Research Statement

Arie Gurfinkel

December 2005

As computers get faster and smaller, they are becoming more pervasive in our daily lives. At the same time, software systems that control them are becoming larger, more complex, and error prone. Software bugs lead to a loss of productivity, denial of service, and security breaches, cost millions of dollars to the economy, and sometimes cause a loss of human life. I believe that formal methods are indispensable for building reliable and trusted software systems of the future. However, formal methods are not yet widely used in industry outside of the safety-critical and hardware domains. An often cited reason is that they are perceived to require a level of mathematical sophistication beyond that possessed by many software developers. I believe that the key to addressing this problem is in constructing automated reasoning techniques and developing methodologies to support their use by software developers.

The dream of a "mechanical verifier" has been pursued by the research community since the 60s. However, only recently the success of such projects as Microsoft's Static Driver Verifier (SLAM) and the increased success of formal verification in hardware industry has clearly demonstrated that this dream is becoming a reality. The main goal of my research is to facilitate the construction of reliable and trusted computer systems by developing tools and methodologies to improve usability and to extend the scope of application of formal methods in software engineering.

I believe that the success in this area crucially depends on the combination of foundational, engineering and software engineering approaches. A strong theoretical foundation is necessary to understand and solve any problem. Building such a foundation requires developing new theories and discovering novel combinations of existing ones. At the same time, turning a theory into an automated analysis tool is often a significant engineering effort. Finally, an automated analysis tool is only effective when combined with a solid software engineering methodology that enables its use by practicing software engineers. In my research, I aim to combine the three approaches by starting from a real-life problem, developing a theoretical solution, engineering systems based on this solution, and finally conducting case studies to study the effectiveness of the approach.

Research Contributions

The focus of my research is on improving the scope and usability of formal methods in software engineering through automation, specifically, via model-checking. During my Doctoral studies, I have explored many aspects of this diverse and interesting problem: extending the scope of model-checking to new logical domains (multi-valued model-checking), source code verification by iterative abstraction (software model-checking), discovering properties of a design (temporal logic query-checking), and improving usability of existing techniques (vacuity detection and understanding of counterexamples).

Multi-Valued Model-Checking

Multi-valued logics, logics that extend Boolean logic of TRUE and FALSE with additional truth values, have long been used in Philosophy [Kle52, Fit02], Database Theory [Bel77], and Artificial Intelligence [Gin88] to capture reasoning with partial and inconsistent information. The most famous of these is the three-valued Kleene logic that extends Boolean logic with an additional value MAYBE to represent "a lack of truth value", and was originally proposed to reason about self-reference. Another is the four-valued Belnap

logic that additionally introduces an explicit value to represent an inconsistency, and has found a variety of applications in databases.

Muti-valued logics are a perfect match to many problems in Software Engineering, where one needs to deal with partial and inconsistent specifications, information loss due to abstraction, combining (potentially contradicting) requirements from multiple stakeholders, etc. A large part of my research has concentrated on extending automated analysis techniques (model-checking, in particular) to handle multi-valued logics. I have worked on all aspects of this problem: from development of the semantics of modeling formalisms and temporal logics and their model-checking algorithms [CDEG03, CDG01, Gur02], to analysis of the running time complexity [Gur02], to introducing a notion and efficient generation of witnesses and counterexamples [GC03a], to understanding the relationship between classical and multi-valued model-checking [GC03b], to designing and evaluating new data-structures [CGD+02], and finally, to implementing the first multi-valued symbolic model-checker χChek [CDG02] and conducting case studies [CDEG03].

Software Model-Checking

My most recent interest is *software model-checking*, a technique that allows application of model-checking directly on the source code of a program. The main challenge in this area is to automatically extract an abstraction of a program that is small enough to analyze, yet rich enough to enable conclusive analysis. We have developed a new abstraction technique based on Belnap logic that extends the scope of software model-checking to arbitrary CTL properties, can be used to both prove and refute a property, and often requires a smaller abstract model for analysis to be conclusive [GWC06]. Based on this technique, I have lead the development of a software model-checker YASM. YASM has been successfully applied to C programs including device drivers and parts of `OpenSSH`, and ranging from a few hundred to thousands of lines of code [GC06]. We are currently working on extending our abstraction to reasoning about recursive programs.

Temporal Logic Query-Checking

Temporal logic query-checking [Cha00] is a technique to discover properties of models. A temporal logic query is simply a temporal logic formula with "holes". A solution to a query is an assignment of propositional formulas to holes that transforms it into a valid formula. Queries can be used to determine the strongest propositional invariant of a system, what is guaranteed to follow a request, or the precondition for generating an acknowledgment. This process facilitates model exploration and model understanding.

We have shown that the query-checking problem can be naturally expressed as a multi-valued model-checking problem where values in the logic correspond to possible solutions to the query [GDC02, GCD03]. This significantly increases the scope of the applicability of query-checking by (a) extending the language of queries to allow for multiple "holes", and (b) by providing a query-checking algorithm to any logic for which a multi-valued model-checking problem has been defined. In comparison, Chan's original algorithm is limited to valid CTL queries – queries that are guaranteed to have a single solution. Based on this work, we have built TLQSolver [CG03], an efficient query-checker for CTL queries, by adapting our existing multi-valued model-checker χChek [CDG02].

Furthermore, we have explored several novel applications of query-checking based on the fact that, in addition to an answer, a multi-valued model-checker produces a witness to justify its analysis. In the case of queries, a witness is a set of traces justifying the solution(s) to the query. These traces can be used as a basis for *guided simulation* in which a query is used to encode a top-level objective of the simulation, and a witness provides values for the input variables to reach this objective. Another use of query-checking is for test-case generation, where a temporal logic query is used to encode test-coverage criteria. A witness to such a query is a (potentially minimal) collection of test-cases that achieve the desired coverage.

Vacuity Detection

One of the main pitfalls of formal verification is that it brings a false sense of security – if a system has passed formal verification, then obviously it is correct. However, before formal verification can be applied, one needs to provide a model of the system and its environment, and describe the requirements as a property in formal logic. If either of these is incorrect, the result of verification is meaningless. One way to improve confidence

in the verification result is to check that the property is satisfied non-vacuously, i.e., the result depends on every part of the property. For example, a property "a request is always eventually acknowledged" is satisfied *non-vacuously* only by a system that generates and eventually acknowledges a request, but it is satisfied *vacuously* by a system that either never generates a request (e.g., due to a bug in the model of the environment) or a system that continuously produces acknowledgments (e.g., due to a bug in the system). Researchers at IBM Haifa Research Lab have found that "... typically 20% of specifications pass vacuously during the first formal verification runs of a new hardware design, and that vacuous passes always point to a real problem in either the design, or its specification, or the environment" [BBDER01].

We have shown that the vacuity detection problem is an instance of multi-valued model-checking, where the additional truth values keep track of how a formula depends on its subformulas [GC04b]. In the process, we have introduced a more general notion of vacuity, *mutual vacuity*, that captures truth or falsity of a property, its vacuity with respect to subformulas, and vacuity with respect to different occurrences of the same subformula. In this case, the witnesses and counterexamples for multi-valued model-checking coincide with the notion of an interesting witness to non-vacuity, and give users all the necessary information for debugging vacuous properties.

Furthermore, we formalized the notion of *robustness* of vacuity [GC04a]. Intuitively, a definition of vacuity is robust if it is independent of any particular modeling formalism and property specification logic, i.e., it cannot be "fixed" just by tricking the analysis tool. This allowed us to combine vacuity detection and abstraction [GC04a], which is essential for any practical application of vacuity detection since model-checking of any non-trivial system requires some from of abstraction.

Finally, we have developed VaqUoT, a tool that combines our vacuity detection algorithm with the state-of-the-art symbolic model-checker NuSMV, and have conducted experiments to study its effectiveness [GGC05].

Understanding Counterexamples

When a desired temporal property fails, a model-checker can generate a counterexample – a trace explaining the reason for the failure. Counterexamples significantly speedup the typical Check/Analyze/Fix cycle during the early stages of development, and are indispensable for the Counterexample-Guided Abstraction Refinement (CEGAR) process employed by many software model-checkers. The counterexample generation ability has been one of the major advantages of model-checking compared to other verification techniques.

Existing model-checkers such as NuSMV are limited to generating linear counterexamples. However, not all temporal properties admit linear counterexamples, and failure of some cannot be explained by a counterexample at all. We have developed a framework for generation, visualization, and exploration of counterexamples [GC03c, CG05a, CG05b]. The framework is based on a simple observation that counterexamples are simply *proofs* by example. It is comprised of two parts: a notion of *proof-like counterexamples*, and an interactive visualization and exploration engine KEGVis. The advantage of proof-like counterexamples is that they (a) preserve the desired usability aspects of counterexamples, i.e., size and close correspondence to the model; (b) can be used to provide feedback even if the counterexample is not available in general; and (c) help users in *understanding* complex counterexamples. Additional information present in proof-like counterexamples is used by KEGVis to allow the user to control which part of a counterexample is presented, guide counterexample generation towards an "interesting" part of the model using strategies, and skip over "uninteresting" parts. We believe that our framework is flexible enough to enable creation of truly user-friendly tools to facilitate effective model exploration and debugging using model-checking technology.

Current and Future Work

I believe that the future of formal methods research is in generation and application of scalable automated verification technologies. My future work will continue to be aimed at expanding the range of projects where analysis and verification can be applied inexpensively and effectively. I am particularly interested in *automated* reasoning, which is likely to be best achieved by a combination of model-checking and theorem-proving, and its careful application to software systems. My work will remain on the border between formal methods, logic, and software engineering. Below, I list some of my current and future threads in this direction.

Verification of Webservices. During my internship at IBM CAS Toronto, I have become interested in applying automated verification, in particular, model-checking and runtime monitoring, to webservices. Webservices pose a major challenge to both verification and security since they are (a) highly distributed, often executing across several machines, (b) long lived, often executing for days or even years by storing their state in a database, and (c) manipulate sensitive information. As a first step, I plan to explore how the combination of our recent work on symmetry reduction [WGC05] and software model-checking can be used to combat the distributive nature of webservices. I hope that my continuing collaboration with IBM will provide me with access to real-life examples.

Secure Software Systems. With proliferation of the Internet, many companies are facing a problem of ensuring security of their computing infrastructure. Building secure software requires a combination of systems engineering and verification methodologies. The former is necessary to develop security-aware infrastructure that is as scalable as existing security-unaware ones. The latter is needed to ensure that applications use this infrastructure correctly, and are therefore secure. Together with Professors M. Chechik and D. Lie and several graduate students in the security group, I have started to explore the use of Virtual Machine Monitors for such an infrastructure, and the use of our software model-checker YASM for its verification. Our preliminary results on ensuring security of the login phase of the `OpenSSH` implementation are very promising. I look forward to continuing this line of research as well as finding other applications of automated analysis to aid in the construction of secure software systems.

Software Model-Checking. Software model-checking is an exciting new area with a lot of open problems, such as handling dynamic memory allocation, concurrency with (and without) recursion, object-oriented languages such as Java, making the analysis more compositional, dealing with the library functions for which the source code is not available, exploring integration with established quality assurance techniques such as testing, etc. Since the software model-checking problem is undecidable, it is impossible to build a tool that will verify an arbitrary property of an arbitrary program. Instead, I plan to concentrate on application of software model-checking to particular domains, such as software architectures, security, and webservices, and work on improving existing technologies to succeed in these areas.

Environment Guarantees. Current research on vacuity detection, including my own, has reduced vacuity detection to discovering when a part of the *property* is irrelevant. However, our recent experience with the analysis of models developed in a graduate course on Automated Verification at University of Toronto has shown that this does not always point to a problem in the design. To address this, we have started to explore a model-based alternative to vacuity, i.e., detecting whether an important part of the *model* is irrelevant for the analysis [CGG05]. In particular, we have concentrated on a class of systems which are specified as a composition of a machine and its environment. In this case, there is an implicit intention that the analysis of a property cannot depend on the environment alone. We have developed a technique to detect such *environment guarantees* and have applied it to a previously verified model of an aircraft collision avoidance system (TCAS II) [U.]. In our case study, we have found that several properties depended on the environment alone, exposing a problem in the model of the environment, and rendering some results of the verification [CAB+98] meaningless. Although our current approach looks promising, it assumes a rather restricted form of a composition between the machine and its environment. In the future, I plan to continue working on addressing this limitation, and evaluate the effectiveness of our approach on realistic case studies.

References

[BBDER01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". *Formal Methods in System Design*, 18(2):141–163, March 2001.

[Bel77] N.D. Belnap. "A Useful Four-Valued Logic". In Dunn and Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 30–56. Reidel, 1977.

[CAB+98] W. Chan, R.J. Anderson, P. Beame, S. Burns, F. Modugno, and D. Notkin. "Model Checking Large Software Specifications". *IEEE Transactions on Software Engineering*, 24(7):498–520, July 1998.

[CDEG03] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". *ACM Transactions on Software Engineering and Methodology*, 12(4):1–38, October 2003.

[CDG01] M. Chechik, B. Devereux, and A. Gurfinkel. "Model-Checking Infinite State-Space Systems with Fine-Grained Abstractions Using SPIN". In *Proceedings of the 8th SPIN Workshop on Model Checking Software*, volume 2057 of *LNCS*, pages 16–36, Toronto, Canada, May 2001. Springer.

[CDG02] M. Chechik, B. Devereux, and A. Gurfinkel. "χChek: A Multi-Valued Model-Checker". In *Proceedings of 14th International Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 505–509, Copenhagen, Denmark, July 2002. Springer.

[CG03] M. Chechik and A. Gurfinkel. "TLQSolver: A Temporal Logic Query Checker". In *Proceedings of 15th International Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 210–214, Boulder, Colorado, July 2003. Springer.

[CG05a] M. Chechik and A. Gurfinkel. "A Framework for Counterexample Generation and Exploration". In *Proceedings of Fundamental Approaches to Software Engineering (FASE'05)*, volume 3442 of *LNCS*, pages 217–233, Edinburgh, Scotland, April 2005. Springer.

[CG05b] M. Chechik and A. Gurfinkel. "A Framework for Counterexample Generation and Exploration". *Submitted to International Journal on Software Tools for Technology Transfer*, September 2005. 16 pages.

[CGD+02] M. Chechik, A. Gurfinkel, B. Devereux, A. Lai, and S. Easterbrook. "Symbolic Data Structures for Multi-Valued Model-Checking". CSRG Tech Report 446, University of Toronto, January 2002. 49 pages, submitted for publication.

[CGG05] M. Chechik, M. Gheorghiu, and A. Gurfinkel. "Efficient Debugging of Environment Models". Submitted for publication, September 2005.

[Cha00] W. Chan. "Temporal-Logic Queries". In *Proceedings of the 12th Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 450–463, Chicago, IL, July 2000. Springer.

[Fit02] M. Fitting. "Bilattices are Nice Things". In *Proceedings of Conference on Self-Reference*, Copenhagen, Denmark, 2002. `http://comet.lehman.cuny.edu/fitting/bookspapers/pdf/papers/BilatticesPhiLog.pdf`.

[GC03a] A. Gurfinkel and M. Chechik. "Generating Counterexamples for Multi-Valued Model-Checking". In *Proceedings of Formal Methods Europe (FME'03)*, volume 2805 of *LNCS*, pages 503–521, Pisa, Italy, September 2003. Springer.

[GC03b] A. Gurfinkel and M. Chechik. "Multi-Valued Model-Checking via Classical Model-Checking". In *Proceedings of 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, pages 263–277, Marseille, France, September 2003. Springer.

[GC03c] A. Gurfinkel and M. Chechik. "Proof-like Counterexamples". In *Proceedings of 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 160–175, Warsaw, Poland, April 2003. Springer.

[GC04a] A. Gurfinkel and M. Chechik. "Extending Extended Vacuity". In *Proceedings of 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *LNCS*, pages 306–321, Austin, Texas, November 2004. Springer.

[GC04b] A. Gurfinkel and M. Chechik. "How Vacuous Is Vacuous?". In *Proceedings of 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *LNCS*, pages 451–466, Barcelona, Spain, March 2004. Springer.

[GC06] A. Gurfinkel and M. Chechik. "Why Waste a Perfectly Good Abstraction?". In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, Vienna, Austria, March 2006. (To appear).

[GCD03] A. Gurfinkel, M. Chechik, and B. Devereux. "Temporal Logic Query Checking: A Tool for Model Exploration". *IEEE Transactions on Software Engineering*, 29(10):898–914, October 2003.

[GDC02] A. Gurfinkel, B. Devereux, and M. Chechik. "Model Exploration with Temporal Logic Query Checking". In *Proceedings of SIGSOFT Conference on Foundations of Software Engineering (FSE'02)*, pages 139–148, Charleston, SC, November 2002. ACM Press.

[GGC05] M. Gheorghiu, A. Gurfinkel, and M. Chechik. "VaqUoT: A Tool for Vacuity Detection". CSRG Technical Report, University of Toronto, April 2005.

[Gin88] M. L. Ginsberg. "Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence". *Computational Intelligence*, 4(3):265–316, 1988.

[Gur02] A. Gurfinkel. "Multi-Valued Symbolic Model-Checking: Fairness, Counter-Examples, Running Time". Master's thesis, University of Toronto, Department of Computer Science, October 2002.

[GWC06] A. Gurfinkel, O. Wei, and M. Chechik. "Systematic Construction of Abstractions for Model-Checking". In *Proceedings of 7th International Conference on Verification, Model-Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *LNCS*, pages 381–397, Charleston, SC, January 2006. Springer.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.

[U.] U. S. Department of Transportation. "Introduction to TCAS II". Federal Aviation Administration.

[WGC05] O. Wei, A. Gurfinkel, and M. Chechik. "Identification and Counter Abstraction for Full Virtual Symmetry". In *Proceedings of 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *LNCS*, pages 285–300, Saarbrücken, Germany, October 2005. Springer.