Existing Tools for Formal Verification and Formal Methods

Research · February 2020

DOI: 10.13140/RG.2.2.12162.22721

1 author:

Wambura Wasira
Lewis University
4 PUBLICATIONS **5** CITATIONS

SEE PROFILE

Summary of Survey of Existing Tools for Formal Verification & Formal Methods: Practice and Experience

W. Wasira., *MS Computer Science Student, Lewis University*

***Abstract*— This paper is a summary of the themes presented in two papers that report on the use of formal methods in the software development industries. It will also present one industrial application – Airbus A340 and A380 flight control system that has incorporated formal methods in its program development cycle. These papers also describe how relevant these formal methods are in conducting software verification and validation when incorporated in the development of software. The final section of this paper will present a summary of a research conducted on formal method SCADE for verifying Airbus's A340 and A380 flight control systems. The paper *"Survey of Existing Tools for Formal Verification"* presents a report of a survey of formal verification tools developed and adopted for verification of software in the respective domains. *The Formal Methods: Practice and Experience* reports findings of a survey of the adaptation of formal verification methods in software development industries.**

***Index Terms*— Formal Verification, Formal Methods, SCADE, Airbus**

I. INTRODUCTION

II. SURVEY OF EXISTING TOOLS FOR FORMAL VERIFICATION

The paper titled *Survey of Existing tools for Formal Verification* presents the findings of a survey conducted in assessing the capabilities of commercial as well as open source formal verification tools and the way in which they can be useful in digital design workflow. The paper first presents an introduction of Formal methods and tools followed by a description of tools for checking abstraction models, hardware description language, correctness of software as well as tools for creating provably correct software designs. The authors add, what systems require are suitable automated mathematical approaches that are able to exhaustively check the state space of the system without actually running the code or perform analytics of a program.

The first part of this paper briefly outlines the limitations of Testing and Simulation explaining that testing and simulation alone provides a system with a variety of input conditions to confirm that it produces the output as it is required to do so; adding that testing and simulation is capable of sampling functional properties but limited in verifying safety and security properties of software.

The paper also introduces formal methods, categorizing them in two separate categories: **Model checkers** that check the design with respect to the specified properties and return results on whether the model satisfied the specified properties; whether the model has not satisfied the specified properties and if the properties were not satisfied, it produces a counter example that would satisfy the specification of the program. Model checkers also produce indeterminate output in which the checker displays areas of the state space where cannot compute results in a reasonable amount of time. The second category outlined are **Theorem provers**, more powerful than model checkers, they combine automated techniques with manual proof to prove correctness of a program's algorithm.

Like computer languages, formal methods are also designed for use at different levels of abstractions. This knowledge has given rise to high-level formal tools that are used to prove specification and low-level tools that are used for reachability and program correctness. The report classified available tools based on the tools' capabilities and usage:

1) Tools for Verifying Correctness of a Model:

Tools used for verifying correctness of abstract models do not assist in the creation of a model, instead they can be used to test out ideas prior to implementation. The tools that are used for verifying abstract models are often language specific, so models must be create using the specific language of these tools. These tools include Spin, Uppaal, SMV, NuSMV, FDR, Alloy, and the Simulink Design Verifier.

Tools for verifying actual design description. These tools are further categorized as **software verifiers** which are described as limited in their capabilities in that they can only check simple properties. They include: Frama-C, BLAST, Java Pathfinder, Spark ADA, and Malpas and **Hardware Description Language (HDL) Verification tools** which were produced due to developers demand for checking hardware designs; this in turn enabled the research into and production of these tools. They include SAT solvers, BDDs, ROBDDs and the SMT solvers.

2) Tools for creating provably correct design:

Based on set theory, these tools provide a formal framework and methodology for mathematically modeling and proving systems' properties. They possess the capabilities of tackling large problems as long as they are configured to be adapted for the domain of the problem area. This can be achieved by narrowing the semantics to deal with a specific problem. They include VDM and Z, B, Event-B and Rodin.

III. Formal Methods: Practice and Experience

This paper reports review of major publications of surveys conducted to collect information of the use of formal method in industrial software development projects. This report also includes issues surrounding industrial use of formal methods, future industrial wise use of these methods as well as the description of the development of a verified repository as part of the worldwide Verified Software Initiative. Subjects discussed in this paper include Software Program Verification; Specification, Verification and Reasoning about Programs; Mathematical Logic as well as Automatic Programming.

The authors define formal methods as mathematical techniques, supported by tools to enable the development of software and hardware systems. They enable developers to analyze and verify models at any stage of the program development lifecycle including requirement engineering, specification, architecture, design, implementation, testing, maintenance and evolution of programs [1]

In requirement engineering for example, formal methods are described as useful in eliciting, articulating and representing requirements and that automated tools can help provide the support needed for checking completeness traceability, verifiability and reusability of software and for supporting requirement evolution, diverse viewpoints and managing inconsistencies. In the specification stage, the methods help to develop precise technical specifications of the purpose of the software. In architecture, formal methods help to develop software architectural models. In software design, these methods support data refinement including state machine specification, abstraction functions and simulation proofs. When the stage reaches implementation, formal methods can be used to verify program source code to prove algorithm correctness theorem. According to the authors, formal methods in testing software, still requires further research. The paper does not describe how formal methods can be used in maintenance and evolution of programs (Woodcock, Larsen, Bicarregui, Fitzgerald, 2009). [1]

One of the surveys conducted, by Austin and Parkin 1993, revealed that majority of the industry heads made use of model-oriented formalism i.e. VDM and that they focused more on specification rather than verification. Furthermore, the surveys conducted by Bloomfield and Craigen and his colleagues, focused on model-oriented approaches i.e. process calculus (CSP) and verification environments. The case studies are of small-scale systems. One challenge presented with regard to the use of formal verification tools is the that immaturity of theories and tool bases resulted in case study subjects requiring higher levels of effort to make use of these formal method tools and theories. The main reason being, tools that are developed are not transferred across platforms or research groups and that they often do not advance methodologically or theoretically.

A technical report produced by Rushby aimed to explain to NASA stakeholders the benefits or adopting formal methods in the development and certification of critical systems in the aerospace domain, the report concluded that academicians saw formal methods as inevitable, but practitioners saw them as irrelevant. The report also documents that the challenges that formal methods face include weaknesses in notations, tools and education.

Section 3 of this paper presents a quantitative survey of industrial practice of formal methods from a number of industrial projects known to have employed formal techniques in software development tools and computing applications in transportation, finance, defense, telecommunications, nuclear, healthcare, consumer electronics, space, semantic web, resource planning, automated car parking, embedded software, engineering, manufacturing as well as office and administration sectors.

Formal methods were shown to reduce the overall time of developing the software and that the specification phase took the longest time to complete. More industries responded positively that formal methods improved productivity and that once the code generation of the tools had been bootstrapped, they helped to reduce the cost of the projects. Fewer industries reported increase in the cost of production due to lack of precise complete information about externally visible behavior of the software product. However, once the code was implemented, the required behavior became clear. 92%

of responses reported an increase in quality of their products; 36% reported that formal methods improved detection of faults; 12% reported improvement in design, 10% reported increased confidence in correctness; 10% reported improved understanding and 4% reported that they were able to identify faults and other issues early on in the development cycle.

Additionally, the authors report that 61% of respondents agreed that the use of formal methods in their projects was successful; 46% agreed that the techniques used were appropriate for the tasks required. In the survey report, 56% of respondents were satisfied with the tools employed in their projects and 75% of the respondents agreed they intend to make use of formal techniques.

Projects involved in this survey include:

The Transputer Project that involved the development of microprocessor chips designed specifically for parallel processing with floating point units that are complex devices prone to device bugs; hence required months of testing. Formal methods were used to verify correctness of the OCCAM programming language. Hoare logic a formal method revealed errors in rounding of remainder operations. This resulted in the development project running three months faster than the alternative informal development that ran concurrently.

Railway Signaling and Train Control is a project launched by MANTRA Transport and the Parisian Public Transporter Operator to computerize signaling systems for controlling RER regional express network commuter trains in Paris with the objective of increasing network traffic by 25% while preserving existing safety levels. The software developed was known as SACEM consisting of 21,000 Modula-2 code. This code which was regarded safety-critical was verified using a method constructed in B language and the proofs were done interactively using automatically generated verification conditions for the code. The formal techniques focused on ensuring safety of the overall system, these included online error detection, software validation and fault tolerance of the onboard-ground compound system. The challenge in this project was the communication gap between verifiers and signaling engineers who were not familiar with the B-method. This was addressed by providing the engineers with a French description manual of the formal specification.

Mondex Smart Card an electronic cash system suitable for low-value cash-like transactions developed by the National Westminster Bank and Platform Seven. The formal verification methods were employed to verify that the card was secure. Formal proofs were employed to verify correspondence between the high-level abstract security policy model and the lower-level concrete architectural design. To ensure that the concrete design obeyed the abstract security properties, the specification and proofs were conducted using Z. The proofs revealed a small flaw in one of the minor protocols. During the testing phase no errors were detected due to previous employment of formal verification methods.

Formal methods were deployed to specify and refine AAMP Microprocessors. This project was not successful due to complexity of the language and the only tools available were syntax and type checkers. In a subsequent experiment on AAM5, a microprocessor with 500,000 transistors with performance between an Intel 386 and 486, PVS was used to formally prove correctness of its microcode instructions. This experiment proved that it was it was technically possible to prove the correctness of microcode. After that additional experiments were conducted on AAMP-FV using PVS and AAMP7 using ACL2 to prove whether that a line-by-line model of the microcode adhered to a security policy when partitions are allowed to communicate. This also yielded encouraging results.

Airbus used SCADE tool which manages the evolution of a system model as requirements change. Benefits reported included a significant decrease in coding errors for the Airbus A340 project including shorter requirement changes with improved traceability as well as major improvement in productivity.

Another project involved in the survey include Maeslant Kering Storm Surge Barrier, a movable barrier protecting the port of Rotterdam from flooding during adverse weather and sea conditions. The project employed formal modelling and verification in the analysis, design and realization phase of the system development cycle focusing on decision-making subsystem and its interfaces to the environment. Data and operations were modelled in Z and later embedded into a Promela model describing control. SPIN model checker was used in design validation, because of its ease of use. Formal techniques employed did not detect many major defects in the system. 85% of the problems arose during development phase and 15% during the reliability and acceptance test.

In the The Tokeneer Secure Entry System, Tokeneer ID Station (TIS) project was used to re-develop one component of the Tokeneer system. The SPARK tools written in Ada were used to generate verification conditions for partial correctness and run-time errors.

The \Mobile FeliCa" IC Chip Firmware was a project that used VDM++ and VDM. It is reported that from a quality perspective, more errors were found in the early phases of the development than in other similar projects at FeliCa Networks. A total of 440 defects were detected during the requirements and specification phases of the project. Of the total errors, 278 were discovered with the help of VDM++. 162 were discovered as a result of review of the model and out of these 116 were discovered using VDMTools interpreter with test cases against the executable VDM++ model.

The final part of this paper presents the authors observation of verification technology and formal methods, which include the lack of widespread adoption and use of these methods in the system development practice with an exception on the development of safety-critical systems. The authors however do indicate that there has been gradual success in application of formal methods to solve problems of industrial scale. They also note that formal methods have potential in software development in certain domains including code verification especially in traditional high-integrity critical applications but need to be performed highly technical specialists.

Challenges include entry cost of formal methods. However, it is noted there could be decrease in cost with repeated use. Emphasis for future use is on developing tools and tool chains as well as developing certification for specialists' practitioners and that automated formal analysis is making it possible to have an impact on large-scale development. The survey uncovered the importance of producing tools that merit commercial applications and that future tools should address practical issues such as providing multi-language support, porting to a variety of platforms, version control, and assistance for co-operative working by multiple engineers on single developments. Increased automation was also suggested, and that proof tools need to be incorporated in other tools used for analysis, testing and design and the overall development environment. The paper reveals that there is not enough to support the relationship between the adoption of formal verification methods and reduction in overall development cost.

The final part of his paper presents the Verified Software Repository. Initially proposed by Hoare in 2003, as a challenge for the computer science community to develop a verifying compiler, it, over the years morphed into a challenge to develop a Verified Software Repository which would contain hundreds of programs and program modules for software verification. Early pilot projects in this challenge include Verified File Store, FreeRTOS a real-time performance mini-kernel, Radio Spectrum Auctions, Cardiac Pacemaker and a Hypervisor.

IV. AIRBUS ADAPTATION OF FORMAL VERIFICATION TOOL SAFETY CRITICAL APPLICATION DEVELOPMENT ENVIRONMENT (SCADE)

The following section presents the Airbus project that has been using formal verification tool SCADE. I chose to present this project because *Formal Methods: Practice and Experience* that presented the Airbus project that was published in 2009, noted that Airbus, adopted formal methods ten years before this paper was published. This means that, despite the claim that formal methods have not been widely adopted in systems development industries, Airbus adopted these tools since late 1990s. This to me suggests that if an industry such as Airbus, that develops safety critical applications and provides services worldwide, has made formal methods part and parcel of their development live cycles for 10 years, then they must have realized the benefits of these methods earlier than many other domains in other industries.

SCADE – Safety Critical Application Development Environment is a formal method used by Airbus in a number of projects including the verification and modelling the designs of the Airbus DO-178B Level A controllers for their aircraft models A340 aircraft series as well as A380 and A400M critical on-board software, and for the secondary flying command system of the A340 aircraft.

SCADE consists of 1. a Graphical editor 2. Simulator 3. a Code generator that automatically translate graphical specification into C code 4. a model checker [3]. Esterel Technologies, reported that SCADE significantly decreased the number of coding errors during the development of Airbus's DO-178B Level A Controllers for the A340 series aircraft.

Another research paper, reports that Airbus has used formal methods for years to specify avionics systems. These methods have helped Airbus to significantly shorted development cycles of their flight control systems especially the automatic generation of code from formal specification tool SCADE. The first project that Airbus in collaboration with ONERA was launched to prove that the use of formal techniques based on SCADE language and SCADE Design Verifier for specifying avionic systems yields positive results.

SCADE which uses on graphical dataflow synchronous language, divides time into discrete instants by a global clock. The synchronous program reads inputs from external environment and computes the output.

Formal verification of properties is achieved by expressing the properties derived from system requirements and formalization and later expressed in SCADE using synchronous observer techniques. The process begins with property P and system S and hypothesis H. What is required is proof that P is satisfied by S under a set of H. To the designer builds a System S' (S prime); the verification is required to prove that the output of system s' is true.

The SCADE Designer Verifier tool aims to prove that the property is proved, otherwise a counter example that falsifies the property is generated. If property could not be proved, then it will display the results as indeterminate, neither true nor false. [4]

REFERENCES

[1] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods," *ACM Computing Surveys*, vol. 41, no. 4, pp. 1–36, Jan. 2009.

[2] R. J. Punnoose, R. C. Armstrong, M. H. Wong, and M. Jackson, "Survey of Existing Tools for Formal Verification.," Jan. 2014.

[3] T. Bochot, P. Virelizier, H. Waeselynck, and V. Wiels, "Model checking flight control systems: The Airbus experience," *2009 31st International Conference on Software Engineering - Companion Volume*, 2009.

[4] G. Durrieu, O. Laurent, C. Seguin, and V. Wiels, "Formal Proof and Test Case Generation for Critical Embedded Systems Using Scade," Building the Information Society IFIP International Federation for Information Processing, pp. 499–504.