Formal Methods

Carnegie Mellon University
18-849b Dependable Embedded Systems
Spring 1998

Michael Collins

Abstract

Formal methods are techniques used to model complex systems as mathematical entities. By building a mathematically rigorous model of a complex system, it is possible to verify the system's properties in a more thorough fashion than empirical testing.

While Rigorous descriptions promise to improve system reliability, design time and comprehensibility, they do so at the cost of an increased learning curve; the mathematical disciplines used to formally describe computational systems are outside the domain of a traditional engineering education. In addition, the metamodels used by most formal methods are often limited in order to enhance provability. There is a notable tradeoff between the need for rigor and the ability to model all behaviors.

Related Topics:

- Requirements Specifications
- Software Testing
- Verification

Contents:

Introduction

Formal methods are system design techniques that use rigorously specified mathematical models to build software and hardware systems. In contrast to other design systems, formal methods use mathematical proof as a complement to system testing in order to ensure correct behavior. As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance.

Formal methods differ from other design systems through the use of formal verification schemes, the basic principles of the system must be proven correct before they are accepted [Bowen93]. Traditional system design has used extensive testing to verify behavior, but testing is capable of only finite conclusions. Dijkstra and others have demonstrated that tests can only show the situations where a system won't fail, but cannot say anything about the behavior of the system outside of the testing scenarios [Bentley89]. In contrast, once a theorem is proven true it remains true.

It is very important to note that formal verification does not obviate the need for testing [Bowen95]. Formal verification cannot fix bad assumptions in the design, but it can help identify errors in reasoning which would otherwise be left unverified. In several cases, engineers have reported finding flaws in systems once they reviewed their designs formally [Kling95].

Roughly speaking, formal design can be seen as a three step process, following the outline given here:

1. Formal Specification: During the formal specification phase, the engineer rigorously defines a system using a modeling language. Modeling languages are fixed grammars which allow users to model complex structures out of predefined types. This process of formal specification is similar to the process of converting a word problem into algebraic notation.

   In many ways, this step of the formal design process is similar to the formal software engineering technique developed by Rumbaugh, Booch and others. At the minimum, both techniques help engineers to clearly define their problems, goals and solutions. However, formal modeling languages are more rigorously defined: in a formal grammar, there is a distinction between WFFs (well-formed formulas) and non-WFFs (syntactically incorrect statements). Even at this stage, the distinction between WFF and non-WFF can help to specify the design. Several engineers who have used formal specifications say that the clarity that this stage produces is a benefit in itself [Kling95].

2. Verification: As stated above, formal methods differ from other specification systems by their heavy emphasis on provability and correctness. By building a system using a formal specification, the designer is actually developing a set of theorems about his system. By proving these theorems correct, the formal

   Verification is a difficult process, largely because even the simplest system has several dozen theorems, each of which has to be proven.

Even a traditional mathematical proof is a complex affair, Wiles' proof of Fermat's Last Theorem, for example, took several years after its announcement to be completed. Given the demands of complexity and Moore's law, almost all formal systems use an automated theorem proving tool of some form. These tools can prove simple theorems, verify the semantics of theorems, and provide assistance for verifying more complicated proofs.

3. Implementation: Once the model has been specified and verified, it is implemented by converting the specification into code. As the difference between software and hardware design grows narrower, formal methods for developing embedded systems have been developed. LARCH, for example, has a VHDL implementation. Similarly, hardware systems such as the VIPER and AAMP5 processors have been developed using formal approaches.

An alternative to this approach is the lightweight approach to formal design. In a lightweight design, formal methods are applied sparingly to a system. This approach offers the benefits of formal specification, but also avoids some of the difficulties.

Formal methods are viewed with a certain degree of suspicion. While formal methods research has been progressing since 1960's, formal methods are only being slowly accepted by engineers. There are several reasons for this, but most of the problems seem to be a result of misapplication. Most formal systems are extremely descriptive and all-encompassing, modeling languages have generally been judged by their capacity to model anything. Unfortunately, these same qualities make formal methods very difficult to use, especially for engineers untrained in the type theory needed for most formal systems.[Bowen93]

Conversely, it is apparent that some form of formal specification is necessary: complex systems require formal models. In addition,the mathematics required for formal methods is becoming a more prominent fixture of engineering curricula, engineering schools in Europe are already requiring courses in VDM, Z and similar formal specifications. Ultimately, formal methods will acquire some form of acceptance, but compromises will be made in both directions: formal methods will become simpler and formal methods training will become more common.

Key Concepts

• Provability And Automated Verification

Formal methods are distinguished from other specification systems by their emphasis on correctness and proof, which is ultimately another measure of system integrity. Proof is a complement, not a substitute, for testing. Testing is an important part of guaranteeing any system's

fitness, but it is finite. Testing cannot demonstrate that a system operates properly; it can only demonstrate that the system works for the tested cases. Because testing cannot demonstrate that the system should work outside the tested cases, formal proof is necessary.

Formally proving computer systems is not a new idea. Knuth and Dijkstra have written extensively on the topic, although their methods of proof are based on the traditional mathematical methods. In pure sciences, proofs are verified through extensive peer review before publication. Such techniques are time-intensive and less than perfect; it isn't unusual for a published proof to contain a flaw. Given the cost and time requirements of systems engineering, traditional proving techniques are not really applicable.

Because of the costs of hand verification, most formal methods use automated theorem proving systems to verify their designs. Automated theorem provers are best described as mathematical CAD tools: they can prove simple propositions and automatically and provide assistance for verifying more complex theorems.

- # Benefits Of Formal Models

Formal methods offer additional benefits outside of provability, and these benefits do deserve some mention. However, most of these benefits are available from other systems, and usually without the steep learning curve that formal methods require.

- **Discipline**: By virtue of their rigor, formal systems require an engineer to think out his design in a more thorough fashion. In particular, a formal proof of correctness is going to require a rigorous specification of goals, not just operation. This thorough approach can help identify faulty reasoning far earlier than in traditional design.[Bowen95]

  The discipline involved in formal specification has proved useful even on already existing systems. Engineers using the PVS system, for example, reported identifying several microcode errors in one of their microprocessor designs.[Miller95]

- **Precision**: Traditionally, disciplines have moved into jargons and formal notation as the weaknesses of natural language descriptions become more glaringly obvious. There is no reason that systems engineering should differ, and there are several formal methods which are used almost exclusively for notation.[Bowen93]

For engineers designing safety-critical systems, the benefits of formal methods lie in their clarity. Unlike many other design approaches, the formal verification requires very clearly defined goals and approaches. In a safety critical system, ambiguity can be extremely dangerous, and one of the primary benefits of the formal approach is the elimination of ambiguity.[Kling94].

- # Weaknesses Of Formal Methods

: Bowen points out that formal methods are generally viewed with suspicion by the professional engineering community, and the propensity of tentative case studies and advocacy papers for the formal approach would seem to support his thesis [Bowen93]. There are several reasons why formal methods are not used as much as they might be, most stemming from overreaching on the part of formal methods advocates.

  - **Expense**:Because of the rigor involved, formal methods are always going to be more expensive than traditional approaches to engineering. However, given that software cost estimation is more of an art than a science, it is debatable exactly how much more expensive formal verification is. In general, formal methods involve a large initial cost followed by less consumption as the project progresses; this is a reverse from the normal cost model for software development.[Bowen93]
  - **Limits Of Computational Models**:While not a universal problem, most formal methods introduce some form of computational model, usually hamstringing the operations allowed in order to make the notation elegant and the system provable. Unfortunately, these design limitations are usually considered intolerable from a developer's perspective.

    An excellent example comes from SML. Statements of proof in SML depend on a purely functional programming model: all data is passed through the parameter/return mechanism of a function, no side effect alterations, modifications of global variables or the like is allowed [Paulson96]. Handling side effects and other aberrancies are a requirement for any system involving input, network operations or other systems which require interrupts, meaning that SML's model is, to some extent, broken.

  - **Usability**:Traditionally, formal methods have been judged on the richness of their descriptive model. That is, 'good' formal methods have described a wide variety of systems, and 'bad' formal methods have been limited in their descriptive capacities. While an all-encompassing formal description is attractive from a theoretical perspective, it invariably involved developing an incredibly complex and nuanced description language, which returns to the difficulties of natural language. Case studies of full formal methods often acknowledge the need for a less all-encompassing approach. [Miller95]

Arguably, many of these failures can be attributed to overreaching on the part of formal methods advocates. This reasoning has led to the lightweight approach to formal specification.

- # The Lightweight Approach

The flaws in formal specifications have been heavily focused on in the past few years, leading to several alternate approaches. The traditional view of formal methods as all-encompassing highly abstracted schemes has led to formal methods being all-encompassing, extremely rigorous, and very expensive. While theoretically appealing, formal methods have generally been ignored by engineers in the field.

The lightweight approach to formal design recognizes that formal methods are not a panacea: there are areas where formal methods are useful, and areas where a formal specification will accomplish nothing. In a lightweight design, formal methods are used in specific locations, and different formal methods may be used in different subsystems, ideally playing to the strengths of each method [Easterbrook 98]. In such a system, Petri Nets might be used to describe the communications protocol, and a LARCH system might be used to model the data storage. For other parts of the system, formal specifications might be avoided entirely: for example, the user interface may be refined using a rapid prototyping system and client interviews.

The lightweight approach is a traditional engineering compromise, and there is a tradeoff. As formal methods become more common, engineers will have to learn type theory, modern algebra and proof techniques. Ultimately, engineers will have to think more like mathematicians.

Available tools, techniques, and metrics

- Larch:Unlike most formal systems, LARCH provides two levels of specification. A general high-level modeling language, and a collection of implementation dialects designed to work with specific programming languages.
- SML: Standard Meta-Language is a strongly typed functional programming language originally designed for exploring ideas in type theory. SML has become the formal methods workhorse because of its strong typing and provability features.
- HOL:HOL, short for Higher Order Logic, is an automated theorem proving system. As with most automated theorem proving systems, HOL is a computer-aided proof tool: it proves simple theorems and assists in proving more complicated statements, but is still dependent on interaction with a trained operator. HOL has been extensively used for hardware verification, the VIPER chip being a good example.
- Petri Nets:Petri Nets are a good example of a very 'light' formal specification. Originally designed for modeling communications, Petri Nets are a graphically simple model for asynchronous processes.

Relationship to other topics

- Requirements Specifications. Formal methods are a more rigorous form of the specifications systems that software engineers have been developing since the 1960's.

- Software Testing. The goal of the formal approach is the same as testing: verifying the proper behavior of a system. While formalists have sometimes claimed that formal methods can replace testing, a more realistic approach is to say that formal verification complements testing.
- Verification. Ultimately, formal methods are another verification tool. Formal verification is a way of ensuring the correctness of the theory behind the design.

Conclusions

While formal systems are attractive in theory, their practical implementations are somewhat wanting. By attempting to describe all of any system, formal methods have overreached, and generally failed. The lightweight approach, which provides a compromise between the necessities of engineering and the goals of formal design, provides a good compromise and is the most productive route for future research.

Annotated Reference List

- The single best source of formal methods papers and the like is the World Wide Web Virtual Library On Formal Methods. This page contains pointers to articles, textbooks and tools for formal design and specification.
- [Bowen93] Bowen & Stavridou: "Safety Critical Systems, Formal Methods And Standards.
  An introduction to formal methods and their role in industry, and in safety-critical systems. An excellent introductory paper which touches on most of the applications and controversy surrounding formal methods.
- [Bowen95] Bowen, Jonathan "Seven More Myths Of Formal Methods.
  This paper is a review of Hall's paper "Seven Myths Of Formal Methods", with the addition of seven other myths. Although the paper does not explicitly name it as such, the general design approach given in this paper is a good description of the lightweight approach to formal methods.
- [Easterbrook98] Easterbrook, Steve et al. Experiences Using Formal Methods for Lightweight Requirements Modeling.
  Easterbrook's paper is a good introduction to the lightweight approach to formal design and serves as a good case study of formal methods in a real environment. As such, it serves as a good companion to the Miller paper mentioned below.
- [Kling94] Kling, Robert. "Systems Safety, Normal Accidents And Social Vulnerability".

Kling's paper is actually an overview on system safety, but includes a subsection on formal verification and its role in safety.

- [Miller95] Miller & Srivas, Formal Verification of the AAMP5 Microprocessor.
  Miller & Srivas discuss the use of a formal verification system on a real project. Unlike the Wong paper, this one is more of a study of the role of a formal method in an engineering project, and takes a much higher-level view.
- [Paulson96] Paulson, Lawrence.*ML For The Working Programmer*.
  Paulson's book is an introduction to the ML programming language and functional programming in general. ML/SML figure predominantly in many formal systems, so a basic understanding of the language and the concepts of functional programming is necessary for understanding how formal tools actually work.
- [Reisig95] Reisig, Wolfgang. *A Primer in Petri Net Design.*
  Reisig's book is a textbook introduction to the use of Petri Nets. It is a good example of how a formal method is used, and the mathematical approach to design that formal methods require.
- [Wong93] Wai Wong, "Formal Verification Of VIPER's ALU"
  Wong's paper is a case study of the steps needed to formally verify the VIPER chip's ALU. The paper is really a study of formal specification in miniature, and serves as a good example of how a formal method is used.

Loose Ends

I'd like to look into the lightweight approach some more, and see if it was possible to quantify the design schemes used in a large scale project. Identifying the roles of formal methods, rapid prototyping, user sessions and the like would make for an interesting project taxonomy.

Go To Project Page