we first solve it with the Alloy Analyzer. The solution is concretized to a test input. Next, we execute the method on that input and build path conditions by negating some branch predicates in the execution path. We run the analyzer on a conjunction of the precondition and one of the generated path conditions. The solutions are translated to test inputs which execute previously unexplored paths. The iterative execution of Whispec can systematically enumerate inputs that maximize code coverage.

Preliminary results using our prototype indicate that for achieving a desired level of code coverage, Whispec generates significantly fewer tests than TestEra.

References

[1] D. Marinov and S. Khurshid. TestEra: A novel framework for automated testing of Java programs. *ASE'01*
[2] D. Jackson. *Software Abstractions: logic, language, and analysis.* MIT Press, Cambridge, MA, 2006.
[3] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, Volume 19, Issue 7, July 1976.