Survey of Existing Tools for Formal Verification & Formal Methods

(Review)

Thomas Swed
Lewis University
1 University Pkwy,
Romeoville, IL 60446
thomasdswed@lewisu.edu

***Abstract*—This paper gives a summary of the concepts presented in "Formal Methods: Practice and Experience" [1] and "Survey of Existing Tools for Formal Verification" [2]. In the first article a series of projects are presented, observations are derived from these projects, and challenges related to verification tools are discussed. The latter article concentrates on an overview of existing formal verification tools and their uses.**

Keywords—Formal methods, formal verification, software verification, formal verification tool, temporal logic.

I. Introduction

Formal methods are used to specify the exact requirements or intent of a software application without describing how it is implemented. They use mathematical principles employed to asses and verify the software automatically at various stages in its development. To clearly articulate a method, specifications are developed between the programmer and the client to define the purpose of the software [3]. This enables the client to assess the software's application based on a standard, while the programmer can use it to guide the development of the code.

Formal methods are also used at the implementation stage of development to verify software. They seek to determine if a program will achieve the effect described in its documentation by affirming a correctness theorem. This is referred to as the *inductive assertion* method which utilizes mathematical assertions to annotate a program [4].

This paper gives a brief overview of the concepts developed in the two papers reviewed. These include the results from various surveys of formal methods applied, observations drawn from industrial projects, issues encountered, and an overview of the tools used for formal verification.

II. Formal Methods Applied

From November 2007 to December 2008, surveys were conducted on 62 industrial projects to obtain standardized data. The results indicated that formal verification is being used in many different applications and is not limited to a single sector. Overall, the responses affirmed the benefits of using formal methods in development. Some 75% of the respondents indicated that they will be using these methods in the future. Model checking has seen tremendous growth in the last decade, going from 13% in the 1990s to 51% a decade later [5]. Significant advances in automated verification techniques continue to be made with the goal of encapsulating them in the future. Many projects reported reduced costs, saved time and improved quality.

III. Projects

A significant challenge exists in achieving automated analysis. As the complexity of the analysis grows, so does the complexity of the verification demands. Static analyzers, such as *lint*, *Splint, PREfix* and *PREfast* exist to identify defects in programs, such as variable use before definition, constant conditions, and out-of-range expressions [6]. The downside to these analyzers is that they may produce inaccurate results due to using instable techniques. *SLAM* is a distinct tool that can verify a program to be free from specific kinds of errors. Tools such as model checkers and abstract interpreters also exist to verify logical design and consistency. Examples include *SPIN* and *ASTRÉE.* Theorem provers such as *Vampire* and *KIV* validate speculations in logic with varying levels of automation.

Some of the major projects surveyed include *The Transputer Project, Railway Signalling and Train Control, Mondex Smart Card, AAMP Microprocessors, Airbus, The Maeslant Kering Storm Surge Barrier, The Tokeneer Secure Entry System* and *The "Mobile FeliCa" IC Chip Firmware* [7]. Despite the success of these methods found in several industrial projects, formal verification methods have yet to be adopted as a standard in a systems development.

IV. Challenges

Despite advances in verification tools, they remain primitive and technical for the average user. Several challenges need to be overcome in order to achieve an easy to use tool.

A. Formal Methods

Feasibility becomes a concern for many project managers as they determine a cost-effective way to implement formal techniques in a system. In practice, various methods are applied to different components and at several stages of a given stage of development.

B. Verification Tool Support

Support for automated deduction, common formats for the interchange of models and analysis, and the lack of responsive support for tools are among the major challenges in this area. The majority of tools used in survey respondents were not widely supported for community use and were of a highly technical nature [8]. Developments still need to be made in order to enable widespread adoption. These include multi-language support, porting to a variety of platforms, version control and assistance for co-operative working by multiple engineers on single developments [9].

C. Increased Automation

Currently, a significant problem with verification tools is their technical and fine-tuned nature, making them difficult and time consuming to implement. They require a high level of knowledge to implement as they must be tailored to a specific system. Efforts are being made to encapsulate proof tools to allow analysis, testing, and design automation to take precedence. These tools should be central parts of development environments, the gap between tools and the standard production process needs to disappear, and certain analysis should be produced within existing development environments [10]. The goal being to enable an easy to use tool with a few simple adjustments.

V. Existing Tools

A. Abstract Models

- SPIN ("Simple Promela Interpreter") is an open source tool for verifying the correctness of a software model. It is an explicit-state model checker that uses models written in Promela. It is built on the C programming language and expresses properties as LTL formulas. It is particularly useful for analyzing concurrent systems [11].
- NuSMV is an extension of SMV (Symbolic Model Verifier) software. It validates temporal logic properties in finite state systems using a custom input language that allows description of synchronous and asynchronous systems [12]. NuSMV was the first model checking tool based on Binary Decision Diagrams (BDDs) and supports the analysis of specifications in CTL or LTL.
- FDR and FDR2 are explicit state model checking tools specifically for communicating sequential processes (CSP), a process calculus for concurrent systems. They take a model of the system, expressed in the algebra of CSP and the property to be checked as inputs to verify [13].
- ALLOY creates models using first-order logic to create Boolean expressions that can be analyzed. These models are characteristically relational and are used to analyze data structures and relationships. Its primary use is for structural analysis of data, as the program does not examine temporal properties [14].
- Simulink is a graphical language for the design and simulation of digital designs with control logic and signal processing [15]. It is largely used as a design and simulation tool utilizing two forms: data-flow diagrams and state machines. Consequently, the Simulink Design Verifier is used to validate formal properties characteristic of Simulink using a set of tools to integrate formal verification in the design process [16].

B. Hardware Description Languages

- Questa Formal is an easy-to-use formal verification tool currently owned by Mentor Graphics. While it lacks certain advanced features of its competitors, it has the benefit of integrating easily with the other tools offered by the company. It offers seven analysis strategies or "engines" [17].
- Solidify was found to be the worst performing and most difficult to use tool for verifying hardware description languages. While it is the most inexpensive tool, it was lacking many features found in other products. Several bugs were also found in the tools evaluation [18].
- JasperGold is sold by Jasper Design Automation as a commercial software tool. It appears to be one of the leading tools in terms of the features it offers and its performance and usage in the industry [19]. The tool was found to perform well in iterative analysis, property analysis, security verification and automatic extraction of counters for abstraction.
- Incisive is currently offered by Cadence Design Systems, which acquired Jasper Design Automation in June 2014. Incisive is similar to Questa Formal and was found to assimilate easily with other tools offered by Cadence [20].

C. Correctness of Software

- Frama-C provides the capability to create functional specifications to statically analyze C code. The specifications are written in a formal language called ANSI/ISO C Specification Language (ACSL) [21].
- BLAST is an advanced automatic tool that uses several methods for checking temporal properties of C software. Using a C program as input it can verify

three properties: reachability, standard C logical assertions, and temporal properties.

- Java PathFinder is a model checker that has evolved to verify Java byte code directly. It supports multiple customizable plugins for various properties and can search for deadlocks, assertion errors, and null pointer exceptions in the code [22]. Because of its storage restrictions however, Java PathFinder is not used to verify entire programs but only sections of code that deal with concurrency.

- SPARK was developed for safety critical applications and high reliability as a distinct subset of the ADA programming language [23]. Properties are indicated by comments alongside the code itself. They can dictate before and after conditions, loop invariants and dataflow relationships.

- Malpas is a static and formal analysis tool that analyzes software in the Malpas Intermediate Language using a translator to convert existing source code. It can verify dead code paths, unitialized data, unexpected dependencies, as well as invariants, assertions and before/after conditions [24].

D. *Proving Correct Design*

- The Vienna Development Method (VDM) is a toolset for modeling computer programs in a formal specification language [25]. It creates unions, Cartesian products and composite types using Booleans, natural numbers, integers, rational numbers, real numbers and characters.

- Z, B and Event-B are all verification tools based on set theory. Z and B are used for the specification of computer programs while Event-B is used for the specification and analysis of systems by modeling them as state machines [26].

- Rodin is a powerful suite of easy-to-use tools that provides a theorem prover and model checker to allow for the added benefit of simultaneous application. It supports multiple plugins making it easily extendable.

VI. Application

The Java PathFinder application was particularly interesting to me as Java is the language I learned how to first code with. Java PathFinder is an explicit-state model cheker, developed at the NASA Ames Research Center that accepts a Java program as an input to verify [27]. Its primary use has been as a Model checker for concurrent systems. Java PathFinder is implemented in Java code as its own virtual machine. It represents the JVM state of the program being checked and performs three main operations on this state representation: bytecode execution, state backtracking and state comparison [28]. While it does have limitations, a primary benefit of this tool is that it is an open system that can easily be extended. We will briefly describe 4 examples of extensions to the Java PathFinder: Untracked State, Undo Backtracking, Delta Execution and Mixed Execution.

A. *Untracked State*

This provides a new functionality to Java PathFinder in that it allows the user to mark that certain parts of the state should not be restored during backtracking rather than restoring the entire JVM state [29]. It is important because it enables data to be gathered on more than one execution path.

B. *Undo Backtracking*

The primary objective of this extension is to increase the speed of backtracking. Rather than storing and restoring the entire JVM state as Java PathFinder currently does, this extension will only keep track of the state changes that happen between choice points in order to restore the state [30].

C. *Delta Execution*

Delta Execution reduces the state-exploration time of Java PathFinder by comparing overlapping executions to perform just the differences found. It introduces a novel representation for a set of concrete states and a collection of efficient operations for that representation [31].

D. *Mixed Execution*

This extension translates the program state from Java PathFinder to the JVM in order to execute certain parts with the goal of increasing performance speeds.

VII. Conclusion

Formal verification has yet to be adopted as a standard for widespread use. Yet due to the limitations of testing and simulation, verification techniques have seen increased use in recent years. Formal verification in instances of security, reliability and safety is often times the only way to provide validation. Further research and developments need to be made with existing tools to extend capabilities and make them more accessible, but the benefits of such tools are clear.

References

[1] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, John Fitzgerald, Article: Formal Methods: Practice and Experience, ACM Computing Surveys (CSUR), v.41, p. 1-36, October 2009.

[2] Robert C. Armstrong, Ratish J. Punnoose, Matthew H. Wong, Jackson R. Mayo. Article: Survey of Existing Tools for Formal Verification, Sandia National Laboratories, December 2014, p. 1-42.

[3] Woodcock, 2.

[4] Woodcock, 2.

[5] Woodcock, 8.

[6] Woodcock, 12.

[7] Woodcock, 13 - 21.
[8] Woodcock, 23.
[9] Woodcock, 23.
[10] Woodcock, 24.
[11] Armstrong, 17.
[12] Armstrong, 18.
[13] Armstrong, 20.
[14] Armstrong, 21.
[15] Armstrong, 21.
[16] Armstrong, 21.
[17] Armstrong, 24.
[18] Armstrong, 25.
[19] Armstrong, 25.
[20] Armstrong, 25.
[21] Armstrong, 27.
[22] Armstrong, 28.
[23] Armstrong, 28.
[24] Armstrong, 28.
[25] Armstrong, 31.
[26] Armstrong, 32.
[27] "Java Pathfinder", Wikipedia, https://en.wikipedia.org/wiki/Java_Pathfinder, Accessed April 26, 2017.
[28] Tihomir Gvero, Milos Gligoric, Steven Lauterburg, Marcelo d'Amorim, Darko Marinov, Sarfraz Khurshid, Article: State Extensions for Java PathFinder, Proceedings of the 30th international conference on Software engineering, May 2008, p. 863 – 866.
[29] Gvero, 864.
[30] Gvero, 865.
[31] Gvero, 865.
[32]