Formal Verification Methods

Research · February 2020

DOI: 10.1007/3-540-

CITATIONS
3

READS
3,077

1 author:

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

SEE PROFILE

Formal Verification Methods

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

***Abstract*— With advancement in technology in recent years, more advanced and complex systems are being deployed. The more advanced the software, the more complex the algorithms and design. As a result, the traditional methods of testing and debugging are proving to being limited in their capabilities of testing, finding errors and debugging complex software and/or computer systems such as embedded systems and safety-critical software systems. To address these limitations, the technology industry has begun adapting Formal verification methods. These methods are not new, they were developed early on during the 20th century. However, they were not fully embraced or adapted until recent years.**

This paper aims to present the formal methods that have become popular in recent years for verifying requirement specification of software. The two methods that will be presented here include Theorem Proving and Model Checking. It will briefly analyze the popular tools for model checking such as SPIN and ProB, including the languages they use, their properties as well as the categories that they fall in. This paper will also look into the application of these formal verification methods and how they are used to address software verification/testing issues in professional activities.

***Index Terms* – Formal Methods, Model Checking, Theorem Proving, Temporal Logic, Explict State Model Checking, Symbolic Model Checking, Bounded Model Checking**

I. INTRODUCTION

In developing information systems, embedded systems and safety-critical software, it is crucial to ensure that specifications of desired requirements are verified early on during the Software Development Life Cycle.

Verification of requirements early in the cycle, saves time and cost of the overall development project. [1] According to a paper presented by Lutz, titled *"Analyzing Software Requirements Errors in Safety Critical Embedded Systems"*, 387 software errors were discovered during integration and system testing of Voyager and Galileo spacecraft, and according to Lutz, these errors most often occurred due to inadequate requirement specification.

This paper therefore, presents a review of formal verification methods that have been adapted to address the limitations of software testing and debugging that use test cases. Formal verification presented in this paper include Theorem proving as well as Model checking.

II. FORMAL VERIFICATION METHODS

As briefly described in the introduction, formal verification methods are defined as methods that aim to prove or demonstrate that the software under investigation complies with the requirement specifications that were put in place during requirement analysis phase of the Systems Development Life Cycle.

Theorem Proving is a formal verification method that makes use of theorems to prove algorithm correctness. Theorem proving makes use of mathematics and logic, where post condition (conclusion) is based on pre-condition (hypothesis). By making use of axioms and rules of interference, this method assumes that if the hypothesis is true then the conclusion must also be true.

The formula: p then q where p is the hypothesis and q is the conclusion.

Unlike theorem proving, model checking on the other hand visits a program state space to perform verification of the program's properties that are formulated in temporal logic. Model checking explores state of a system and aims to prove that a predefined software model, which can either be a set of requirements or a simulation model, comply with the specification properties. Here, the system is the sematic model and the properties are described using logical formulas.

Most model checking tools are developed using model checking languages. Two popular model checkers include, SPIN that come with its language – PROMELA and ProB that come preinstalled with B language.

Model checking is based on Computational Tree Temporal logic (CTL), which combines Linear Temporal Logic (LTL) in which various operators are provided to describe events along a single computation path. Model as well as Branching-time logic that provides operators to quantify over a set of states that succeed the current state. (Reinbacher, T., n.d).

III. MODEL CHECKING TOOLS

Model checking tools are categorized into classes:

Explicit State Model Checking tools use an explicit representation of the system's global state graph, usually given by a state transition function. An explicit state model checker evaluates the validity of the temporal properties over the model by interpreting its global state transition graph as a Kripke structure. Property validation in explicit model checking explores a partial or complete state space (Edelkamp S., Leue, S., Lluch-Lafuente A., 2013).

Model checkers that fall into this class include: C/ESAR/ALDEBARAN Development Package (CADP), SPIN and FDR2.

CADP is a toolbox for protocol engineering that also performs formal software verification. It offers an interconnected set of tools/components that validate and verify protocols though the development cycle. These include OPEN/CAESAR that offer interactivity and random simulation; OPEN/CAESAR and ALDERBARAN combined, offer either partial and/or exhaustive deadlock detection; ALDERBARAN performs verification of behavior specification with respect to bi-simulation relations; TVG is responsible for generating sequence of tests; XTL and EVALUATOR perform verification of specification of the branching time temporal logic [2].

FDR2 is an explicit state model checker for CPS, a process algebra. FDR2 supports classical algebra such as prefix, choice, parallel composition with synchronization, sequence composition and guards. It checks refinement, deadlocks, live-locks, and determinacy of process expressions. FDR2 also acts as a symbolic 'implicit' state model checker because it gradually builds the state transition graph, compressing it using state space reduction techniques while checking properties. (Dong J., Zhu H., 2010). [3]

Developed in 1980, Simple PROMEA Interpreter (SPIN) is one of the very first model checkers. It is a verifier as well as a simulator. It is designed to provide correctness of process interaction. SPIN uses propositional LTL (Linear Temporal Logic), a modal logic of reasoning about dynamic scenarios. LTL uses an infinite sequence of states where each point in time has a unique successor based on linear time perspective (Ashari R., Habib s., n.d.).

SPIN also uses computer transition system while checking property verification. Unlike CADP and FDR2 that compute transition systems prior to property verification, SPIN compute transition systems while checking property verification. It does this by transferring its Process/Protocol Meta Language (PROMELA), a verification modeling language into a finite state automaton a mathematical computation model that can be in exactly one state of a finite number of states at any given time. SPIN then creates state spaces by interleaving the finite state automata to construct state spaces.

However, interleaving of the finite state machine may at times result in State Explosion, a situation that may occur when there are too many interleaves and interactions in the model, resulting in the state space growing exponentially such that it cannot be checked thoroughly.

Symbolic Model Checking is another class of model checking tools that computes transition systems as a Boolean formula. Model checking tools that fall under this category, address the issue of state explosion that may at times occur when model checking is conducted using explicit state model checkers (McMillan K., n.d.).

One model checker that falls in this category is NuSMV which comes with a text-only interactive shell as well as a graphical user interface. NuSMV features include partitioning techniques for synchronous models, disjunctive partitioning of asynchronous models as well as the generation of counterexamples.

This model checker allows specifications to be written in both CTL and LTL. System properties can be checked with BDD-based symbolic model checking and/or Bounded Model Checking. NuSMV uses the Symbolic Model Verifier description language to specify finite state machines.

IV. AREA OF APPLICATION FOR FORMAL VERIFICATION METHODS

Some area of applications that make use of formal verification methods include verifying a model of Concurrent/Distributed Systems. An example is model checking of concurrent and distributed systems using SPIN model checker. SPIN can be used to exhibit design flaws in these systems and help to improve the design.

Formal verification methodologies can be also used to check data integrity. In validating the design of system on chip (SoC) designs for example and due to the complex nature of these designs, both formal verification and logic simulation are used. Umezawa T. and Shimizu T., of Fujitsu Laboratories of America, Inc., Sunnyvale, CA, USA, presented a report for a project in which they applied formal verification techniques on component chips for server platforms. The internal data paths registers, state machines and counters were all protected by parity bits. The requirements that were verified include parity protection in data paths, parity protection in key control structures i.e. FSM and counters as well as Illegal state detection for FSM and counters. As the number of checkpoints for the chip specification reached 1300, it was not possible to perform exhaustive verification due to time constraints. Hence model checking was applied locally at each leaf module. During verification, detected parity error was logged and reported according to the severity of the errors themselves. [4]

V. REFERENCES

[1] Lutz, R. (2018). [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/c77e/fb40fa7ad307314e109b7f8fa7429cbacd4b.pdf [Accessed 20 Sep. 2018].

[2] J. C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighreanu, “CADP A Protocol Validation and Verification Toolbox.” [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-61474-5_97.pdf. [Accessed: 24-Sep-2018].

[3] J. S. Dong and H. Zhu, *Formal Methods and Software Engineering 12th International Conference on Formal Engineering Methods, ICFEM 2010, Shanghai, China, November 17-19, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. [Accessed: 24-Sep-2018].

[4] Y. Umezawa and T. Sh imizu , “A Formal Verification Methodology for Checking Data Integrity ,” *0710.4848.pdf*, 2005. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/0710/0710.4848.pdf. [Accessed: 26-Sep-2018].

M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar, “Comparison of Model Checking Tools for Information Systems,” *Formal Methods and Software Engineering Lecture Notes in Computer Science*, pp. 581–596, 2010.

T. Reinbacher, “http://ljournal.ru/wp-content/uploads/2017/03/a-2017-023.pdf,” *Introduction to Embedded Software Verification*, 2017.