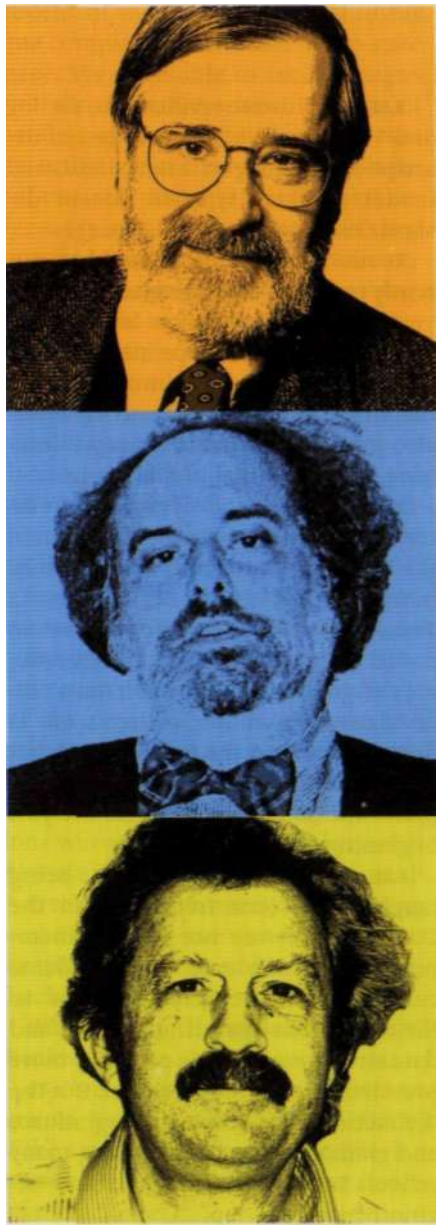
Leah Hoffman

Q&A
Talking Model-Checking Technology

A conversation with the 2007 ACM AM. Turing Award winners.

EDMUND M, CLARKE, E. Allen Emerson, and Joseph Sifakis were honored for their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries.



Let's talk about the history **of formal software verification.**

E. ALLEN EMERSON By the late 1960s, we recognized that a program should be viewed as a mathematical object. It has a syntax and semantics and formally defined behavior engendered by that syntax and semantics. The idea was to give a mathematical proof that a program met a certain correctness specification. So one would have some axioms characterizing the way the program worked for such-and-such an instruction and some inference rules, and one would construct a formal proof of the system, like philosophers do sometimes.

But it never really seemed to scale up to large programs. You ended up with something like 15-page papers proving that a half-page program was correct. It was a great idea but didn't seem to pan out in practice.

What about the history of model checking?

EDMUND M. CLARKE The birth of model checking was quite painful at times. Like most research on the boundary between theory and practice, theoreticians thought the idea was trivial, and system builders thought it was too theoretical. Researchers in formal methods were even less receptive. Research in the formal-methods community in the 1980s usually consisted of designing and verifying tricky programs with fewer than 50 lines using only pen and paper. If anyone asked how such a program worked in practice on a real computer, it would have been interpreted as an insult or perhaps simply as irrelevant.

EAE The idea behind model checking was to avoid having humans construct proofs. It turns out that many important programs, such as operating systems, have ongoing behavior and ideally run forever; they don't just start and stop. In 1977, Amir Pnueli suggested that temporal logic could be a good way to describe and reason about these programs. Now, if a program can be specified in temporal logic, then it can be realized as a finite state program—a program with just a finite number of different configurations. This suggested the idea of model checking—to check whether a finite state graph is a model of a temporal logic specification. Then one can develop efficient algorithms to check whether the temporal-logic specification is true of the state graph by searching through the state graph for certain patterns.

EMC Yes, Allen and I noticed that many concurrent programs had what we called "finite state synchronization skeletons." (Joseph Sifakis and J.P. Queille made the same observation, independently.) For example, the part of a mutual-exclusion program that handles synchronization does not depend

[CONTINUED ON P.110]

[CONTINUED FROM P.112]
on the data being exchanged in the critical sections. Many communication protocols had the same property. We decided to see if we could analyze finite-state programs by algorithmic means.

How exactly does that work?

EAE You have a program described by its text and its specification described by its text in some logic. It's either true or false that the program satisfies the specification, and one wants to determine that.

JOSEPH SIFAKIS Right. You build a mathematical model [of the program], and on this model, you check some properties, which are also mathematically specified. To check the property, you need a model-checking algorithm that takes as input the mathematical model you've constructed and then gives an answer: "yes," "no," or "I don't know." If the property is not verified, you get diagnostics.

And to formalize those specifications, those properties...

EAE What people really want is the program they desire, an inherently pre-formal notion. They have some vague idea about what sort of program they want, or perhaps they have some sort of committee that came up with an English prose description of what they want the program to do, but it's not a mathematical problem.

So one benefit of model checking is that it forces you to precisely specify your design requirements.

EMC Yes. But for many people, the most important benefit is that if the specification isn't satisfied, the model checker provides a counterexample execution trace. In other words, it provides a trace that shows you exactly how you get to an error that invalidates your specification, and often you can use that to find really subtle errors in design.

How have model-checking algorithms evolved over the years?

EMC Model-checking algorithms have evolved significantly over the past 27 years. The first algorithm for model checking, developed by Allen and myself, and independently by Queille and Sifakis, was a fixpoint algorithm, and running time increased with the square of the number of states. I doubt if it could have handled a system with a thousand states. The first implementation, the EMC Model Checker (EMC stands for "Extended Model Checker"), was based on efficient graph algorithms, developed together with Allen and Prasad Sistla, another student of mine, and achieved linear time complexity in the size of the state space. We were able to verify designs with about 40,000 states. Because of the state-explosion problem, this was not sufficient in many cases; we were still not able to handle industrial designs. My student Ken McMillan then proposed a much more powerful technique called symbolic model checking. We were able to check some examples with 10 to the one-hundredth power states (1 with a hundred zeros after it). This was a dramatic breakthrough but was still unable to handle the state-explosion problem in many cases. In the late 1990s, my group developed a technique called bounded model checking, which enabled us to find errors in many designs with 10 to the 10,000 power states.

EAE These advances document the basic contribution of model checking. For the first time, industrial designs are being verified on a routine basis. Organizations, such as IBM, Intel, Microsoft, and NASA, have key applications where model checking is useful. Moreover, there is now a large model-checking community, including model-checking users and researchers contributing to the advance of model-checking technology.

What are the limitations of model checking?

JS You have two basic problems: how to build a mathematical model of the system and then how to check a property, a requirement, on that mathematical model.

First of all, it can be very challenging to construct faithful mathematical models of complex systems. For hardware, it's relatively easy to extract mathematical models, and we've made a lot of progress. For software, the problem is quite a bit more difficult. It depends on how the software is written, but we can verify a lot of complex software. But for systems consisting of software running on hardware, we don't know how to construct faithful mathematical models for their verification.

The other limitation is in the complexity of the checking algorithm, and here we have a problem called the state-explosion problem (that Clarke referred to earlier), which means that the number of the states may go exponentially high with the number of components of the system.

EMC Software verification is a Grand Challenge. By combining model checking with static analysis techniques, it is possible to find errors but not give a correctness proof. As for the state-explosion problem, depending on the logic and model of computation, you can prove theoretically that it is inevitable. But we've developed a number of techniques to deal with it.

Such as?

EMC The most important technique is abstraction. The basic idea is that part of the program or the protocol you're verifying doesn't really have any effect on the particular properties that you're checking. So what you can do is simply eliminate those particular parts from the design.

You can also combine model checking with compositional reasoning, where you take a complex design and break it up into smaller components. Then you check those smaller components to deduce the correctness of the entire system.

How large are the programs we can currently verify with model checking?

EMC Well, first of all, there's not always a natural correspondence between a program's size and its complexity. But I would say we can often check circuits with around 10 to the 100th power states (1 with a hundred zeros after it).

JS Right. We know how to verify systems of medium complexity today—it's difficult to say but perhaps a program of around 10,000 lines. But we

don't know how to verify very complex systems.

EMC We're always playing a catch-up game; we're always behind. We've developed more powerful techniques, but it's still difficult to keep up with the advance of technology and the complexity of new systems.

Can we use model checking to check concurrent programs?

EAE Arguably, model checking is a very natural fit for parallel programming. Typically, we treat parallelism as a nondeterministic—or, informally, random—choice, so, in a way a parallel program is a more complex sequential program, with many nondeterministic behaviors. Model checking is very well suited to describing and reasoning about the associated coordination and synchronization properties of parallel programs.

EMC Concurrent programs are much more difficult to debug because it's difficult for humans to keep track of a lot of things that are happening all at once. Model checking is ideal for that.

JS But if you have programs that interact with the physical environment, time becomes very important. For these systems, verification is much more complicated.

Do we have any algorithms that can operate directly on implementable code?

EMC To verify the process of translating a design to code, or to verify the code itself, is much more difficult. Some successful model checkers use this approach, however. The Java Pathfinder model checker developed at NASA Ames generates byte code for a Java program and simulates the byte code to find errors.

JS The best available technology is proprietary technology that was developed by U.S. companies. But most of the code-level model checkers are used to verify sequential software. If you want to verify concurrent software, then you need to be very careful.

EMC The SLAM model checker developed at Microsoft Research for finding errors in Windows device drivers is probably the most successful software model checker. It is now distributed to people who want to write device drivers for Windows. However, it is hardly a general-purpose software model checker.

EAE In hardware verification, Verilog and VHDL are widely used design description languages. Many industrial model checkers typically accept designs described in these languages.

Is model checking something currently taught to undergraduates?

JS Formal verification is definitely taught in Europe. Europe has traditionally had a stronger community in formal methods, and I'd like to say it has also traditionally had a stronger community in semantics and languages.

EMC Yes, there's always been more interest in verification in Europe than in the U.S. Most of the major universities here—CMU, Stanford, UC Berkeley, U. Texas, and so on—do offer courses in model checking at both undergraduate and graduate levels, but it hasn't filtered down to schools where no one is doing research in the topic. Part of that has to do with the availability of appropriate textbooks; good books are just beginning to come out.

EAE Formal methods are being taught with some frequency [in the U.S.], but they are not broadly incorporated into the core undergraduate curriculum as required courses to the extent that operating systems and data structures are. It is probably more prevalent at the graduate level. But the distinction between undergraduate and graduate is not clear-cut. At many schools advanced undergrad and beginning grad overlap.

What's in store for model checking and formal verification?

EMC I intend to continue looking at ways of making model checking more powerful. The state explosion phenomenon is still a difficult problem. I have worked on it for 27 years and probably will continue to do so. Another thing I want to do is focus on embedded software systems in automotive and avionics applications. These programs are often safety-critical. For example, in a few years, cars will be "drive-by-wire"; there will be no mechanical linkage between the steering wheel and the tires. The software will definitely need to be verified. Fortunately, embedded software is usually somewhat simpler in structure, without complex pointers; I think it may be more amenable to model checking techniques than general software.

JS Personally, I believe we should look into techniques that allow some sort of compositional reasoning, where we infer global properties from local properties of the system, because of the inherent limitations of techniques based on the analysis of a global model. I'm working on this, as well as on theories of how to build systems out of components, component-based systems.

EAE Model checking has caused a sea change in the way we think about establishing program correctness, from proof-theoretic (deductive proof) to model-theoretic (graph search). I think we will continue to make more or less steady progress, but the pace of development of hardware and software is going to accelerate. Whether we ever catch up I don't know. Systems that are being designed are getting bigger and messier. The seat-of-the-pants approach will no longer work. We'll have to get better at doing things modularly, and we'll have to have better abstractions.

Leah Hoffman writes about science and technology from Brooklyn, NY.