Formal Methods:
Future Directions & Transition To Practice
Workshop Report

Ranjit Jhala[1], Rupak Majumdar[2], Rajeev Alur[3], Anupam Datta[4], Daniel Jackson[5], Shriram Krishnamurthi[6], John Regehr[7], Natarajan Shankar[8], and Cesare Tinelli[9]

[1] UC San Diego
[2] Max-Planck Institute For Software Systems
[3] University of Pennsylvania
[4] Carnegie Mellon University
[5] MIT
[6] Brown University
[7] University of Utah
[8] SRI International
[9] University of Iowa

1 Introduction

Recent years have seen many success stories where formal methods, or ideas influenced by formal methods, have transitioned successfully from the research lab to development tools. Yet, formal methods have remained the broccoli of the computing world: nutritious but not palatable, or, at any rate, an acquired taste.
In domains where there is a catastrophic cost of failure (e.g., in hardware design, in safety critical embedded applications, or in security-critical domains), formal method techniques have become an indispensable part of the design and implementation process.
But in the broader computing world, the potential for formal methods in the development process is not widely understood, and formal techniques are often treated with skepticism or dismissal. Thus, in December 2012 the National Science Foundation sponsored a workshop that brought together formal methods researchers and practitioners, from academia, industrial research laboratories, government institutions and tool vendors, to take stock of the situtation, identify successes, the obstacles towards broader adoption, and to chart future directions for formal methods and its effective transition to practice.

1.1 What have we Achieved?

We started the workshop off by polling the participants about what they perceived to be the successes of the field, which could be used as a lens to examine how the formal methods landscape had changed over the course of the last decade and half. The responses can be grouped into several categories:

- **Specification, Design & Modeling** These include high-level modelling tools like StateCharts and Alloy which can be used to analyze and refine *designs*, domain-specific tools such as Simulink, Scade, and Stateflow in embedded systems, and others. Furthermore, the last decade saw a great deal of progress on standardizing extensible specification languages, such as JML, which has streamlined the development of various formal tools for *implementations* as well.
- **Language Technologies** Perhaps the most pervasive formal methods are those that are not even viewed as formal methods anymore! These include the pervasive use of static type systems in mainstream languages like Java and C#, the use of synthesis tools like parser generators, the reliance on aggresive compiler transformations for program optimization, and liberation from manual memory management through garbage collection.
- **Hardware Analysis** The last decade and half saw a huge adoption of methods in the design and implementation of hardware. These include VLSI synthesis and equivalence checking. It is routine for large EDA vendors like Cadence, Synopsys and Mentor Graphics to ship tools that use model checking for assertion-based verification. Companies like Intel and AMD have in-house teams that use interactive theorem provers like HOL or ACL2 to formally verify the correctness of various logic blocks.
- **Software Analysis** Perhaps the least anticipated successes came in the realm of software, where remarkably effective tools based on model checking (such as SPIN), abstract interpretation (SLAM/SDV and ASTREE), or dataflow analysis (e.g. Coverity, KlocWork, Fortify) made static analysis mainstream.
- **Proof Assistants** While interactive proof assistants like ACL2, PVS, Isabelle, and HOL have been in development since the seventies and eighties, and adopted by hardware vendors like AMD and Intel, the last few years have seen a large upsurge of interest in the systems and languages communities thanks to several landmark projects such as the seL4 formally verifed microkernel, and the CompCERT compiler.
- **Formal Testing** In a related vein, the last decade saw the adoption of a variety of formal approaches to software testing. Two prominent strands of work include the development of *property based testing* pioneered by Quickcheck and now adopted across many modern languages, and the widespread use of symbolic execution for automatic testing in software and hardware.
- **Core Technologies** Finally, many of the above successes rely crucially upon advances made in core algorithmic techniques such as improvements with Binary Decision Diagrams (BDDs), propositional satisfiability (SAT) solvers, and satisfiability solvers for first-order theories (SMT solvers). Decades of sustained research on these problems came to fruition as we saw a revolution in the scalability of these tools, enabling their widespread use as generic constraint solving mechanisms in domains from dependency management to biology.

1.2 What has Changed? What is Changing?

Thus, as is apparent from the litany of successes above, there has been a silent revolution in formal methods since the mid-1990s, as the stakes for reliable software and systems have grown vastly higher.

- **Application Areas** Our news media remind us daily of security concerns about our infrastructure and the privacy concerns of individuals. The growing use of medical devices with significant computational features demands greater attention. The pervasive embedding of computing devices in all aspects of our lives, from smartphones to smart meters to self-driving cars, means these concerns will only grow. The deep reliance of our financial system on software adds to these pressures; and recently, attention has been drawn to the risks of bad computation adversely affecting policy decisions.
- **Tools** The widespread adoption of safe languages makes it more likely that tools will be able soundly to establish the absence of certain classes of errors. The widespread popularity of non-trivial type systems in mainstream languages like Java and C# further strengthens this power. Moreover, we are now seeing the advent of a new generation of tools to help programmers—from partial verifiers to checkers for rich type systems—released by academic groups, industrial labs, and sometimes even commercial development groups. Finally, popular software engineering techniques like model-driven programming have the potential to be large-scale enablers of more advanced formal methods.
- **Resources** A lack of resources of diverse sorts—in computing power, in training materials, in insights into the needs of users—has hindered the adoption of formal methods. In all these areas, change is appearing. Cloud services provide enormous computing power; large quantities of data, such as open-source program repositories, make it easier to study the patterns of errors programmers make and identify effective interventions; the Web enables large-scale dissemination (and sharing) of materials such as books and courses, as well as making it possible for expertise to be shared easily.
- **Attitudes** Major corporations, which influence views about dependability and security in the public's mind, are paying more attention to security and safety risks; and the posture of the companies that are addressing security issues most aggressively will help change perceptions about the need for better development techniques, and for certification. At the same time, the formal methods community has made a conscious effort to reach out to practitioners. In recent years, the formal methods community has focused on providing usable tools to programmers which, while they fall short of the ideal of complete functional correctness proofs, are nevertheless very useful in improving programmer productivity.

1.3 What did we find?

In short, the pervasiveness and growing centrality of computing makes this a challenging and exciting time for formal methods. The goal of the workshop was to reflect upon the most effective ways for the formal methods community to influence the computing revolution. To this end, we structured the workshop along four (related) focus areas.

- **Infrastructure & Community Synergies** As the field develops, it is increasingly harder for researchers to make progress working in isolation. Building large scale verification tools requires a lot of effort in developing scalable infrastructure. Often, much of the effort is duplicated across projects. The goal of this area was to identify community synergies, for example, in defining standard intermediate formats.
- **Education & Outreach** A major stumbling block to the adoption of formal methods is often lack of expertise in the workforce. If formal methods are to be broadly adopted, they must be accessible to all engineers, not only specially trained Ph.D.'s. The purpose of this direction is to recommend education and outreach activities that will lower the bar to the adoption of new techniques, as well as inform tool developers about the usability requirements of developers and quality assurance engineers.
- **Roadmap to Adoption** The goal of this area was to identify effective ways of transitioning research tools to industrial practice. Adoption requires better scalability and effectiveness of the tools, demonstrating effectiveness, and better usability of the tools.
- **Frontiers & New Challenge Domains** The rapid spread of computing machinery through a variety of critical domains has necessitated the development of specialized techniques that ensure the reliability and correctness of the software used in those domains.
  The purpose of this focus area was to identify the new domains where rigorous guarantees are required and which are ripe for current- or next-generation formal methods and tools.

This report, we summarize the discussions from the four areas, in each case, identifying successes, obstacles and promising thrusts for future work. With strong, co-ordinated and strategically chosen pushes we can significantly influence all these trends for the better, allowing society to continue to reap benefits from computing technology, while helping to build a foundation for a safer, more secure and more dependable future.

2 Recommendations

2.1 Infrastructure & Communities

While formal methods technology has advanced significantly in recent years, the fruits of these advances are not yet readily available to the average software or hardware engineer. First, we outline the role of community and shared infrastructure in consolidating and extending the success of formal methods.

Obstacles

- Robust and Usable Infrastructure Developing robust and usable infrastructure for formal methods is a difficult task, but it is often a prerequisite for impactful research.
- The absence of standard formats and benchmarks (e.g. for SAT and SMT problems) across the board, has been an obstacle in large-scale infrastructure development and cross-group and cross-disciplinary collaboration.
- Without an adequate body of user manuals, tutorials, and technical documentation, it is hard for people outside the formal methods community to identify if a specific method is a fit for their problem, and to invest in deploying and learning a tool in the context of a real project.

Recommendations

- Follow the example of efforts like SMTlib and JML and develop standard formats for various formal technologies. Use these formats to develop and maintain public benchmark and problem suites.
- Create large-scale shared hardware infrastructure for experimental evaluation of formal methods technologies, e.g. deployed via community portals such as SATLive, StarExec, and CPS-VO. Correspondingly, build shared suites of software utilities that can be used in developing formal toolchains.
- Build research communities around specific formal methods and problems, that can motivate good research and in both broadening and deepening the impact of the results and can also help create a cadre of early adopters, people who are willing to apply formal techniques in the development of new hardware and software designs.
- Develop incentive and reward mechanisms in academia that are directly aligned with promoting community-oriented research activity, and which reward high-quality experimental work and infrastructure development.

2.2 Education & Outreach

Ultimately, the adoption of formal methods will depend crucially on our ability to educate software and hardware professionals in the importance and application of formal tools, techniques, and thinking.

Obstacles

- Dearth of suitable material to support teaching of formal methods content by non-specialists.
- Academics in other fields tend to have an antiquated view of formal methods, leading to poor decisions about curricula and student advising.
- The success of bug-hunting and post-facto verification gets in the way of thinking clearly about more subjective, and hard-to-define issues like *design*, which may be even more crucial.

Recommendations

- Develop educational materials on *Big Ideas In Formal Methods* which incorporate of the key ideas that typify formal methods thinking, and illustrate some of the breadth and depth of the field.
- Develop classes that teach Formal Methods *In Context*, e.g. within project-driven classes on embedded systems or security, hardware design, cryptography, or distributed systems.
- Develop *online classes* (so-called "MOOCs") and modules that can be shared across universities with diverse expertise, can reach out to motivated practitioners, and eventually encourage employers to mandate such classes as required certification prior to hiring.

2.3 Adoption

In order to transition formal methods into widespread practice, we must identify the key technical problems that must be addressed to develop usable methodologies and tools, but perhaps more importantly, determine the dissemination, incentive and reward mechanisms that will facilitate adoption among students and industry practitioners.

Obstacles

- A formal tool will only be used by practitioners if it scales to large, real-world designs and code bases, and if it is seamlessly integrated into the development workflow. One significant challenge to building such tools is that they are fashioned from many small(er) but sophisticated pieces. The semantic gap between these pieces, e.g. input and output formats or different assumptions about the environment, is a significant obstacle to building widely usable tools.
- There are often large *cognitive gaps* between formal tools and their human users. The user must possess a clear understanding of how to express the properties, and have a reasonable understanding of how the tool works, at the very least, to know what knowledge about the system, and its interfaces are required by the tool and where the tool may lose information, and to interpret the results returned by the tool to fix errors or modify the design or implementation.
- Adoption is a *social* and *psychological* process. For some users (and use-cases and tools), formal methods are perceived as tedious while others view them as addictive like "video games". Similarly, broader adoption requires developing *relationships* with particular communities of potential users. The formal methods community has not studied the question of *understanding users* and *use cases*.
- Finally, there is a lack of *metrics* and data on the benefits offered by formal approaches, which in turn, is a major obstacle towards persuading engineers and managers that development processes should be shifted to accomadate formal methods.

Recommendations

- To address the problems of scalability and integration with the development workflow would be to devise new theories and tools targeted towards *incremental verification*. New artifacts – hardware or software – are not developed from scratch, they are typically modifications that build upon and retain much of the behavior of older versions. Thus, the key research question is, suppose we have already paid a high cost – compute power or human expertise – to analyze an original version, say from one year ago, can we quickly and cheaply analyze the changes to verify the version from one week or one hour ago?
- Some tool interoperability and composability problems may be addressed by the development of suitable standards, which would allay the fears some commercial users have of being "locked into" particular tools or vendors. However, other problems, e.g. connecting different kinds of analyses – like alias- and value- abstractions – may require *new abstractions* that facilitate interoperability.
- Mechanisms like gamification and competitions can be used as incentives for long term infrastructure development, and dually, to help users familiarize themselves with and gain expertise with formal methods and tools. Competitions have proven to be extremely effective catalyst for infrastructure and community building in particular domains (e.g. SAT solvers) – by providing a stimulus for standardizing formats, evaluating tools, and providing an incentive for continuously improving the tools.

- *Web-based tools* would establish a two-way communication with users. Users would have access to a variety of different tools, and researchers would be able to get insight into the kinds of problems the tools were being used for, precise usage logs, and particular usability concerns that may not be apparent *a priori*.
- Formal methods researchers should build bridges with experts in HCI and Psychology & HCI to properly understand the target audience for formal methods: the human developer. A clear understanding of developers' mental processes is key to identifying the shortcomings and limitations of informal methods during the design, implementation and validation of computing systems, which can then be profitably complemented by formal approaches. HCI experts, psychologists and sociologists have started to devise creative ways to use the web to dramatically increase the scale and fidelity of their experiments, and hence, could prove to be invaluable collaborators in the design and evaluation of various gamification approaches.
  Funding agencies like the NSF could encourage the building of these bridges. Future proposals can look at previous work for guidance in how to perform these experiments

2.4 New Frontiers

What is next for formal methods? Does it have the potential to transform existing domains and to catalyze new domains? For long-term viability of the field, it is important to take a step back and identify problem domains where formal methods can play a crucial role.

Foundations

- *Constraint-based Analyses and Solvers* e.g. SAT/SMT solvers, have been a huge success story for formal methods. Research in core constraint-solving technologies (more expressive logics, domain-specific theories) and their interfaces (e.g., direct support and optimizations for specific domains) continue to be important, and solvers are increasingly being integrated into programming languages and IDEs, giving the programmer or even the end-user the power to declaratively state their requirements and let the system figure out the details.
- *Synthesis* Improved verification technology allows detection of subtle bugs, but this only means a costly reiteration of the design and programming phases, and not necessarily a quality improvement. A plausible alternate that has emerged as a vibrant area of research is *synthesis*, which starts with a higher level specification of the desired behavior and produces implementations which are correct by construction.
- *Design space exploration* An ambitious goal is to use modeling and analysis for both correctness and performance optimization. There is a lot of theoretical work incorporating quantitative models into verification — a shift from "model checking" to "model measuring" and "model optimization" — we expect to see the development of efficient analyses and integration into the system-design cycle.

Inroads

- *Security* The importance of developing scientific foundations for security has received wide recognition in government, academia, and industry. These initiatives recognize that foundational advances in security science is needed to develop systems that can resist attacks by sophisticated adversaries. Formal methods have an essential role to play in the development of this science.
- *Systems Biology* Computational modeling and analysis of biological processes is increasingly common. Computational models such as state machines, hybrid automata, Petri nets, and stochastic transition systems have been applied to modeling biological processes, leading to new biological insights and core algorithmic problems in formal methods, for example, the algorithmic analysis of stochastic systems, notions of metrics on systems, and the use of statistical techniques in model checking.
- *Safety-critical Cyber-physical Systems* like controllers in automotive, medical, and avionic systems, comprise many software modules reacting to a continuously evolving environment. The cost-effective, but high assurance, development of cyber-physical systems will be an increasingly important challenge for formal methods.

– *High-performance computing* is often regarded as "the third pillar of science" and is central to new scientific discoveries and engineering solutions. Formal methods are essential to manage the intellectual complexity of large-scale designs, to develop focused testing methods to find defects early, and to ensure that long-lived APIs are well described to ensure uniformly understandable descriptions and portable implementations.

Targets of Opportunity

– *Clean slate system design* Traditionally, the role of formal methods have been to prove or disprove properties of system components (such as device drivers, communication protocols). The maturing of formal tools has resulted in recent years to functional verification of full-fledged applications, like Sel4 and CompCert. Success of these projects leads us to the next challenge for formal verification: can we design a complete system (ideally all the way from applications such as a web-browser, through language-implementation tools such as a compiler and an operating sys- tem, down to the hardware execution platform) with an accompanying proof of correctness?
– *End-user Programming* More people have access to programmable devices than ever before. Yet, programming remains the job of an expert. Can formal tools and techniques help novice program- mers to construct systems? Can novice programmers design working programs in an interactive environment by presenting example behaviors?
– *Cloud Computing* Recent years have seen the phenomenon of computing as a service, where companies provide computing and storage infrastructure that can be shared across many different users. In addition to the obvious verification questions for cloud programs – e.g. verifying distributed implementations, ensuring integrity and confidentiality of executions, etc. – there is an opportunity to introduce new programming models that deal with dynamic provisioning, distribution, and failure handling. Finally, the formal methods community can engage in software systems development by developing and maintaining precise specifications, models, and standards for widely used systems components.
– *Privacy and Trust* have become a significant concern in modern society as personal information about individuals is increasingly collected, used, and shared, often using digital technologies, by a wide range of organizations. There is a strong need for systematic development of foundations of privacy, including formalizing privacy concepts and developing methods and tools for enforcing privacy properties. In general, Formal methods provide the tools to clearly state and analyze policies, and to design enforcement and audit mechanisms, and hence can also be an effective tool in broader matters of public policy.

3 Conclusion

This report summarized the discussions from the four areas, in each case, identifying successes, obstacles and promising thrusts for future work. With strong, co-ordinated and strategically chosen pushes we can significantly influence all these trends for the better, allowing society to continue to reap benefits from computing technology, while helping to build a foundation for a safer, more secure and more dependable future.