Software Model Checking via Static and Dynamic Program Analysis

Patrice Godefroid

Bell Laboratories, Lucent Technologies, god@bell-labs.com

Abstract

In this tutorial, I will review and discuss the current approaches to software model checking. The presentation is intended to provide a representative, but not exhaustive, overview of this active area of research. I will emphasize techniques and ideas within my area of expertise. The tutorial consists of three parts.

Part I: The Dynamic Approach (Systematic Testing)

Part I presents the dynamic approach to software model checking. This approach, pioneered in VeriSoft [6], consists of systematically exploring the state space of a concurrent software system by driving its executions via a run-time scheduler. Several technical innovations made this possible: the use of a run-time scheduler to systematically drive process executions, a construct to simulate nondeterminism at run-time, and the use of partial-order reduction to make a search in the state space of concurrent operating-system processes tractable even without storing any intermediate states in memory. Many of these features have now been adopted in other software model checkers (like Java PathFinder [17], CMC [16], etc.).

In this part, I will present VeriSoft and the dynamic approach to software model checking. I will also discuss applications, strengths and limitations, as well as technology-transfer issues that arise when applying software model checking in an industrial environment [8].

Part II: The Static Approach (Automatic Abstraction)

Part II presents the static approach to software model checking. This approach consists of automatically extracting a model out of a software application by statically analyzing its code and abstracting away details, applying traditional model checking to analyze this abstract model, and then mapping abstract counter-examples back to the code or refining the abstraction (e.g., [1, 15, 4]). In this part, I will start by reviewing the basic notions of software model checking via abstraction, including a brief introduction to predicate abstraction.

I will then present recent work on *three-valued abstractions* [2], in which properties of a system are either true, false or unknown. Whereas traditional conservative abstractions can only prove universal properties, model checking three-valued abstractions (also called *may/must abstractions* [12]) can be used to both

prove and disprove any temporal-logic property. Also, verification results can be more precise with *generalized model checking* [3], which checks whether there exists a concretization of an abstraction satisfying a temporal-logic formula. Generalized model checking generalizes both model checking (when the model is complete) and satisfiability (when everything in the model is unknown), probably the two most studied problems related to temporal logic and verification. I will discuss algorithms and complexity bounds for three-valued model checking and generalized model-checking for various temporal logics [3, 11]. Then, I will discuss applications to program verification via automatic abstraction [10]. I will show examples of programs and properties that can be verified by generalized model checking but not with current abstraction-based verification tools [11]. I will also present classes of temporal-logic formulas for which model checking is guaranteed to always have the same precision as generalized model checking [9]. Finally, I will briefly discuss three-valued abstractions for reasoning about open systems [7] and about games in general [5], as well as completeness issues (i.e., given an infinite-state program and a property, is there a finite-state abstraction of that program that satisfies this property?).

Part III: Combining the Static and Dynamic Approaches

The dynamic (systematic testing) and static (automatic abstraction) approaches to software model checking inherit the well-known advantages and limitations of, respectively, dynamic and static program analysis, and are therefore complementary [13, 8].

Part III discusses some current work to improve on the two previous approaches by combining their strengths, namely the precision of dynamic analysis and the efficiency of static analysis. In particular, I will present such a recent approach called *Directed Automated Random Testing* (DART) [14], which fully automates software testing by combining three main techniques: (1) *automated* extraction of the interface of a program with its external environment using lightweight static analysis; (2) automatic generation of a test driver for this interface that performs *random* testing to simulate the most general environment the program can operate in; and (3) dynamic analysis of how the program behaves under random testing and automatic generation of new test inputs that *direct* the execution along alternative program paths. The main strength of DART is that testing can be performed completely automatically on any program that compiles – there is no need to write any test driver or harness code. Since DART attempts to sweep through all the feasible execution paths of a program using dynamic test generation techniques (Step 3), it can be viewed as one way of combining static (interface extraction, symbolic execution) and dynamic (testing, run-time checking) program analysis with model-checking techniques (systematic state-space exploration).

In this part, I will present DART, compare static and dynamic test generation, and discuss applications and experimental results obtained on C program benchmarks. I will also quickly present a new (yet unpublished) algorithm for

compositional dynamic test generation that is necessary to make the DART approach scalable to very large programs (hundreds of thousands of lines of code and more). I will conclude by discussing possible directions for future research in this area [13].

Acknowledgements. This tutorial is largely based on results presented in the papers found in the bibliography below. I am grateful to my co-authors for such inspired collaborations, and to my competitors (often good friends, actually) for challenging those views, contributing their own, and hence speeding up progress.

References

1. T. Ball and S. Rajamani. The SLAM Toolkit. In *Proceedings of CAV'2001 (13th Conference on Computer Aided Verification)*, volume 2102 of *Lecture Notes in Computer Science*, pages 260–264, Paris, July 2001. Springer-Verlag.
2. G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In *Proceedings of CAV'99 (11th Conference on Computer Aided Verification)*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287, Trento, July 1999. Springer-Verlag.
3. G. Bruns and P. Godefroid. Generalized Model Checking: Reasoning about Partial State Spaces. In *Proceedings of CONCUR'2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182, University Park, August 2000. Springer-Verlag.
4. J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting Finite-State Models from Java Source Code. In *Proceedings of the 22nd International Conference on Software Engineering*, 2000.
5. L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-Valued Abstractions of Games: Uncertainty, but with Precision. In *Proceedings of LICS'2004 (19th IEEE Symposium on Logic in Computer Science)*, pages 170–179, Turku, July 2004.
6. P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proceedings of POPL'97 (24th ACM Symposium on Principles of Programming Languages)*, pages 174–186, Paris, January 1997.
7. P. Godefroid. Reasoning about Abstract Open Systems with Generalized Module Checking. In *Proceedings of EMSOFT'2003 (3rd Conference on Embedded Software)*, volume 2855 of *Lecture Notes in Computer Science*, pages 223–240, Philadelphia, October 2003. Springer-Verlag.
8. P. Godefroid. Software Model Checking: The VeriSoft Approach. *Formal Methods in System Design*, 26(2):77–101, March 2005. Also available as Bell Labs Technical Memorandum ITD-03-44189G, March 2003.
9. P. Godefroid and M. Huth. Model Checking Vs. Generalized Model Checking: Semantic Minimizations for Temporal Logics. In *Proceedings of LICS'2005 (20th IEEE Symposium on Logic in Computer Science)*, pages 158–167, Chicago, June 2005.
10. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based Model Checking using Modal Transition Systems. In *Proceedings of CONCUR'2001 (12th International Conference on Concurrency Theory)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, Aalborg, August 2001. Springer-Verlag.

11. P. Godefroid and R. Jagadeesan. Automatic Abstraction Using Generalized Model Checking. In *Proceedings of CAV'2002 (14th Conference on Computer Aided Verification)*, volume 2404 of *Lecture Notes in Computer Science*, pages 137–150, Copenhagen, July 2002. Springer-Verlag.
12. P. Godefroid and R. Jagadeesan. On the Expressiveness of 3-Valued Models. In *Proceedings of VMCAI'2003 (4th Conference on Verification, Model Checking and Abstract Interpretation)*, volume 2575 of *Lecture Notes in Computer Science*, pages 206–222, New York, January 2003. Springer-Verlag.
13. P. Godefroid and N. Klarlund. Software Model Checking: Searching for Computations in the Abstract or the Concrete (Invited Paper). In *Proceedings of IFM'2005 (Fifth International Conference on Integrated Formal Methods)*, volume 3771 of *Lecture Notes in Computer Science*, pages 20–32, Eindhoven, November 2005. Springer-Verlag.
14. P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation)*, pages 213–223, Chicago, June 2005.
15. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages*, pages 58–70, Portland, January 2002.
16. M. Musuvathi, D. Park, A. Chou, D. Engler, and D. Dill. CMC: A pragmatic approach to model checking real code. In *Proceedings of OSDI'2002*, 2002.
17. W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *Proceedings of ASE'2000 (15th International Conference on Automated Software Engineering)*, Grenoble, September 2000.