Formal Methods

Tools and Applications

Brian Dutremble
CPSC 59700
Lewis University
Romeoville, IL, USA
Brian.dutremble@lewisu.edu

***Abstract*— Tools and techniques for testing and verifying the correctness of digital systems lag behind the systems they aim to test and verify. The growing complexity of current systems and the predicted complexity of future systems demand that new techniques be created as it is not feasible to test a system as extensively with traditional testing methods as would be required to find all errors. Utilizing formal methods to verify designs requires the use of a vast array of tools. Formal verification is in its infancy in most sectors and its usage will only increase as automation becomes ubiquitous and we create more data than we can handle with traditional verification processes.**

Keywords—formal methods; automation; application; abstract models; testing and simulation; provably correct design; digital systems; verification tools

1 Introduction

Tools and techniques for testing and verifying the correctness of digital systems lag behind the systems they aim to test and verify. The growing complexity of current systems and the predicted complexity of future systems demand that new techniques be created as it is not feasible to test a system as extensively with traditional testing methods as would be required to find all errors. Utilizing formal methods to verify designs requires the use of a vast array of tools. Each tool has applications for which it is best suited. Some, for instance, work well for creating provably correct systems through refinement, while others excel at checking the correctness of existent software. Choosing the correct tool can be as important as choosing the correct parameters for verification. Formal verification is in its infancy in most sectors and its usage will only increase as automation becomes ubiquitous and we create more data than we can handle with traditional verification processes.

2 Digital Systems

Formal methods treat descriptions of systems as mathematical objects, analyzing them in order to make quantitative statements about their safety and security. There are two main types of formal methods: model checkers and theorem provers. Model checkers use algorithmic shortcuts while theorem provers require user guidance. Theorem provers are generally the more powerful of the two options, but the difference between the two systems is not as great as is often claimed.

Commercial off-the-shelf tools are easy to use but are usually very limited in their scope and are often not customizable. If absolute verification is needed (rather than just debugging), most companies will opt to invest tools that are better suited to their products. The choice often comes down to the level of expertise of the user.

2.1 Limits of Testing and Simulation

Testing and simulation involve giving a system inputs and ensuring that the desired outputs are produced. Even small digital systems require an infeasible amount of testing with traditional methods if one wants to test the functional properties of the entire state space that is created by all of the possible valid inputs. The safety and security properties of a program refer to whether the program is able to not produce an output that is forbidden. The state space concerning which inputs cause the system to move toward insecurity is obviously vast and beyond the reach of testing and simulation. The extensive state space of digital systems makes it unlikely that comprehensive testing is possible for the vast majority of systems. The solution is to use an automated mathematical approach that decides propositions about a digital system using formal methods.

2.2 Levels of Abstraction

Formal tools are designed for use at different levels of abstraction and can be classified by their capabilities and usage scenarios. Available tools can be divided into two groups: those that verify the correctness of a model and those that are used to construct a model or design. The tools that are used to verify the correctness of a model can be further divided into the three categories below.

2.3 Tools for Checking Abstract Models

Formal verification requires formal descriptions and practical design languages are not created with this in mind. Many languages lack formal semantic definitions and care

much more about usability and efficiency. The tools in this category are used to reason about the high-level properties of a system and are targeted toward specific application domains. The size of the model that can be dealt with is limited.

2.3.1 Spin

Spin is an explicit-state model-checker used for modeling interconnected components of asynchronous systems. Models are written in the Promela language and are generally compact, expressing high-level properties. Spin is used for small models when exploration of detailed properties is required or for large systems when high-level properties are to be evaluated. Spin is manipulated via the command line but also features a graphical interface.

2.3.2 Uppaal

Uppaal is similar to Spin but differs in that it incorporates the notion of time. What started as an academic tool is now a commercial product that features an interactive development environment.

2.3.3 SMV, NuSMV

NuSMV and its predecessor SMV were designed to model hardware logic design and are thought of as being to hardware what Spin is to software.

2.3.4 FDR

Failure-Divergences Refinement (FDR) was designed at Oxford and is an explicit-state model-checker used to check models expressed in the algebra of CSP. FDR can be used to create a hierarchical description of the system and utilizes a set of processes connected by what are known as synchronization events.

2.3.5 Alloy

Alloy is tool that uses a language of the same name to translate specifications into Boolean expressions and connects them to SAT solvers in order to analyze them. Alloy is most often used for structural analysis of data and does not have the ability to analyze temporal properties.

2.3.6 Simulink Design Verifier

Simulink Design Verifier (SDV) is used for the verification of formal properties written in the Simulink graphical language. The two forms in which Simulink represents designs in data-flow diagrams or state machines, both of which are well suited for formal analysis. Simulink differs from the other tools listed in that it is primarily a design and simulation tool, not a formal verification tool. It allows engineers to incorporate formal verification into their normal design processes.

2.4 Checking Hardware Description Languages

In addition to the tools used for checking abstract models there are also tools that are able to check hardware descriptions that are provided as Verilog or VHDL design files. For these tools, a set of properties is written against a synthesizable design. This is important because the Verilog and VHDL standards have features that are only used for simulation. Tools of this sort are currently very expensive but can be the most effective method of analysis in some situations.

Properties for tools that analyze hardware description languages are written in the aptly named Property Specification Language (PSL) or as System Verilog Assertions (SVA). Both of these languages can deal with concurrent and temporal logic assertions. The tools that utilize these languages often have add-ons that can analyze common hardware design problems. Primarily used in model checking, these tools transfer the hardware descriptions into simple Boolean logic and registers. There are three outputs common to all of the hardware description analysis tools (property failure, property pass, and indeterminate).

There are four major hardware description analysis tools available. Questa Formal works reasonably well, is easy to use, and integrates well into the other tools made by the same company. Solidify has been around longer than the other tools but it is filled with flaws. It is relatively inexpensive. JasperGold is one of the leading tools in this area and performs very well. Lastly, Incisive has capabilities similar to those of Questa Formal.

2.5 Correctness of Software

In order to verify a system, a formal model of the system is needed. It's important to ensure that there are no discrepancies between the model to be verified and the implementation. As was mentioned previously, software languages are not designed for verification, so there's a need for tools that are specifically designed to verify designs written in these languages.

There is a limited set of tools available for this purpose. That set includes Frama-C, a collection of tools for the static analysis of C source code that allows the user to specify complex functional specifications. BLAST is used to check the temporal properties of C programs and is one of the most advanced model-checkers for software. Java Pathfinder is used to analyze portions of a program that deal with concurrency or to analyze an abstracted model of a program. It began as an explicit-state model checker for Java bytecode. Spark ADA is utilized for analyzing safety-critical applications and software that requires high reliability. Lastly, Malpasis primarily a static analysis tool and can analyze programs that are in the Malpas Intermediate Language.

2.6 Provably Correct Design

The tools mentioned earlier are designed to verify a system after it's been created. Designing for correctness instead involves starting with an abstract model or design that satisfies required properties and then refining that initial design until it is ready to be implemented. The two major kinds of tools for provably correct design are VDM and Z, B, Event-B, and Rodin.

2.6.1 VDM

VDM originated at IBM Laboratory in Vienna and it is a collection of tools and techniques rooted in a common specification language. VDM provides a rich set of data types and operators on those data types. It supports functional and state-based modeling. There are both commercial and open source tools available for the analysis of VDM. Refinement with VDM is a two-step process consisting of data reification and operation decomposition.

2.6.2 Z, V, Event-B and Rodin

Event-B is a formal language based on set theory and it's used to describe abstract state machines. Event-B's most powerful feature is the notion of refinement. It allows for additional detail to be added to an initial simple abstract model. Rodin is a set of tools designed to work with Event-B models. Rodin contains both a theorem prover and a model checker and can utilize both simultaneously. Rodin/Event-B is the most advanced tool of its kind and is generally used in situations where safety is paramount.

3 Practice and Experience

3.1 Current Use and Trends

Many industries currently employ formal techniques in their software-based projects. The collected data provides a look into the current reality and long-term trends of the use of formal methods. The largest application domains for formal methods are currently transportation and finance, and the respondents indicated that real-time and distributed applications were the most commonly utilized application types. Across all application domains and application types, nearly all respondents reported cost savings and increased efficiencies resulting from the use of formal methods. Increases in computational power and formal method efficiencies will drive the expansion of the use of formal methods in many industries.

3.2 Examples of Applications

In the late 1980s, a correct-by-design floating-point formal method was used to develop a floating-point unit from natural language to silicon for the Transputer series of microprocessor chips. This technique, which utilized the Occam programming language, was three months faster than the informal development that ran concurrently and each month of delay in production cost the company approximately $1 million.

Also in the late 1980s, formal verification was utilized in the development of a railway signaling and train control system. The goal was to create software that would increase network traffic while preserving safety standards. The specification for the proprietary software was written in B and the proofs were done using automatically generated verification conditions for the code. This system is still used today.

The National Westminster Bank and Platform Seven developed Mondex, a smartcard-based electronic cash system, in the early 1990s. Obviously, security was of the utmost importance, so the decision was made to certify the system to one of the highest standards available at the time. The system, among its many requirements, mandated the use of formal methods to specify the high-level abstract security policy model and the lower-level physical architectural design. Mondex eventually received its certification, which would not have been possible without the use of automated formal verification methods.

Formal methods of verification were also essential in dramatically reducing the development costs of the AAMP processors, as well as reducing the costs of the extremely expensive Airbus design cycle. Spin and its Promela language were chosen for the work in formally verifying the software responsible for deciding when to deploy the Maeslant Kering, a storm surge barrier protecting the port of Rotterdam. Additionally, formal methods of verification were used in the refinement of the NSA's Tokeneer Secure Entry System, a security system that utilizes the analysis of biometric tests to limit access to areas of the NSA's facilities.

While as of the date of the paper formal methods have not been utilized widely in all domains, their growing usage in domains where security and safety are paramount suggests that their benefits are cost-effective and thus worth implementing in other domains, particularly as computing power increases and costs of formal verification decrease.

4 Relevant Application

One of the most interesting applications for formal verification is its use in intelligent public transportation systems. Such transportation systems involve large machines and often times high rates of speed making safety extremely important. Formal verification represents an opportunity for designers of these systems to improve the safety and efficiency of the systems while reducing costs. Formal methods work particularly well in railway control systems, as railway control systems must follow a very specific set of rules.

The European Train Control System (ETCS) is a system that allows for the dense packing of trains and higher throughput at speeds of 200 mph. [3] This system must be able to maintain safe separation between vehicles on the tracks while ensuring that track time is used as efficiently as possible. Such a system requires a specification and verification language in which the correctness of the system can be stated and proven systematically. Differential Dynamic Logic (dL), the language used in the ETCS verification process, got its name because calculating the best behaviors for trains in the system often requires differential equations.

The ETCS verification process uses a compositional verification approach verifies the correct functioning the massive ETCS system by decomposing it into smaller subsystems. The tool of choice for implementing this verification system has been the new verification tool KeYmaera.

5. Conclusion

The use of formal methods has become necessary as the complexity of digital systems exceeds the capability of other verification techniques. A vast selection of tools is available for the purpose of verifying systems using formal methods. These tools require varying degrees of expertise and all have applications for which they work best. Our essential systems depend on the use of these tools to ensure safe and efficient operation. Such formal verification methods will continue to increase in value as systems increase in complexity.

[1] J. Woodcock, P. G. Larsen, J. Bicarregui, J. Fitzgerald, "Formal Methods: Practice and Experience," ACM Computing Surveys, pp.1-40.

[2] R. Armstrong, R. Punnoose, M. Wong, J. Mayo, "Survey of Existing Tools for Formal Verification," Sandia National Laboratories, pp.1-42. December 2014.

[3] A. Platzer, "Verification of Cyberphysical Transportation Systems," IEEE Intelligent Systems, Vol. 24, Issue 4, July-Aug 2009, pp.1-4.