Applications of Formal Methods

A Review of Literatures on Formal Methods

John Hornik
Assignment 4
Research in Computer Science
Lewis University
Romeoville, IL, USA
johnphornik@lewisu.edu

***Abstract*—This paper reviews concepts presented in *Formal Methods: Practice and Experience* [1] and *Survey of Existing Tools for Formal Verification* [2]. It begins by giving summaries of [1] and [2], with almost all of the concepts and data taken directly from [1] and [2]. It then presents a focused section on the Verified Software Repository. This section also presents a summary of [3].**

Keywords—formal methods; formal verification; software development; Verified Software Repository

I. Introduction

This paper is a review of formal methods within the realm of software development. It summarizes the concepts and ideas covered in three different literatures on the topic. The papers main focus is to present summaries of these literatures as well as cover the importance of a Verified Software Repository. Formal methods are extremely important due to how much people rely on software on a daily basis. With the implementation of these methods, humanity could potentially be led to a world were software is 100% reliable and allows us to achieve things that we have never before thought possible.

II. Review of Formal Methods: Practice and Experience

A. Introduction

[1] is an assessment of the most recent advancements in formal methods. It begins by covering a number of surveys on the topics of formal methods as well as verification technology and how they are used in industry. The results of a new survey are then presented on industrial practice and how it has changed over the last 20 years. The paper then goes on to describe industrial projects spanning the last 20 years. The findings of the writers are then presented regarding the current state of the art technologies. After this, the paper covers the Verified Software Repository that is being designed and built in response to a lack of technical and cost-benefit evidence from current formal methods and verification technologies. The paper concludes with some finishing statements regarding current practice and experience.

B. Surveys of Formal Methods Practice

Many different surveys from the 1990's are presented as a means to introduce the state of formal methods within industrial development processes of that time period. Many of the authors of these surveys used tools such as questionnaires as well as literature reviews and interviews in order to obtain more information on the applications. The findings conclude that the surveys take different viewpoints throughout and don't focus on just one response. Some of them focus on a group of applications while others have a wide range of industry academia covered. One thing that is certain is the challenges that are present with applying these methods to industrial applications.

C. A Survey of Current Use and Trends

The authors also undertook a survey of their own in order to gain more information from numerous industrial projects. They did this to further understand trends as well as advances to address some of the challenges presented in the previously covered surveys. The survey was a structured questionnaire which was administered from November 2007 through December 2008 for a combined total of 62 different industrial projects. The application areas which the survey covered were (in decreasing order) transport, financial, defense, telecom, office/administration, nuclear and consumer electronics, and healthcare. Others were also covered, however, their responses were far fewer than the preceding application areas. There were many application types covered within these projects surveyed. The largest were real-time applications, distributed applications, transaction processing, and high data volume applications. The number of projects by year and the size of projects in lines of code (LOC) are also presented. The most projects were started around 2004 through 2007, with 2006 having the most projects at ten. There were 13 projects with 10-100 kLOC, 10 projects with 1-10 kLOC, and nine projects with 100-1000 kLOC. As far as numbers of projects using each technique go, specification/ modeling had the most followed by execution, inspection, model checking, refinement, proofs, and test generation.

The outcomes of the formal techniques showed some interesting points. As far as time goes, overall, 53% reported that there was no effect/no data, 35% reported an improvement, and 12% reported a worsening. With cost, 56% reported that there was no effect/no data, 37% reported an improvement, and 7% reported a worsening. Finally, with effects on quality, 92% reported an improvement and 8% reported no effect/no data. There were no negative reviews for the quality aspect. The final surveys administered measured the respondents' feelings towards the surveys. 61% strongly agreed and 34% agreed that the use of formal methods was successful. There were no strong disagreements in this category as the rest had mixed opinions. 49% agreed and 46% strongly agreed the techniques were appropriate for the tasks required. Again, the other 5% had mixed feelings. Only 9% disagreed with the tools being able to cope with the tasks undertaken while 56% agreed and 28% strongly agreed. In the end, 75% agreed that they would be implementing similar methodologies in the future, and the other 25% believed that they would possibly be implementing similar methodologies in the future. It was also mentioned that those who gave an opinion on the matter reported that more projects had reduced timescales, costs, and improved quality by implementing formal techniques.

D. *Highlighted Projects*

After the surveys, several projects were highlighted within the paper. The first is a Transputer project which discusses a series of microprocessor chips which were designed for parallel processing applications. These chips required months of testing due to their complex floating-point units. Each month delay was reported to cost the company one million dollars. Eventually, a formal development of a correct-by-construction floating-point unit was developed and this resulted in development time of approximately three months faster than its predecessor. The next project presented was the Mondex Smart Card. This was a smartcard-based electronic cash system meant for low-value cash transactions. Being a monetary system, security was essential to its success. A company called Logica took on the challenge. They implemented Z for the proof and specification. In the end, there were no errors found in the system due to the use of formal methods. Additionally, the cost of mechanizing the Z proofs were 10% of the original development cost which proved that they weren't as expensive as initially believed. After this project, one on AAMP Microprocessors was covered. In this project, the AAMP5, a proprietary microprocessor used by Rockwell Collins, was formally verified using PVS. The microprocessor contained 500,000 transistors and the company specified it at the both the register transfer and instruction set levels. This project proved that it was technically possible to prove correctness of microcode and that formal specifications could be read and written. The next project covered Airbus and its implementation of SCADE. The benefits include a decrease in coding errors, shorter requirements changes, and major productivity improvement. Having achieved all of these benefits from SCADE, Airbus adopted it for many future projects. After this, the Maelslant Kering, a movable barrier protecting the port of Rotterdam, was discussed. This barrier operates (deploys and reopens) based on data acquired by a computer. It was decided that modeling and verification technology would be implemented to test the system. A number of problems arose in which 85% arose during development and 15% arose during reliability and acceptance testing. After their fixes, no defects have been reported due to the utilization of formal techniques. The Tokeneer Secure Entry System was another project discussed. This relates to access control in the form of a system that carries out biometric tests of a user and, in turn, decides whether or not to open an entrance for that user. Implementing formal methods, the company Praxis found two separate bugs. These are bugs that were not caught during the original project. Finally, the last project covered was the Mobile FeliCa IC chip firmware. For the testing of these IC's, Sony (the manufacturer) used VDM++ and were able to find 440 defects with this method as a result.

E. *Observations*

The authors then go into pointing out some of their observations. These include concerns raised in the surveys as well as progress, trends, and remaining challenges of recent projects. Lightweight and Heavyweight Formal Methods, Tool Support, Increasing Automation, and Cost Effectiveness are all discussed in depth.

F. *The Verified Software Repository*

In this section, the concept of building a Verified Software Repository is introduced. The repository would contain hundreds of programs with full or partial specifications, designs, test cases, assertions, history, and documentation. The end goal would be to have an immense amount of verified software in one place so that it can present less errors and be more trustworthy. Verified File Store, FreeRTOS, Radio Spectrum Auctions, Cardiac Pacemaker, and Hypervisor are all applications which are thoroughly covered. The file store being inspired by space applications and the FreeRTOS was inspired by real-time embedded systems. The Radio Spectrum Auctions deals with real bidding applications. The pacemaker relates to existing medical devices while the Hypervisor relates to a planned product by Microsoft.

III. Review of Survey of Existing Tools for Formal Verification

A. *Introduction*

[2] outlines the capabilities of commercial and open source formal tools as well as the ways in which they can be implemented in digital design workflows. It begins with an introduction to formal methods and tools. It then covers some tools for checking abstract models followed by a section for tools used for checking hardware description languages (HDLs). Following this, there are sections on tools for checking the correctness of software and tools to create a provably correct design. It concludes with a summary of topics covered.

B. *Introduction to Formal Methods and Tools*

In the introduction, model checkers are defined to check the design with respect to the specified properties encoded in a modeling language while theorem provers combine automated techniques with manual guidance to prove correctness. The most notable difference between the two is that model checkers can be used by inexperienced developers that need a go-to tool, however, they may be limited in their capabilities. Theorem provers, on the other hand, are more hands-on but require expertise. Formal tools can further be classified into two classes: tools which verify correctness of a model and tools which create models or designs that are correct by construction. The former of these have properties such as having well defined semantics, having concise languages while lacking elaborate design details, and having features unique to verification. The latter tools provide a formal framework and methodology to model and prove properties of a system.

C. *Tools for Checking Abstract Models*

In this section, a number of tools for reasoning about high-level properties of a system are covered. The first is Spin which is a model checker targeted for the formal verification of software algorithms. It is well suited for detailing properties of small models and high-level properties of large systems. The next tool is Uppaal. This is a tool for model-checking real-time systems. NuSMV (and its predecessor, SMV) are software tools for the formal verification of temporal properties of finite state systems. Failure-Divergences Refinement (FDR) is a tool to check models expressed in the algebra of Communicating Sequential Processes (CSP), a process calculus for concurrent systems. Alloy is an object and structure modeling language which is based on set theory. Finally, Simulink Design Verifier is a tool for the verification of the formal properties of a Simulink design.

D. *Tools for Checking Hardware Description Languages*

Tools for checking HDLs primarily employ model checking. They generally let the user know property failure, property pass, or indeterminate when analyzing a system. These tools are mainly commercial which means that costs are high. Questa Formal is one of these tools. It is a formal verification tool which is easy to learn and works well with other products developed by its manufacturer, Mentor Graphics. It does, however, lack certain features such as abstraction of counters and advanced liveness features. Solidify is another tool discussed although it is mentioned that it also lacks abstraction of counters and advanced liveness features. It was also noted to contain several bugs by the authors of this paper. It is meant to support VHDL and some parts of SystemVerilog. JasperGold is covered as one of the more capable tools tested by the authors. This tool contains automatic extraction of counters for abstraction as well as manual counter extraction. The last tool is Incisive, a formal verification tool offered by Cadence, which seems to work well from the authors viewpoints.

E. *Tools for Checking the Correctness of Software*

In this section, a limited set of tools are discussed which can be used to do a formal analysis of the source code of a software system. The first is Frama-C which is meant for the static analysis of C source code. It enables the user to specify complex functional specifications and to prove correctness based on these specifications. BLAST is an automatic tool for verification of the temporal properties of C programs. It checks reachability, standard C logic, and temporal properties. Java Pathfinder is a tool to check Java byte code. It has proven useful for analyzing concurrency as well as abstract models of programs. Spark ADA is utilized with the ADA programming language and can specify properties by pre and post conditions to sub programs, loop invariants, and dataflow relationships. Malpas is a static function tool which can also check whether or not a program meets its specification. Specifications for Malpas are provided through pre and post conditions, invariants, and assertions.

F. *Tools to Create a Provably Correct Design*

Software systems that are designed for correctness require tools which allow the developer to implement formal techniques in the initial stages of the design and then maintain these properties throughout the design. VDM is a toolset used for modeling computer programs and is rooted in a common formal specification language. Event-B is a language based on set theory and it describes abstract state machines. Z and B are also based on set theory and are used for the specification of set theory. Rodin is utilized in the design and analysis of Event-B models.

IV. Review of The Verified Software Repository

As mentioned in [1], the Verified Software Repository is a concept to have a repository with a number of different of different programs which could help developers in verifying the validity of their own code. This has been chosen as an area of focus for this paper due to the benefits that a complete repository would bring to the world of software development. In [3], this very topic is discussed. The paper begins with an introduction and background before diving into the Verified Software Repository and Current Plans.

A. *Introduction*

A repository in the scientific sense is a collection of data which accumulates from experiments and observations and is tied in together with an evolving library of programs which process the data. The paper displays a solid vision for the future of the repository. “The long-term goal of the repository is to assemble, develop, and integrate a software engineering tool-set for use in the construction, deployment, and continuous evolution of dependable computer systems, and to prove the capability of these tools against a representative portfolio of real computer applications [3]”. With the implementation of such a repository, it is noted that there would be a significant reduction in costs of bugs which present themselves during design, development, testing, and all other phases of software development cycle.

B. *Background*

Grand Challenges are introduced as a means for funding computing research which would eventually lead to the idea of a Verified Software Repository (VSR). The Verifying Compiler, a tool-set also discussed in [1], would be capable of automatically proving that a given program satisfies all of its specifications before it can run. A proposed first step is given to develop a Verifying Compiler – to first develop a scientific repository within software engineering. From there, as the maturity of the repository grows, as will the individuals or groups which develop it. In the end, this can eventually spread to a point where companies turn it mainstream and it becomes a staple in the software development cycle.

C. *The Verified Software Repository*

The VSR would aid in the development of software by facilitating access to a collection of analysis tools and challenge codes in order to exercise these tools. The tools that would be implemented include SLAM, Alloy, SPIN, HOL, and SPARK Ada among many others. Essentially, this project would extend the range of formalization and collaborate with others researching the problem. Implementing these tools and concepts would be more beneficial than simpler mathematical modeling and formalization.

D. *Current Plans*

After the launch of the community repository project discussed previously, the authors of [3] list some key outcomes:

- Development of code from specifications
- Development of assertions from code
- Development of assertions from tests
- Verification condition generation
- Theorem Proving

After these initial outcomes, it is hoped that experiments will apply these tools to case studies and code challenges eventually leading to a complete VSR.

V. CONCLUSION

As discussed in [1], verification technology and formal methods have not yet been adopted on a wide scale, despite of their success. Applications of verification and formal methods have shown benefits in industrial settings and many hardware developers continuously apply these concepts to their processes. Ideally, if more industries applied these methods to their software development methods, software would be the most reliable portion of any system. This, joined with the completion of a full Verified Software Repository, would certainly aid in future development of flawless software systems.

REFERENCES

[1] J. Woodcock, P. Gorm Larsen, J. Bicarregui, J. Fitzgerald, “Formal Methods: Practice and Experience”, ACM Computing Surveys.

[2] R. Armstrong, R. Punnoose, M. Wong, J. Mayo, “Survey of Existing Tools for Formal Verification”, Sandia National Laboratories, 2014.

[3] J. C. Bicarregui, C. A. R. Hoare, J. C. P. Woodcock, “The Verified Software Repository: a step towards the verifying compiler”, Formal Aspects of Computing.