# A Role for Formal Methodists*

Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853

November 11, 1993

Proving "correctness" of entire systems is not now feasible, nor is it likely to become feasible in the foreseeable future. Establishing that a large system satisfies a non-trivial specification requires a large proof. Without mechanical support, building or checking such a proof is not practical. Even with mechanical support, designing a large proof is at least as difficult as designing a large program. We are barely up to the task of building large and complex systems that almost work; we are certainly not up to building such systems twice — once in a programming language and once in a logic — without any flaws at all.

On the other hand, the use of formal methods for determining whether small but intricate protocols satisfy subtle properties is well documented. Many large systems are constructed from such small protocols, so formal methods certainly have a role here. Sadly, this role has been a small one. It requires identifying the protocols and constructing abstractions of the environment. Both activities require time, effort, and taste. Practitioners of formal methods are rarely in a position to build the abstractions; they lack the necessary detailed knowledge about the system. Experts on any given

*This material is based on work supported in part by the Office of Naval Research under contract N00014-91-J-1219, the National Science Foundation under Grant No. CCR-8701103, and DARPA/NSF Grant No. CCR-9014363. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of these agencies.

system are rarely in a position to apply formal methods; they lack the knowledge about the use of these methods.

Formal methods can play another role in analyzing large systems. In fact, it is one that I find quite promising. If the property to be proved can be checked automatically, then the size of the system being analyzed is irrelevant. Type checking is a classical example. Establishing that a program is type-correct is a form of theorem proving. It is a weak theorem, but nevertheless, one that is useful to have proved. More recently, we see tools to check whether interfaces in a large system are being used properly. And, in circuit design, we see model checkers to determine if a digital circuit satisfies elements of its specification. The properties to be checked are "small," the system being checked might be large, so the leverage can be great.

For large systems, though, it is unlikely that we can anticipate *a priori* all of the properties that might be of interest. Therefore, the chances are slim that the literature will contain the formal method that we seek or that a suitable tool will have been implemented. This means we must be prepared to design and use "custom" formal methods for the problem at hand. For example, in the design of the AAS system, the IBM/FAA next-generation U.S. air traffic control system, we found it necessary to avoid certain types of race conditions. This form of race was specific to the application and the design; it was not anticipated in the system requirements. Checking for these races was simple: it involved writing equations that described data dependencies for the system (they occupied about 2 pages) and implementing a program to check these equations for circularities. In short, benefits accrued from the freedom to identify and check properties on-the-fly as the design proceeded.

Generalizing, I see here a new role for formal methods in large systems. We must be prepared to design formal methods for the task at hand rather than design our systems so they can be checked with the formal method at hand. This new role has implications for how projects are staffed, how formal methods are taught, and what support tools are required.

**Staffing.** Having a "formal methodist" associated with a project allows targets of opportunity to be identified — properties of interest that can be checked mechanically. The formal methodist must be a full-fledged participant in the system project, for only then can he appreciate what properties are important to check and what assumptions are plausible in building models. Other project participants need not be practitioners of any formal method, but will still benefit from having the formal methodist working with them.

**Education.** In order to be effective, the formal methodist must be comfortable using a wide collection of formal methods. Polytheism is better than monotheism, since different formalisms are suited for different tasks. In fact, the formal methodist must be comfortable with the prospect of learning or designing a new formal method — almost at the drop of a hat. This suggests a rather different education than is traditional for today's practitioner of formal methods. Rather than producing zealots in one or another method, we must educate formal methodists who are non-partisan.

**Support Tools.** Not only must the formal methodist have access to tools that support specific formal methods, but access is also needed to tools that enable special-purpose formal methods to be designed and used. We need open systems, rather than closed, monolithic, theorem-proving environments. The formal methodist should be able to quickly cobble together software to support a new formalism: editors, checkers, printers, etc.

The time has come to admit that our vision for formal methods has been flawed. For whatever reason, programmers do not regularly employ program correctness proofs in building programs. Commercial software vendors do not perceive great benefits in writing formal specifications for their products, so they don't write them. The formal-methods "solution" to the problem of constructing high-integrity software is too often ignored.

Some believe the problem is education — today's programmers are just not comfortable with formalism. Others believe the problem is a lack of tools — there is inadequate automated support for programmers who seek to use formal methods. I believe the problem is our vision. We are too concerned with the grand challenge — provably "correct" systems — to appreciate the real and immediate opportunity: special-purpose and custom formal methods that provide leverage to the system designer or implementor.