

Highly Assured Computer Engineering

Shiu-Kai Chin and Susan Older

Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, New York 13244, USA

<http://www.ecs.syr.edu/faculty/chin> <http://www.cis.syr.edu/~sueo>

Abstract

Engineering is fundamentally about the creation of systems with desired properties. Engineers strive to *calculate* properties of their implementations before actual construction begins. Our goal is to elevate professional practice in computer engineering to the point where system properties such as correctness, safety, and security are routinely verified for hardware and software systems. Achieving this goal requires devising a practical “linear-systems theory” for computer engineering, as well as incorporating into the engineering curriculum design and analysis techniques that use predicate calculus and mathematical logic. If successful, the consequences are that rigorous design assurance would be routine and complexity would be managed by a theory of composition or block-diagram algebra similar to the way linear-systems theory and system-transfer functions manage complexity in electrical engineering.

1 Introduction

Engineering is about building things that work and knowing prior to actual construction that what we build will work. In the older engineering disciplines of civil, mechanical, and electrical engineering, the capacity to *calculate* the properties of structures is well established. Statics is used to determine if a given structure of girders will support a given load. Laplace transforms are used to determine the stability of a structure of system-transfer functions. Algebra and calculus are the mathematical basis for engineering design and analysis. Thus, the disciplines of civil, mechanical, and electrical engineering all rely on rigorous mathematics, and their design and verification methods are able to rigorously relate structure to behavior.

In contrast, computer engineering has yet to reach a similar level of maturity. Current methods are unable to routinely determine or assure if a system with many concurrent and interconnected subsystems is operating correctly, safely, and securely *for all* cases of interest. This lack of assurance is particularly troubling given the pervasive use of networked and embedded systems in everything from automobile braking systems and flight controllers to medical electronics and banking. Current design practice based on simulation is merely a form of virtual prototyping. Prototyping or “build and fix” methods alone are expensive and unreliable ways for determining the presence or absence of flaws.

Elevating the level of practice in computer engineering requires design and verification methods capable of:

- Describing and verifying all cases
- Composing behavioral descriptions and checking properties of the composition
- Creating highly assured software

- Accounting for security

Also necessary for raising the level of practice are computer engineering curricula that provide a rigorous foundation for assurance. This paper summarizes our efforts to develop methods and curricula that address these needs.

The rest of this paper is organized as follows. Section 2 expands briefly each of the above points. Section 3 provides abstracts for five of our papers on assurance, which provide the background for the talk. We conclude in Section 4.

2 Improving Computer Engineering Practice

The following subsections briefly expand on the issues of verification, composition, software, security, and education. Formal mathematical logic, and its refinement to practical engineering methodology, has an important role to play in addressing each of these issues.

2.1 Describing and Verifying All Cases

The relentless increase in the numbers of transistors on a single chip has made it possible for entire systems to be implemented on a single chip. This fact effectively makes impossible exhaustive simulation as a means for design verification. However, mistakes are potentially costly. For example, some estimates of the cost of Intel's FDIV (floating point divide) flaw are up to \$480 million. Some semiconductor companies estimate that well over 50% of their development costs are associated with design verification.

Predicate calculus with recursion and induction allows us to describe and reason about all cases of interest. One of the advantages of using predicate calculus for formal verification is that designers are forced to consider all possible cases. Sometimes unanticipated cases are revealed that the designer never considered. There are straightforward methods of describing hardware designs and behavior in predicate logic. Some of these methods are discussed in [CCI⁺95], whose abstract appears in Section 3.1.

2.2 Behavioral Composition

Hardware designers are often faced with the task of faithfully implementing a user's model of behavior. These behavioral descriptions are usually at a higher level of abstraction than the implementation descriptions. For example, a specification may be given at the instruction-set architecture (ISA) level while the implementation may be described at the register-transfer level (RTL). Usually implementations make use of existing components. These components have their own behavioral descriptions, such as ISA or algorithmic state machine (ASM) descriptions. Current design and verification methods do not address the question of how to formally verify that the combined behavior of several components is equivalent to a specified overall behavior. For example, given a target instruction set, conventional verification methods do not rigorously show that a microcoded implementation using a microcode ROM, ALU, and sequencer is equivalent to the target instruction set.

Structural operational semantics can be used to define the behavior of an instruction set as a set of rules. These rules describe how outputs and memory are changed for each instruction, memory state, and input. In effect, they describe an abstract machine. [CCI⁺95] describes how higher-order logic (logic that allows functions to return functions as values and permits quantification over functions) is used to describe the operational semantics of ISA descriptions.

[CK98] (whose abstract appears in Section 3.2) describes the use of process algebra to examine the equivalence of synchronously composed algorithmic state machines and ISA descriptions to behavioral descriptions. The notion of equivalence used is *bisimulation*, which can be thought of as “plug compatibility” in hardware. Process algebras have algebraic properties that simplify design and verification. For example, if machine M_1 is equivalent to machine M_2 , then the behavior of M_1 composed with another machine M_3 is equivalent to M_2 composed with M_3 . This property is very useful for controlling complexity, particularly when M_1 is an implementation description and M_2 is a purely behavioral description.

2.3 Constructing Assured Software

Constructing assured software is difficult. Ideally, we would like to construct software systems from assured components and be able to synthesize new components in an assured manner. This view of constructing software systems by reusing trusted components is similar to the approach taken by automatic synthesis of VLSI systems and the use of parameterized macro-cell generators.

[ZKOC99] describes an experimental approach that combined formal specification and synthesis using higher-order logic with code generation using category theory. Top-level security properties and protocols were defined in higher-order logic, and the protocols were verified to satisfy the required security properties. The protocols were then instantiated by adding specific data structures and operations; the instantiations were verified to be correct in higher-order logic. The verified design specifications were then translated into a code-synthesis system based on category theory. This system refined the specifications to C++ code through stepwise refinements and composition of these refinements. The abstract of this paper is in Section 3.3.

2.4 Accounting for Security

Security is an intricate property that is achieved by a combination of sufficiently strong cryptographic algorithms and protocols, correct implementation of hardware and software, and appropriate assumptions about trusted authorities. Assuring that all these factors are present and correctly integrated to form a secure system is difficult—if not impossible—without the use of rigorous and formal analytical techniques.

[OC02b] gives an overview of some formal methods whose goal is to lend assurance that a system using cryptographic protocols will behave securely. These methods rely on mathematical logic and are accessible to engineers. These methods include the use of predicate calculus, process algebra, and modal logic (logical systems that reason about possibility and necessity). The abstract of the paper appears in Section 3.4

2.5 Education

Our Information Assurance graduate program in the Center for Systems Assurance (CSA) has earned Syracuse University a National Security Agency designation as a Center of Academic Excellence in Information Assurance Education. At the CSA, we have adopted a concept of Systems Assurance that emphasizes the ability to establish *with high confidence* that a system behaves correctly, including its availability, integrity, confidentiality, and scalability. This emphasis on high-confidence design permeates both our research and our educational efforts. CSA’s educational purpose is to prepare students to design, develop, and deploy complex systems with confidence in the systems’ ability to meet assurance requirements. With this purpose in mind, we have developed a graduate-level Certificate of Advanced Study in Systems Assurance, which is offered by the

Computer Science and Computer Engineering programs. Our goal is that the successful students of our Systems Assurance program will develop a broad background in security and information assurance, distinguishing themselves by their ability to:

- Comprehend the concepts underlying security and system assurance
- Apply those concepts to constructing assured systems
- Critically analyze and evaluate systems' conformance to their requirements

A key—and, we believe, unique—component of our program is our emphasis on using formal mathematics and logic to provide a rigorous basis for the assurance of information and information systems.

The abstract for [OC02a], which describes our experiences in incorporating mathematical logic into our educational program, appears in Section 3.5.

3 Abstracts of Papers

In this section we present abstracts from five of our papers that attempt to address the issues raised in Section 2. In presenting these papers, we by no means intend to convey that these papers are definitive solutions to the issues raised. Our intent is to share our thinking and experience to promote discussion.

3.1 On Using Predicate Calculus [CCI⁺95]

Extending VLSI Design with Higher-Order Logic

Anand Chavan, Shiu-Kai Chin, Shahid Ikram, Jang Dae Kim, and Juin-Yeu Lu

Abstract

Extending VLSI CAD with higher-order logic integrates formal verification with synthesis. The benefits of doing so are: (1) relating instruction-set descriptions to implementations, (2) designing at a higher level of abstraction than at the level of schematics, (3) verifying by proof, (4) reusing verified parameterized designs, (5) automatically compiling designs in higher-order logic to parameterized cell generators and layouts, and (6) validating electrical and functional properties by simulation. Such an integration is demonstrated by linking the Cambridge Higher-Order Logic (HOL) theorem-prover with the Mentor Graphics GDT design environment. We illustrate its application by creating a parameterized macro-cell generator for an n-bit Am2910 microprogram sequencer, whose design is formally verified with respect to its instruction-set architecture specification.

3.2 On Synchronous Composition of Instruction Sets [CK98]

An Instruction-Set Process Calculus

Shiu-Kai Chin and Jang Dae Kim

Abstract

We have created a calculus for reasoning about hardware and firmware at the algorithmic state machine (ASM) and instruction-set processor (ISP) levels of description. The calculus is a value-passing process algebra that extends the Mealy machine model to include parallel composition. It supports reasoning about the composed behavior of synchronous ASM and ISP components and microcode. We present an overview of the calculus and its application including an example showing the equivalence of a microcoded machine to its target instruction set specified by both ASM and ISP descriptions. The calculus, its properties, and the examples have been deeply embedded, proved and verified as conservative extensions to the logic of the Higher Order Logic (HOL90) theorem prover.

3.3 On Assured Software Development [ZKOC99]

Formal Development of Secure Email

Dan Zho, Joncheng Kuo, Susan Older and Shiu-Kai Chin

Abstract

Developing systems that are assured to be secure requires precise and accurate descriptions of specifications, designs, implementations, and security properties. Formal specification and verification have long been recognized as giving the highest degree of assurance. In this paper, we describe a software development process that integrates formal verification and synthesis. We demonstrate this process by developing assured sender and receiver C++ code for a secure electronic mail system, Privacy Enhanced Mail. We use higher-order logic for system-requirements specification, design specifications and design verification. We use a combination of higher-order logic and category theory and tools supporting these formalisms to refine specifications and synthesize code. Much of our work is applicable to other secure email protocols, as our development is parameterized, component-based, and reusable.

3.4 On Assuring the Security of Protocols [OC02b]

Formal Methods for Assuring Security of Protocols

Susan Older and Shiu-Kai Chin

Abstract

Establishing the security of a system is an intricate problem with subtle nuances: it requires a careful examination of the underlying assumptions, abstractions, and possible actions. Consequently, assuring that a system behaves securely is virtually impossible without the use of rigorous analytical techniques. In this article, we focus on a single cryptographic protocol (Needham-Schroeder) and show how several different formal methods can be used to identify its various vulnerabilities. The vulnerabilities include susceptibility to freshness attacks and impersonations.

3.5 On Education [OC02a]

Building a Rigorous Foundation for Assurance into Information Assurance Education

Susan Older and Shiu-Kai Chin

Abstract

Syracuse University is one of thirty-six National Security Agency designated Centers of Academic Excellence in Information Assurance Education. Our IA program was developed within the Center for Systems Assurance (CSA), whose mission is to promote improvement in systems and information assurance through research, education, and technology transfer. The goal of the CSA educational program is to develop students with a broad background in security and information assurance who distinguish themselves by their ability to (1) analyze, synthesize, and make judgments based on engineering and computer-science principles, and (2) use analytical techniques to evaluate the implications of policies, standards, and procedures; the ramifications of changes; and the potential dangers of refinements. A key—and, we believe, unique—component of our program is our emphasis on using formal mathematics and logic to provide a rigorous basis for the assurance of information and information systems. All students in our program must take a combination of courses that provide hands-on experience both in building systems and in using formal models to analyze and evaluate system behavior. In this paper, we discuss our experiences in developing and delivering a Systems Assurance program in which mathematical logic is an integral component.

4 Conclusions

How will we know when we have succeeded in raising the level of practice in computer engineering? Our thinking is that the following list will be considered routine and not unusual:

- Computer engineers will be able to formally relate descriptions and behaviors at different levels of abstraction.
- The use of mathematical logic will be pervasive in design and verification methods and tools.
- Computer engineering curricula will incorporate a rigorous and formal basis for assurance.

The collection of papers we have presented represents almost ten years of attempting to make progress on the above list. It has been hard but not intractable. Hopefully, with the recent emphasis on assurance by chip companies and by the security community, we can look forward to continued progress at an accelerated pace.

References

- [CCI⁺95] Anand Chavan, Shiu-Kai Chin, Shahid Ikram, Jang Dae Kim, and Juin Yeu Lu. Extending VLSI Design with Higher-Order Logic. In *IEEE Int. Conf. Computer Design*, Austin, Texas, October 2 – 4 1995.
- [CK98] Shiu-Kai Chin and Jang Dae Kim. An Instruction-Set Process Calculus. In *Formal Methods in Computer Aided Design (FMCAD98)*, number 1522 in Lecture Notes in Computer Science. Springer-Verlag, 1998.

- [OC02a] Susan Older and Shiu-Kai Chin. Building a Rigorous Foundation for Assurance into Information Assurance Education. *George Washington University Journal of Information Security*, 1(2), 2002.
- [OC02b] Susan Older and Shiu-Kai Chin. Formal Methods for Assuring Security of Protocols. *The Computer Journal*, 45(1):46–54, 2002.
- [ZKOC99] Dan Zhou, Joncheng Kuo, Susan Older, and Shiu-Kai Chin. Formal Development of Secure Email. In *Proceedings of the 32nd Hawaii International Conference on Systems Sciences*, January 1999.