Electronic Communications of the EASST
Volume XXX (2015)

Proceedings of the
15th International Workshop on
Automated Verification of Critical Systems (AVoCS 2015)

Yacon: Using Equivalence Partitioning for Invariant Detection

Rosemary Monahan, Worakarn Isaratham

3 pages

Guest Editors: Gudmund Grov, Andrew Ireland
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Yacon: Using Equivalence Partitioning for Invariant Detection

Rosemary Monahan, Worakarn Isaratham

Department of Computer Science, Maynooth University, Ireland

Abstract: We investigate an approach to automatically detect program assertions for use in program verification. Our preliminary investigations recover partitioning information from a given test suite. These can be used in the detection of assertions.

Keywords: Program Verification, Dynamic Invariant Detection, Test Suites, Equivalence Partitioning, Boundary Analysis, Daikon, Yacon

1 The Problem

Most programmers have assertions about their program properties in mind when they develop software [Ern00]. However, these are not often explicitly expressed. Likewise, automated test suites require their author to have an understanding of such program properties. Our research investigates how to utilize test suites as an information source, to assist the discovery of assertions for use in program verification.

2 Our Hypothesis

Daikon [EPG+07] is a mature implementation of an approach to discovering assertions from program executions. This approach, known as 'Dynamic Invariant Detection', works by observing and reporting properties that are *invariably* true for every execution of the program. Hence, Daikon refers to program assertions as invariants. *Conditional invariants*, which have the form $p \Rightarrow q$ present a challenge to Daikon, as trying all of the possible conditional predicates is computationally infeasible [DLE03].

We hypothesise that test suites are an excellent source of conditional invariants. A well-organised test suite typically consists of a collection of test cases that follow the basic *four-phase test* pattern – setup, exercise, verify, and teardown [Mes07]. Non-trivial software usually behaves differently given the same *exercise*, depending on the *setup* step. In other words, the setup present in the test suites forms the conditional invariant. Our proposed research investigates the use of test suites in the discovery of conditional invariants.

3 Preliminary Research

For Daikon to detect a conditional invariant $p \Rightarrow q$, the antecedent p needs to be supplied, in the form of a *splitting condition*. We created *Yacon*, a prototype implementation of our solution. Its goal is to produce splitting conditions for Daikon from test suites. Its operation is divided into 2 phases: an *extraction* phase analyses a given test suite to produce an intermediate file called the *partition file*, which will then be converted into splitter files in a *translation* phase.

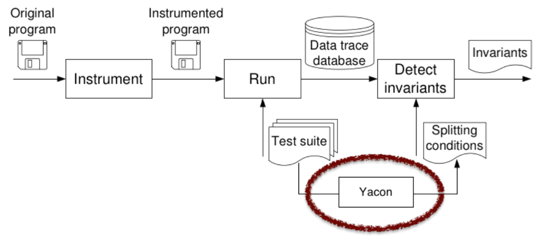

Figure 1: Yacon as an additional component to the Daikon system

Extraction is a complicated process of employing heuristics, called *recovery strategies*, to deduce appropriate splitting conditions. We experiment with 2 such strategies, Boundary Value Recovery and Test Suite Invariants, to discover *equivalence classes* on the inputs of methods. Each equivalence class can then be interpreted as a splitting condition. The translation phase is a straightforward process of converting one file format to another, generate splitting conditions from partitioning information.

4 Preliminary Results

Our *partitioning recovery effectiveness* metric suggests that from the two recovery strategies used, there is no clear dominant strategy – each one can work better in different circumstances. However, the combination of both strategies produces the best result most of the time.

The number of expected invariants that are discovered using Yacon is higher than those discovered with Daikon but not as high as those generated from manually-written splitting conditions. We found that by trying to improve the recall rate, Yacon produced many more invariants, with a lot of them incorrect or irrelevant, according to the *correctness*, *usefulness*, and *precision* metrics we defined. Our runtime also increases by a factor of 1.5-5. For full details of our results see github.com/arkorwan/yacon/wiki/Experimental-Results.

5 Some Concluding Research Questions

Our preliminary research suggests that test suite information can assist in uncovering conditional invariants but our recovery strategies have not fulfilled the potential yet. Resulting research questions include: How can we effectively use test suite information to detect program assertions? What recovery strategies can be used to deduce appropriate information (particularly splitting conditions) from test suites? How can we best evaluate the use of these strategies? What metrics are most effective? Can we automatically assess the quality of the generated assertions? Is this approach applicable in a more general setting? Is our approach suitable for generating assertions for legacy code which requires formal specification in order to be automatically verified.

Acknowledgements: Thanks to the European Commission who supports Mr Isaratham via a scholarship on the Erasmus Mundus MSc in Dependable Software Systems (DESEM).

Bibliography

[DLE03] N. Dodoo, L. Lin, M. D. Ernst. Selecting, refining, and evaluating predicates for program analysis. Technical report MIT-LCS-TR-914, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, MA, 2003.

[EPG^{+}07] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, C. Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69(1):35–45, 2007.

[Ern00] M. D. Ernst. *Dynamically discovering likely program invariants*. PhD thesis, University of Washington, 2000.

[Mes07] G. Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.