Systematic Software Testing: The Korat Approach

ACM SIGSOFT Impact Paper Award 2012

Chandrasekhar Boyapati
Google Inc.
Mountain View, CA 94043
boyapati@google.com

Sarfraz Khurshid
University of Texas
Austin, TX 78712
khurshid@utexas.edu

Darko Marinov
University of Illinois
Urbana, IL 61801
marinov@illinois.edu

ABSTRACT

At ISSTA 2002, the three authors (then Ph.D. students) published the paper "Korat: Automated Testing Based on Java Predicates", which won one of the first ACM SIGSOFT Distinguished paper awards. In 2012, the paper won the ACM SIGSOFT Impact Paper Award. The authors briefly recount the motivation behind Korat research, the ideas presented in the original paper, and some work it inspired.

1. BRIEF KORAT STORY

Testing, the most commonly used methodology for validating the quality of software, is conceptually simple: create some inputs, run them against the program, and check its outputs. In practice, however, testing is expensive, often requiring much manual effort, and ineffective, often failing to find crucial bugs. Automated testing can substantially reduce the cost of software development and maintenance.

A promising approach for automation is *specification-based testing*, where specifications, which describe *expected* program behavior, provide the basis for test input generation and correctness checking. Researchers realized the importance of specifications in testing over three decades ago [3]. However, effective techniques for programs written in commonly used languages remained scarce and ad hoc.

At ASE 2001, the second and third authors published TestEra [7], which was the first specification-based technique for *bounded-exhaustive* testing of object-oriented programs. The key idea of TestEra was to capture the properties of the desired inputs using a logical formula, termed an *input constraint*, enumerate the desired inputs using systematic constraint solving, and check the correctness of the program under test by (1) running it against *all* non-isomorphic inputs within a given bound on the input size and (2) checking each input/output pair using an *output constraint*, which states the expected relation between inputs and outputs and serves as a *test oracle*. TestEra used declarative constraints written in the Alloy language [4], based on first-order logic, and used Alloy's SAT-based back-end for input generation and correctness checking. TestEra enabled systematic testing of Java programs in the spirit of the bounded-exhaustive checking that the Alloy tool-set provided for declarative models.

During a research presentation on TestEra by the third author, the first author, who was not familiar with the Alloy language, asked whether constraints written in Java could be used to provide the same functionality as TestEra. The ensuing effort to support Java constraints resulted in the Korat approach for systematic testing, which was first presented at ISSTA 2002 [1]. Korat introduced the idea of using declarative constraints written in an imperative language for bounded-exhaustive testing and presented a dedicated solver for such constraints. The main insight into the Korat solver was *execution-driven pruning and isomorphism breaking*, where executions of the given input constraint on select candidate inputs provided the basis of pruning the space of all possible inputs and generating exactly those inputs that were valid and non-isomorphic. A key contribution of Korat was enabling specification-based testing *without* requiring the specifications to be written in a language greatly different from the underlying programming language – a requirement of all previous approaches for specification-based generation of test inputs.

Systematic test generation using input constraints brought the spirit of model checking to checking properties of programs that operate on structurally complex data. Previous work on software model checking focused on properties of control, such as deadlocks, but not on data, such as data structure invariants. Systematic testing was able to find subtle bugs in a number of applications [6]. The ideas at the heart of Korat provided the foundation for other complementary techniques, including symbolic execution of programs with structurally complex inputs [5], glass-box software model checking [9], and runtime error recovery using data structure repair [2]. Korat has been used for parallel and incremental test generation and execution for enhanced efficiency and effectiveness [8, 10]. Korat is available online for public download: "`http://mir.cs.illinois.edu/korat`".

2. REFERENCES

[1] C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on Java predicates. In *ISSTA*, 2002.

[2] B. Elkarablieh. *Assertion-based Repair of Complex Data Structures*. PhD thesis, UT Austin, 2009.

[3] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE TSE*, June 1975.

[4] D. Jackson. Alloy: A lightweight object modeling notation. *ACM TOSEM*, 11(2), Apr. 2002.

[5] S. Khurshid, C. Pasareanu, and W. Visser. Generalized symbolic execution for model checking and testing. In *TACAS*, 2003.

[6] D. Marinov. *Automatic Testing of Software with Structurally Complex Inputs*. PhD thesis, MIT, 2005.

[7] D. Marinov and S. Khurshid. TestEra: A novel framework for automated testing of Java programs. In *ASE*, 2001.

[8] S. Misailovic, A. Milicevic, N. Petrovic, S. Khurshid, and D. Marinov. Parallel test generation and execution with Korat. In *ESEC/FSE*, 2007.

[9] M. E. Roberson. *Glass Box Software Model Checking*. PhD thesis, U. Michigan, Ann Arbor, 2011.

[10] J. H. Siddiqui. *Improving Systematic Constraint-driven Analysis using Incremental and Parallel Techniques*. PhD thesis, UT Austin, 2012.