Formal Verification: Practice and Experience and Survey of Existing Tools

Edward Escobedo
Lewis University
CPSC-59700 Week 4 Assignment
One University Parkway, Romeoville, IL 60446
edwardescobedo@lewisu.edu

***Abstract*—Formal methods are used as mathematical models for analysis and verification in the program life-cycle. These methods are now in worldwide use because of the effectiveness in verifying if a system is meeting its specified requirements. To do this, we must use information gathering techniques to compile statistics. These stats than can be compared and used as lessons learned to help guide future projects. The goal here is to determine which tool to use and apply it early in the specification and design process. This paper will summarize the two articles, Formal Methods: Practice and experience which discusses the use of formal methods by reporting how surveys are used and the observations obtained from these studies. The next paper analyzes how formal methods are used and their effectiveness of commercial and open source tools. Next, we will discuss a tool for checking abstract models and the differences between them.**

Keywords—Formal Methods, Surveys, Current Use and Trends, Spin, Uppaal, SMV, NuSMV, FDR, Alloy, Simulink Design Verifier Tools for Checking Hardware and Software

I. Introduction

These Formal methods in use today are mathematical techniques, which are often supported by tools, which help develop hardware and software systems. The rigor of mathematics enables developers to analyze and verify these tools and models at any point in the life-cycle which includes the specification, engineering, design, architecture, testing g, implementation, and maintenance. As we can see, the aim here is to ensure the development and monitoring of systems correctly and to ensure the detection of any issues that may arise. The earlier the detection has a significant impact on time, money and resources that will be needed to correct. With the ever increase of technology and our dependence on it, business spends more time on the maintenance than the development and design of the product. The complexity of software requires careful planning and specification. This is where to models and tools can help.

The formal models used today are a collection of techniques used to analyze the description of our information systems as a mathematical component. These methods are useful in going through the large amounts of combinatorial spaces of information systems and making quantitative assessments of its properties in which testing misses or cannot accomplish. There are two standard categories, automated model checkers, and theorem provers. Ensuring the correctness of systems under any and all inputs is a complicated process. The surveys used in these readings is to help identify existing formal methods tools and techniques for system verification. Of course, these reviews are not all inclusive, but they do capture all common aspects.

II. Formal Methods: Practice and Experience

A. Surveys of Formal Methods Practice

Formal methods inclusion into the industry has been an ongoing objective for decades. The benefits gained by reducing defects in designs and code need to be achieved at reasonable costs and constraints within the business process. In the 1990's questions arose whether formal methods can ever be used. This brought about several questionnaires and surveys used to identify challenges in the practice and experience that research looks to address [6].

B. Current Use and Trends

A questionnaire was distributed to the industry from November 2007 to December 2008. We must understand the breakdown of the different application domain and types to receive a fair assessment. The transport sector and the financial sector had the biggest input from all the responses gathered. The biggest application types are real-time applications, distributed and transaction processing industries. Most of the projects surveyed had thousands of lines of code and varied in the techniques used [2].

C. Outcomes

The effects of the formal techniques used are calculated against time, cost, and quality. The survey showed that the effect on time was an average benefit. 35% reported an improvement on time vice 12% for worsening. Over half of the respondents indicated it had no effect.

Of the effects on costs, 37% reported a reduction. A substantial improvement was noted due to the code generation from the specifications from the tool. Those that reported an increase in costs was due to the lack of information about the behavior of the product.

Of the three areas assessed, quality by far had the biggest improvement with the use of formal techniques. A 92% increase in the areas of detection of faults, development of

designs, increase confidence in correctness, improve understanding and early detection of defects are the areas of impacted positively [2] [6].

The conclusion of this survey had a positive influence on the successful use of formal methods by 95% agreeing or strongly agreeing. What this shows that applying formal methods for system verification most likely then have a positive outcome and it can be a valuable tool in the design of the application of the product and should be used.

III. Survey for existing tools and formal verification

Applying formal methods is just the use of mathematics to the program whether it is hardware or software. Because of its large combinatorial state makes the predictability a problem. The Turing Halting Problem describes how systems are deterministic and unpredictable which is the cause if cyber security issues. Formal methods along with the design process look to address this security and safety concern.

Limitation of testing and simulation arise from functional and safety and security properties because of a systems large state in which comprehensive testing is rarely possible even in simple systems. This is why formal methods are used and as mentioned earlier, fall into two categories [1]

- Model checkers – checks the design with secured properties in modeling language automatically without must human interaction. Outcomes are either satisfied, not satisfied and indeterminate.
- Theorem Provers – combines automated techniques along with manual guidance to prove correctness. It is a preferred method between the two.

A. Tools for Checking Abstract Models

Tools that fall into this category are created in custom languages that are amenable to formal verification. Their targets are specific application domains. The following is a brief synopsis of the most common abstract model tools [1].

Spin was developed in 1980 by Bell Labs and was used for call processing on telephone switches. It is an open source tool and currently maintained by NASA/JPL. It is a formal verification targeted at software algorithms. It uses the Promela language which is a C-like syntax. Spin models are used to analyze two property types, specific properties of small models and high-level properties of large systems.

Uppaal is a model checking tool for times automated. It has the similar functionality of Spin but except the treatment of time. It was initially, and educational tool but is now used in the commercial sector.

NuSMV is used for hardware design. SMV is the follow on the NuSMV but was created to be used as a software tool for verification of temporal properties of finite systems.

Failure Divergence Refinement is a modeling tool used to check models expressed in an algebra of Communication Sequential Processes. FDR is shaped by a set of process connected by synchronization events.

Alloy is primarily an object and structure modeling language based on set theory. It can translate first order logic into boolean expressions. It is used mostly for analyzing the consistency of software data structures. It does not examine temporal properties.

The last abstract tool discussed is Simulink. This is a tool a primarily a design and simulation tool used by engineers and not a formal verification tool. The Simulink Design Verifier is the toolbox for Simulink which allows engineers to incorporate formal verification into the design process.

B. Tools for Checking Hardware

There are several hardware checking tools used today, and all are commercial tools. A yearly license is required to operate. Some of the hardware checking tools are Questa Formal, Solidify, JasperGold, and Incisive. Questa Formal is an easy tool to understand. It provided a basic but established commercial tool, lacking some of the advanced formal methods research features found on other instruments. Solidify has been on the market the longest but has a low market share due to critical bugs found on this tool. JaperGold seems to be the most capable of all the hardware tools. It is the leading tools regarding features, performance, and usage. Incisive is the last tool discussed and has some of the same features as Questa Formal [5].

C. Tool for Checking Software

For software verification, the tools would be required to analyze the design directly. For software verification, this would need the tool to analysis the source code, which is difficult because of the software languages are not intended for verification. Unfortunately, there is a limited number of tools that can be used since they limited only to permit of a subset of language features so that formal verification can be traceable. The tools discussed are Frama-C, BLAST, Java Pathfinder, Spark ADA, and Malpas [1].

Frama-C allows users to specify complex functional specifications and prove the correctness of the source code. BLAST is an automatic tool that checks for temporal properties. The properties it checks for reachability, standard C logic and temporal properties specified by sequencing. BLAST is concerned an advanced model checker for software. As the name suggest, Java Pathfinder is a checker for Java bytecode. Plugins allow users to customize the checks. Pathfinder is useful for analyzing the portions that deal with concurrency and analyzing the abstract model. Spark ADA defines subsets, and its properties can be defined as pre-and post-conditions to subprograms, loop invariants, and data flow relationships. Finally, Malpas checks the mathematical conformance to specifications.

D. Tools to Create a Provably Correct Design

Up to this point, the tools discussed are after the fact verification. All this means is that the design is created then checked for verification. The tools described in this section involve formal techniques is the creation of conception. Once the design is completed, it is correct by construction.

The first tool is called the Vienna Development Method. This is a collection of techniques used for modeling programs. The process contains data types, model creation, model analysis, and refinement. These tools perform modeling at an abstract level which is then transformed into a detailed design using refinement. Event B is based on set theory to describe abstracts and Rodin are tools used to aid in the development of Event-B models.

IV. System verification in telecom applications

There has been a very proactive search investigating the process of formal methods for solving problems in telecommunications. There is a clearly defined need, and many in the industry question whether formal methods have failed or neglected to deliver. Some of the major concerns are whether the models used can scale up to the industry requirements, unknown interactions, simple techniques used, benefits outweighing the cost, and the complexity of the tools. Historically failure has arisen due to the program design without analysis methods in place at every stage of the lifecycle. An example of this is the programming languages used is the 70 and 80 where no requirements, specification, testing, and design were defined [3][7].

A. Model Checking

This is an automatic technique used to verify the finite state concurrent system. This method has its advantages over the traditional methods and has been successfully used to verify complex sequential circuit designs and communication protocols. Two main approaches apply – logic based and automate based. The logic model is based on temporal logics, and the automata model is based on language containment. The model checker provides the means to ensure the design meets the given specifications, but it impossible to determine whether all given specifications cover all the properties. The challenge with model checking is dealing with state space explosions problem which occurs when many components interact with one another, and the structures can assume different values. A successful approach to counter this is to use symbolic model checking. This technique use transition and output functions during reachability analysis and encoded by ROBDD. Some ROBDD verification tools are SMV, and VIS and FormalCheck [4][7].

B. Equivalence Checking

Equivalence checking is a technique that proofs the functional equivalence of two designs models at the same or different levels of abstraction. The two categories for this model are combinatorial or sequential. In combinatorial checking, functions of the two descriptions are changed into a canonical form which is compared. The major advantage to this is the efficiency for a large variety of combinatorial circuits. For Sequential checking, verifying the equivalence between two sequential designs at each state. This considers the behavior of the designs. The output of the designs is then matched to ensure the equivalence of the behavior. The drawback to sequential checking is that it cannot handle large designs and complexity which leads to rapidly reaching to state space explosion.

The challenge and complexity of telecommunication systems along with the rapid requires the development of techniques to deploy new systems quickly and securely rapidly. Performance requirements are paramount in telecommunication applications. Telecommunications touches all domain whether its healthcare, financial, transportation…etc. The challenge with verification tools will be as the technology confuse to advance; these tools will need to make progress with it. Software updates are pushed out by vendors, and compatibility issues can become an issue [7].

Conclusion

As we can see, functional verification takes up a significant share of the design cycle. With a significant amount of new technology deciding which tools to use can be complicated. There is no straight forward answer, and it can become confusing and costly. The techniques and the tools needed for the verification need to be agreed early in the design cycle. Business often underestimates the complexity of the design and the expertise required to run these tools. The verification methods describe in this paper are just a few resources available. The goal here is to address design issues and arrive at the end of the project with a verified correct design at a reasonable cost. Too many times, we simply just fix errors as they occur which leads to a reactive mindset which is costly. As systems evolve, it is critical to remain alerted to the new technologies and the tools developed for process improvement throughout the life cycle of any technology.

References

[1] Armstrong, R. (2014, December). Survey of Existing Tools for Formal Verification. Retrieved April 26, 2017, from https://lewisuniversity.blackboard.com/bbcswebdav/pid-2799234-dt-content-rid-15852875_1/courses/SP17-CPSC-59700-004/Survey%20of%20Existing%20Tools%20for%20Formal%20Verification.pdf

[2] Clarke, E. (1996, December). Formal methods: state of the art and future directions. Retrieved from http://dl.acm.org/citation.cfm?id=242257

[3] Drechsler, R. (n.d.). Gate camp: Equivalence Checking of Digital Circuits in an Industrial Environment. Retrieved April 26, 2017, from http://www.informatik.uni-bremen.de/agra/doc/work/workshops/gatecomp.PDF

[4] Hui-min, L. (n.d.). Model Checking: Theories, Techniques, and Applications. Retrieved April 26, 2017, from http://en.cnki.com.cn/Article_en/CJFDTotal-DZXU2002S1001.htm

[5] Leroy, X. (n.d.). Formal Proofs of Code Generation and Verification Tools. Retrieved April 26, 2017, from http://link.springer.com/chapter/10.1007/978-3-319-10431-7_1

[6] Woodcock, J., & Larsen, P. (n.d.). Formal Methods: Practice and Experience. Retrieved April 26, 2017, from https://lewisuniversity.blackboard.com/bbcswebdav/pid-2799234-dt-content-rid-15852874_1/courses/SP17-CPSC-59700-004/Formal%20Methods%20Practice%20and%20Experience.pdf

[7] Zubair, H. (2001, April). Modeling and Formal Verification of a Telecom System Block Using MDGs. Retrieved April 26, 2017, from http://www.collectionscanada.gc.ca/obj/s4/f2/dsk3/ftp04/MQ59312.pdf