The formal Methods Approach to Software Engineering

Moayad Almarzook

Abstract- The field of mathematics is intertwined with computer related fields. So it is natural to think of a mathematical solution to software related issues. The problem is and always will be that it is very hard to implement a reliable software project. Here lays the need of formal method as an approach to minimize the chances of an error especially in critical software systems. Not say that it is an easy task but the declining of software dependency in a world, where the use of software is increasing rapidly. Urges for a better solutions.

Introduction

Many tools, approaches, and techniques have been devised for improving software reliability and dependability. Some were more successful than others in specific domains. One approach is called formal methods, in which a specification notation with formal semantics, along with a deductive tool for reasoning, is used to specify, design, analyze, and implement a hardware or software system. This approach is thought to be hard to apply and require a significant mathematical knowledge. The more complex the system is, the more specialization in mathematics-and engineering-related areas are needed for more difficult formal methods related tasks such as verification and refinement and writing formal specification.

Three world-renowned experts in software engineering, abstract interpretation, and verification of concurrent systems Contributed in this article: Michael A. Jackson, Patrick Cousot, and Byron Cook. Their contribution was based on their speeches at the IEEE's Fifth International Conference on Software Engineering and Formal Methods, which was held in 2007 in London. The goal of the conference was to bring practitioners and researchers together to exploit synergies and further the understanding of specialization, abstraction, and verification techniques.

Formal Methods and Formal Design

In Michael A. Jackson speech he argues that computer system are influencing all aspects of people's lives. From the moment they wake up until they fall asleep. Computers are used increasingly in critical application that could cause disasters if they fail to operate as expected. This raises the question about the reliability of the software. This is a major issue for software engineers because, software has been exhibiting declining levels of dependability. Jackson attributed Many reasons to this issue including:

1. The belief that anyone who can write a software can write a good software.
2. Running a few representative test cases indicates that the software is correct or adequate.
3. Not realizing that a good design is more important than producing a lot of code.
4. Not realizing that making unnecessary, uncontrolled, and careless changes. Can affect the validity of the software.

Software intensive systems are intended to interact with human behavior and use real-time information to provide required functionality. These types of systems are a challenge because of the non-formal nature of human behavior. The software can be regarded as a formal system for all but the most extremely critical systems the computer behavior can be assumed to conform to the program semantics. For a dependable system. There must be an adequate formal model of the world, and the software must be designed to reflect the assumption that the world conforms to this model. The computer behaves as if the world model is valid. Once the world deviates the model fails. Fault-tolerant techniques can be implemented, but they are not enough to eliminate all the problems.

The development of a formal model is an engineering task. In which, establishing engineering branches, challenges are met by experience accumulated in each particular product class and captured in a normal design discipline. Jackson uses the aeronautical engineer W. G. Vincenti's explanation of normal design: "The engineer knows at the outset how the device in question works, what are its customary features, and that, if properly designed along such lines, it has a good likelihood of accomplishing the desired task." The deviations in a normal design are sufficiently improbable, these deviations are implicitly handled by the configuration and which must be considered in the calculations and checks mandated by the normal design discipline.

Software engineering and computer science have many specializations; however, these specializations are not enough to build dependable software intensive systems. The development of more specializations is a must. They have to be heavily focused on narrow classes of end products and their subsystems. Dependency can only be achieved by bringing together the contributions of

specialists in all the different aspects, parts, and dimensions of the design task.

Patrick Cousot defines formal methods as a mathematical techniques for specifying, developing, and verifying of software and hardware systems. Establishing satisfaction of a property by a formal model of the system behavior is called semantics. The semantic domain is a set of all such formal models of system behaviors. A property of the system is a set of semantic models that satisfy this property. The satisfaction of a specification by a system is called its collecting semantics. The semantics and the specification of a complex system are very difficult to define. That makes the Formal verification methods very hard to put in practice. Even when it is possible the proof cannot be without costs.

Abstract Interpretation is a theory of sound approximation of mathematical structures.

The abstraction idea is to consider a sound over approximation of the collecting semantics, a sound under approximation of the property to be proven, and to make the correctness proof in the abstract. It formalizes the intuition about abstraction, and allows the systematic derivation of sound reasoning methods and effective algorithms for approximating undecidable or highly complex problems in a number of computer science fields. Abstract Interpretation is currently use on complex hardware and software computer systems security.

Verification by Static is a fully automatic analysis of a computer system by directly inspecting the source or object code describing the system with respect to the semantics of the code, and it is proven by computing an abstraction of the collecting semantics of the system. Few successful examples are aiT (www.absint.com/ait), where they computed an over approximation of the worst-case execution time, and Astree (www.astree.ens.fr), where they computed an over approximation of the collecting semantics to prove the absence of runtime.

Byron cook said that various applications involve concurrency, where parallel activities are in progress. This could create unexpected complex interactions, making verification a complex problem. Traditional methods of specifying and automatically reasoning about computer systems are not sufficient in this case. Specialized tools would be required for verification; however, recent advances addresses these issues.

Thread modularity is a common theme in those advances, which refer to existing sequential-program proofs that can be used to prove the correctness of concurrent programs if appropriate abstractions can be found to represent the other threads in a concurrent system. Heuristics must be developed to deal with the difficulty of thread-modular techniques, because the space of abstractions is so large that finding the right one for a given program verification is difficult. Also through the use of experimental evaluations, these heuristics must be shown to work in the common case.

Conclusion

Summing up, only through solid design principles that software intensive system can achieve dependability, and that in turn is achieved through the understanding of the product and specialization of engineers. Abstract interpretation can reduce the complexity of proving properties correctness of complex software systems. Verification techniques have seen some recent advances, which made it possible to verify concurrent systems. Formal methods has characterized the first 20 years of this field making it an art. Today it is not restricted to few users anymore. First transition of formal methods was a move from specification only toward tool based semantics analysis. The second transition was a move from proof assistants to fully automatic analyses embedded in usual development environment. This shift would enable developers to use these techniques easily, making it an everyday practice.

References

Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J., & Margaria, T. (2008). Software engineering and formal methods. *Communication of ACM*, *9*(51), 59-59.

IJSER