2021

Formal Verification of Cyberphysical Systems

Michael, James Bret; Drusinsky, Doron; Wijesekera, Duminda

IEEE

Formal Verification of Cyberphysical Systems

James Bret Michael and Doron Drusinsky, Naval Postgraduate School
Duminda Wijesekera, George Mason University

Computer hosts a virtual roundtable with seven experts to discuss the formal specification and verification of cyberphysical systems.

In *Computer*, virtual roundtables (VRTs) are virtual panels. We ask a series of questions to a group of experts via email to ascertain the panelists' thoughts about a topic du jour. One difference between VRTs and face-to-face panels is that no expert knows who the others are. That is different from an in-person arrangement, where answers from one participant can affect the responses of others. In this VRT, our topic of discussion is the formal verification (FV) of cyberphysical systems (CPSs). FV is the act of proving the correctness of algorithms with respect to certain formal specifications, using formal methods. *Correctness* may mean logical definitions of safety, liveliness, and other objectives such as confidentially, integrity, availability, and some version of privacy.

Digital Object Identifier 10.1109/MC.2021.3055883
Date of current version: 27 August 2021

FV has its roots in formal reasoning, dating back at least to Gottfried Wilhelm Leibniz's work on algorithms, computing machines, and mathematical logic in the 17th century.[1] FV as we know it has its roots in the 1960s and 1970s with the contributions of E.W. Dijkstra, who famously coined the phrase, "Program testing can be used to show the presence of bugs, but never to show their absence."[2] Some believed FV to be a silver bullet for attaining dependable software and hardware. The excitement over formal methods is evidenced by the relatively large body of published research on the topic; a Google scholar search for the FV term yields approximately 180,000 results.

Nevertheless, except for a few well-funded research projects, industry was rather slow to adopt FV. An exception to this statement is the semiconductor design community, also known as the electronic design automation (EDA) community. This community realized that the cost and delays incurred by labor-intensive manual testing justified a different verification approach, one that applied FV. Moreover, since manual testing cannot guarantee the absence of bugs, there is an inevitable cost for containing the impact of flaws that are undetected. A classic example is the Intel Pentium FDIV bug, which was difficult for testing to uncover: containment required replacing all flawed Pentium

ROUNDTABLE PANELISTS

Knut Åkesson is a professor of automation in the Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden. His research interests include rigorous methods for verification and control with applications in safety-critical autonomous systems, the optimization and configuration of products and production systems with high variability, and applications of computer vision and deep machine learning. Åkesson received a Ph.D. from Chalmers University of Technology in 2002. Contact him at knut.akesson@chalmers.se.

Dimitra Giannakopoulou is a research computer scientist at the NASA Ames Research Center, Mountain View, California, USA, and a member of the Robust Software Engineering Group. Her research interests include applying modular and compositional formal verification techniques to autonomous systems and architectures. Giannakopoulou received a Ph.D. in computer science from Imperial College, University of London, in 1999. Contact her at dimitra.giannakopoulou@nasa.gov.

Klaus Havelund is a senior research scientist at Jet Propulsion Laboratory, Pasadena, California, USA, specifically in the Laboratory for Reliable Software. His research interests include the development of runtime monitoring techniques, including the design of powerful monitoring logics. He is the chair of the Formal Methods Europe industry committee, a member of International Federation for Information Processing 1.9/2.15 working group, and a member of the *Transactions on Foundations for Mastering Change* editorial board. Havelund received a Ph.D. in computer science from the University of Copenhagen in 1994. Contact him at klaus.havelund@jpl.nasa.gov.

Sayan Mitra is a professor of electrical and computer engineering at the University of Illinois at Urbana–Champaign, Champaign, Illinois, USA. His research interests include the formal verification and synthesis of cyberphysical and autonomous systems. Sayan received a Ph.D. in computer science from the Massachusetts Institute of Technology. Contact him at mitras@illinois.edu.

Corina Pasareanu is the technical professional leader in data science for KBR, Houston, Texas, USA. She is part of the NASA Ames Robust Software Engineering group, performing research in software engineering and is affiliated with the Carnegie Mellon University (CMU) CyLab, CMU Silicon Valley, and the CMU Department of Electrical and Computer Engineering. Pasareanu received a Ph.D. in computer science from Kansas State University. Contact her at corina.s.pasareanu@nasa.gov or pcorina@andrew.cmu.edu.

Sanjit A. Seshia is a professor in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, California, USA. His research interests include formal methods for dependable and secure computing, with application to cyberphysical systems, computer security, machine learning, and robotics. Seshia received a Ph.D. in computer science from Carnegie Mellon University. He is a Fellow of IEEE and the Association for Computing Machinery. Contact him at sseshia@eecs.berkeley.edu.

Oleg Sokolsky is a research professor in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, USA. He is member of the university's Research in Embedded Computing and Integrated Systems Center and Real-Time Systems group. His research interests include ensuring the safety of real-time and cyberphysical systems (CPSs), in addition to related areas of applying formal methods to the design and verification of CPSs, formal foundations and online monitoring for embedded systems and CPSs, hybrid systems, the automated extraction of specifications from source code, and formal methods in software engineering, particularly embedded software. He serves as *Computer*'s area editor for cyberphysical systems. Contact him at sokolsky@seas.upenn.edu.

processors on request, with Intel taking a US$475 million charge against earnings.[3,4] Fast forward to 2021, and the EDA community has embraced FV as part of mainstream development and verification processes, languages, and environments. For example, formal specification is an integral part of the SystemVerilog IEEE standard.[5]

Two of the moderators for this roundtable (Michael and Drusinsky) were authors of an article that recommended applying lightweight formal methods to the interfaces between the cyber and physical parts of a CPS.[6] This recommendation, along with our discussion of open questions in formal methods, drew a lot of interest; for instance, see the exchange between the authors and Michael Jackson.[7] The feedback we received from our

counterparts in the formal methods community, in combination with the third moderator's (Wijesekera's) experiences in applying formal methods to software-intensive systems, inspired the three of us to organize a roundtable in which we enlisted seven experts to identify the reasons for the slow adoption of FV by the software industry, in general, and the verification of CPSs, in particular.

The panelists contend there are several factors that have slowed the adoption of formal methods, such as the sheer size and complexity of software systems, the diversity of software products, the perception that FV is a low-return-on-investment academic exercise, and the fact that FV tools are not part of mainstream software development and testing environments nor are the tools directly associated with mainstream programming languages. The FV of CPSs is believed to be particularly challenging because it is a hybrid on many fronts, including hardware and software, classical control and logical reasoning, and artificial intelligence (AI)/machine learning (ML) algorithms and logical reasoning.

In this VRT, the panelists responded to six questions. Their written responses may have undergone minor edits. However, as organizers, we attempted to keep their words as verbatim as possible. The seven panelists are Knut Åkesson (Chalmers University), Dimitra Giannakopoulou (NASA), Klaus Havelund (Jet Propulsion Laboratory), Sayan Mitra (University of Illinois at Urbana–Champaign), Corina Pasareanu (KBR), Sanjit A. Seshia (University of California, Berkeley), and Oleg Sokolsky (University of Pennsylvania). See "Roundtable Panelists" for the participants' biographical sketches. Note that the opinions of the experts are their own, with no input from the editors. We hope readers who are concerned with the dependability and trustworthiness of CPSs will find the questions and responses enlightening.

***COMPUTER*:** Unlike with EDA, in which FV is well integrated into the development of chips and printed circuit boards and where engineers with expertise in the method are in high demand, FV has had much less acceptance as a mainstream ingredient of software development and quality assurance. What do you think are the reasons for that, and do you think the situation will change with CPS projects, such as those involving autonomous vehicles?

KNUT ÅKESSON: A major challenge is that the closed-loop model is described using a combination of tools, different modeling languages, and programming languages. Significant efforts have been made to unify how to describe physical systems coherently. For example, the Modelica language (https://modelica.org/modelicalanguage.html) is an essential step in this direction. However, CPSs might also contain ML algorithms for perception and might run optimization for decision making. These are all rapidly evolving and have their dedicated languages and tools. Thus, CPSs inherently combine code written in various programming languages, ML frameworks, optimization modules, and control logic generated from high-level modeling languages. FV has its place in safety-critical components but should be complemented by rigorous automated test methods for situations where it is not feasible or practical to use.

A major challenge is that the closed-loop model is described using a combination of tools, different modeling languages, and programming languages.

DIMITRA GIANNAKOPOULOU: Software development is more diverse and evolves faster than EDA in terms of programming languages, paradigms, and patterns; data structures, algorithmic approaches, and types of applications; libraries and runtime environments; and heterogeneity and distribution across different computers. After deployment, software applications get updated to address vulnerabilities and to include new features, and they may even be adaptive by design. Correctness criteria and specifications vary widely by application domain, and quality assurance depends on the criticality of software. For example, is it a game on someone's phone, or is it software that controls a passenger aircraft?

For FV to become a mainstream ingredient of software development, it must achieve some usability goals. First, it must be relatively easy to formulate specifications for the target system. Second, FV must be able to directly handle the languages in which the software is written or the modeling languages from which the software is synthesized. Finally, FV should be able to scale. The diversity and complexity of software applications means that to be successful, FV approaches must be targeted and customized to address specific problems within safety-critical application domains.

The expected exponential rate of introduction of autonomous vehicles (ground and air) puts enormous pressure on ensuring their safe operation. There is incentive for commercial and federal stakeholders to collaborate on developing certification and assurance standards for these applications. As a consequence, I expect advances in the near future. On the one hand, FV approaches will be developed that efficiently address specific problems of such CPSs. On the other hand, there will be increased incentives in CPS projects to use programming paradigms and environments designed with FV in mind.

KLAUS HAVELUND: Electronics designs have the characteristic that once

they leave the factory, they usually cannot be changed. A substantial error can cause a unit to be recalled, with large amounts of money at stake. The motivation is therefore high to "get it right" before shipment. Software, on the other hand, can often be fixed with an update at a customer's location, making errors less catastrophic. Even in space missions, errors can be corrected by uplinking bug fixes from a distance of millions of miles. This relaxed view of software errors might, however, be changing as software, to an increasing extent, autonomously controls equipment such as cars, which can cause loss of life in case of failure.

Another, perhaps more important, reason for the lesser acceptance of FV in the software community is that the verification problem appears less tractable for software systems, due to higher complexity and the possibility of more execution paths. Theorem provers require a considerable amount of manual effort to apply, even for smaller models, let alone real-world software systems, and model checkers are challenged by the large state spaces of realistically sized software applications. This means that the application of FV techniques requires either a big verification effort or a big modeling effort, where a simplistic model is created of the software and then verified. A software engineer, not supported by management to carry out such proofs/modeling, will see very little incentive to do so.

SAYAN MITRA: FV is being used in mainstream software already, propelled first by major outages and breaches at big tech firms, then by successful applications of verification technology in bug finding, and more recently in the application of verification for generating proofs as "more extensive tests." Static analysis tools are part of the core developer workflow at Google and deployed on the 2-billion-line code base.[8] Amazon Web Services (AWS) developers are writing formal specifications and proofs for hypervisors, boot loaders, and Internet of Things operating systems.[9] The Infer static analysis engine is integrated with the code base at Facebook and does continuous reasoning on iOS, Android, and Instagram and WhatsApp applications. Hundreds of bugs are reported and fixed every month.[10] Bugs in CPSs and autonomous systems can compromise safety. This raises the stakes as well as the incentives for the adoption of FV. But the adoption of CPS verification also presents barriers that were not present in the software ecosystem.

CORINA PASAREANU: The reason is that FV for software is much harder (for example, programs are much larger, potentially unbounded, use many external libraries, and contain programming language constructs that are hard to analyze). Yes, CPS projects are often safety critical and justify the high cost of FV. Furthermore, the software involved in CPS projects is simpler than general-purpose software and therefore more amenable to verification.

SANJIT A. SESHIA: There is a spectrum of FV methods, from assertion-based testing and model-based testing to static analysis and model checking and interactive theorem proving. So, if we define FV broadly to include this entire spectrum, I contend that FV is already used widely in software in much the same way as it is employed in hardware. Of course, software comes in many different flavors, and so we will find FV used more for software in safety-critical, mission-critical, and high-availability applications. FV is also used in certain industrial CPS applications; for example, the simulation-based falsification of temporal logic (TL) properties has been successfully applied in the automotive industry.[11] Over the past decade, I have seen big growth in interest from the CPS industry in applying formal methods to CPS design, although that interest has yet to fully translate into a wider deployment of tools.

OLEG SOKOLSKY: The main reason is that the software verification problem is inherently much harder. Software tends to be much less structured and much more complex compared to hardware. Finite-state models, which are much easier to verify than infinite-state ones, are a more natural fit for hardware than for software. From this perspective, CPSs are likely to make verification problems only harder. Embedded processors are becoming ever more powerful, enabling more and more complex software on board. In addition to software, physical environments need to be included in the model, making the challenge even bigger. If there is any silver lining, modern CPSs—in particular, autonomous vehicles—offer more room for lighter-weight applications of FV. Runtime verification techniques, that is, formally specified monitoring and adaptation, as well as applications of online reachability computation, appear to be very promising in autonomous CPSs.

***COMPUTER*:** An often neglected issue related to FV is the reliance of most techniques on expressively weak and hard-to-use formal specification languages (in the sense of creating correct specifications), such as dialects of TL. How serious do you think this problem is, and how can it be addressed?

ÅKESSON: For maintenance reasons, it is important to ensure that specifications and implementations are closely linked. Specifications have to be understandable by the engineer doing the implementation, and they have to be refined during the implementation phases. It should also be possible for the same engineer to update them. In our experience working with industrial partners, writing correct specifications is challenging, and it is often the case that an identified violation of a specification is due to a mistake in the formalization of the specification and not in the implementation. While FV tools have a well-defined specification

language, it might be useful to consider high-level, domain-specific specification languages that integrate well with the implementation language and to consider automatically translating from this domain-specific language to the FV specification language being used.

GIANNAKOPOULOU: Creating specifications is typically an exploratory process aimed at nailing down the intended behavior of a target system, avoiding overspecification, underspecification, and ambiguity. What FV requires is a lack of ambiguity and a formal language to communicate with. In terms of ambiguity, even a simple sentence containing a condition under which some system behavior is expected has many possible interpretations. Figuring out the interpretations and picking the intended one is not straightforward. Writing a formal specification that is precise with respect to the intended interpretation is even harder. In my experience, nontrivial specifications are challenging even for experts. A way to address this problem is to build environments that assist in the process of gradually constructing specifications that are unambiguous and capture user intentions. Such environments would ideally enable users to write and explore their specifications through a variety of approaches: natural language, diagrams, use case scenarios, and interactive simulation. Formal specifications should then be produced automatically and through trusted algorithms. The problem of producing specifications can also be alleviated through the support of domain-specific specification patterns. Even in this case, however, it is crucial to provide a user-friendly environment for exploring and understanding the details of such patterns.

HAVELUND: Two problems are mentioned here: expressively weak specification languages and hard-to-use specification languages. I think the second problem, with hard-to-use specification languages, might be a nonissue. Just consider the complexity of C++, which programmers happily learn. Specification languages are no harder to learn, and in many cases, they are simpler than programming languages. Programmers have no difficulty writing the programs, so they can probably write specifications, as well. Some of the more simplistic languages (such as linear TL) can be hard to use for writing more complex properties, but there are solutions to that, such as specification patterns and graphical solutions, potentially translated into the harder-to-use formalisms. The real problem, in my view, might not be the difficulty of learning a specification language but the lack of willingness among developers to deal with another complex language in addition to the programming language. There is an argument for developing specifications in the programming language itself. Specification languages must be highly expressive to meet practical needs. I have developed numerous specification languages for software monitoring, and it is usually the languages that support an escape to a general-purpose programming language (when the logic formalism falls short) that appear most attractive to users.

MITRA: Verification tools must communicate with developers using artifacts and interfaces that are already part of their workflow. Requiring developers to learn a new language or a formalism is a nonstarter. Chong et al. discuss a four-year experience in which the loss of expressive power (or not using TLs, for example) was more than offset by the benefits of using the same programming language for coding and specifications.[9] This is a recurring theme at other firms adopting FV. Using common artifacts and interfaces reduces the "developer's cognitive burden and allows them to view proofs as 'just another test suite,' albeit a vastly more thorough one."[9] The integration of development and verification workflows was also a precursor to the success of hardware verification through description languages such as VHDL and Verilog.

One challenge for CPS verification is that existing tools—of which there are many strong ones—rely on mathematical models that are disconnected from developer workflows. There are no open and standard CPS languages and development ecosystems for plugging in verification tools. MATLAB is popular but, unfortunately, neither open nor standardized. The solution is to move away from model verification tools to tools that verify CPS code written in open languages, such as C, C++, and Rust, and testing and verification environments that use open simulators, including CARLA (https://carla.org/) and Gazebo (http://gazebosim.org/).

One challenge for CPS verification is that existing tools—of which there are many strong ones—rely on mathematical models that are disconnected from developer workflows.

Second, some CPS components have to be treated as black boxes. The code for a component may be too complex, and it may be proprietary. The physical models may be impossible to represent as $\dot{x} = f(x)$ or as a hybrid automaton. For such black-box components, verification has to rely on statistical methods. We will need to integrate verification approaches that can combine black-box methods with model-based techniques within the development ecosystem. One approach in this direction is discussed in our DryVR framework, which has been used to verify several industrial-scale systems that combine black- and white-box components.[12,13] TLs have been fundamental in understanding the complexity of

verification and synthesis problems with respect to different specification classes. Extrapolating those scientific advances to a world in which developers learn TLs and start using them as specification languages for day-to-day development, in my view, is not realistic.

PASAREANU: Formal specification languages are hard to understand even for an expert in formal methods. Natural language representations, patterns, and tool support can perhaps address the problem.

SESHIA: I think we can learn a lot from hardware verification. TL-based assertion languages are now widely used in hardware design, and yet the average developer does not need to be an expert in logic to use them. They have been incorporated into more accessible assertion languages, integrated with user interfaces, and generated by tools for the automated inference of specifications. In fact, tools for specification mining, learning properties from execution and simulation traces, are a very promising approach for easing the specification burden. In our own work with industry, we have seen that a specification mining tool can ease the initial burden of writing TL properties, which demonstrates to industrial users the value of formal specifications, becoming a virtuous cycle where users actively seek to learn to write logic properties due to the added value it brings them.[14] Specifications can also be integrated as "blocks" into tools that industrial users already employ; for instance, see Kapinski et al.[15]

SOKOLSKY: There are two related problems here. One is that, indeed, formal specification languages are hard for engineers to fully understand and use effectively. To a large extent, this drives the need for formal methods experts and stands in the way of transferring verification technology to engineers. The other problem is that, as specifications become more complex and harder to grasp, they become increasingly error prone themselves. Both issues can be partially addressed with better specification languages and tool support.

***COMPUTER*:** There is a perception that human involvement in the creation of formal specifications limits our ability to apply FV to CPSs. ML-based specifications are limited, at present. Can specifications created by ML algorithms be trusted? In other words, who will guard the guard (the first guard being ML-created formal specifications used for FV)?

> I believe we have quite a bit of work to do before we can trust ML to produce correct specifications, especially if we are liable for them.

ÅKESSON: Writing high-quality specifications is a very challenging task for both humans and computers. But algorithms (AI, ML, and others) can play an important role in assisting humans by proposing specifications and suggesting possible extensions. I believe that the process of formalizing specifications is as important as the verification process. During this, assumptions have to be expressed explicitly, and it has to be defined what the expected behavior should be for all corner cases. These insights are lost if ML is used to generate specifications. Thus, I see that the primary role for ML is in assisting humans by helping with the process of identifying untold assumptions and corner cases.

GIANNAKOPOULOU: Specification mining is not a new idea. In fact, several approaches have been developed that try to bypass human involvement in the creation of formal specifications. Naturally, ML is also involved in this quest. After all, it is, by now, involved in every aspect of software engineering. In my experience, ML is extremely tricky to get right, as it relies on the amount and quality of available training data and may not transfer well to other domains. One avenue that is being explored toward increasing trust is to develop ML frameworks that explain their decisions. In general, I believe we have quite a bit of work to do before we can trust ML to produce correct specifications, especially if we are liable for them. On the other hand, ML could be a valuable aid for CPS designers toward discovering, formulating, and repairing specifications.

HAVELUND: Specifications generated by ML techniques will undoubtedly become increasingly important. Just from a philosophical point of view, it is an evident trend. It is, however, nearly impossible to predict how much such systems can be trusted. They will, for sure, play advisory roles and eventually safety-critical ones. The most obvious approach to deal with such systems, in my view, is to monitor their execution and ensure that they behave within a more traditionally defined safety region. Hence, the guards of the guards are monitors. ML can also be used to propose formal specifications to be approved by humans.

MITRA: When specification writing becomes part of the development process, with tangible benefits, and it is no longer seen as an isolated activity, then the creation of specifications may not be viewed as burdensome. One study reports that AWS developers spend considerable energy writing proof harnesses, which are essentially assertions written in the programming language and that guide the verification engine and provide much better coverage.[9] ML-created specifications are an intriguing idea. Obviously, generating

labeled data for any such approach will still require curation and expertise.

PASAREANU: I believe there will always be some human involvement and domain expertise in the creation of formal specifications. I am not sure what you have in mind with "ML-based specifications." If these are specifications mined from data and/or systems, then I think a human expert can validate them. FV tools can be used to formally verify them.

SESHIA: As I mentioned earlier, learning specifications from data and other artifacts is a promising approach to ease the specification burden. One way to generate trust in ML-created formal specifications is to validate them against available code and models, with human oversight. This is exactly the approach we took in a collaboration with Toyota, where, when an engineer felt our generated specification was incorrect, the validation pointed to a corner case bug in a large Simulink model the company was analyzing.[14] In other words, mining specifications and finding corner case bugs are two sides of the same coin. This specification mining approach is a special case of a more general methodology for high-assurance ML termed *oracle-guided learning* or *oracle-guided inductive synthesis*.[16,17]

SOKOLSKY: On the one hand, we clearly need a way to keep tabs on machine-generated specifications, to make sure they capture our intuitive goals and that there are no unintended aspects. On the other hand, we must remember that human-created specifications are not perfect, either. Thus, the question is not whether we should trust machine-generated specifications more or less than ones crafted by humans. Whatever the source, we should be able to perform sanity checks on a specification or, better yet, verify it with respect to higher-level requirements.

***COMPUTER*:** How much of the verification of a CPS is physics, and how much is logic and traditional reasoning tools? Where do you think this ratio is headed? Similarly, how much is logical inference versus statistical inference? How much of ML algorithms can translate into traditional reasoning, and what is lost in the process?

ÅKESSON: Physics plays a vital role in restricting the behavior of a closed-loop system. However, it is the perception and decision-making code that is rapidly increasing in complexity.

GIANNAKOPOULOU: Instead of commenting on the ratio of physics to logic, I will share some observations. In my experience, many novel algorithms for autonomous decision making (collision avoidance, for example) are constructed using models (often probabilistic) of the physical systems involved. Finding the right level of model abstraction to combine scalability with safety is an art. Ensuring the conformance of physical models to the real world is key when verifying CPSs. The need to deal with uncertainty and optimization, which are intrinsic in autonomy, creates a natural shift toward statistical inference. In my opinion, the major challenge with reasoning about ML algorithms is that their logic is not explicit, making it hard to formulate and assess the correctness of their behavior.

HAVELUND: As long as there is traditional software in CPSs, it will need to be verified and tested. Furthermore, such systems will increase in complexity, meaning even more software to be verified and tested. A big part of such future systems will therefore be traditional testing and logic-based reasoning tools to the extent that they scale to the problem.

MITRA: The physics-to-logic ratio in CPS verification evolves across development stages. As physical processes become better understood and controlled, design and verification complexity shifts to the computing stack, with the goals of achieving better efficiency, less energy use, and utilization. The early adopted methods are usually the ones that are stable and easier to interpret. My view is that the early adoption of CPS verification will be dominated by the more traditional proofs, logical inference, and absolute guarantees, while statistical approaches will dominate testing. For end-to-end and system-level verification, the verification results of heterogenous components have to be composed. There are very interesting ideas about incorporating ML in verification, particularly for handling black-box components we mentioned earlier, but these approaches are still in their infancy.

PASAREANU: I think it is hard to quantify. It seems, indeed, that we have a bit of all of them.

SESHIA: Your first question goes to the crux of how CPSs are defined. According to Edward Lee and myself, CPSs are integrations of computation with physical processes whose behavior is defined by both cyber and physical components.[18] Thus, every CPS verification problem involves reasoning about the "physics" and reasoning about computation. Now, to achieve scalability, we typically must take a modular approach, where we break up the CPS verification problem into several subproblems, some purely cyber, some purely physical, and some cyber-physical. With respect to your second question, I think inductive learning, also known as *ML*, is central to the process of proof. The combination of inductive and deductive reasoning has been at the heart of many advances in FV over the past 20 years, including counterexample-guided abstraction refinement and techniques for invariant synthesis, where inductive learning is combined with deductive reasoning by using hypotheses about the structure of proof artifacts being synthesized.[16] So, ML algorithms do fit in a natural way into "traditional" reasoning. It remains to be seen how

useful deep learning, specifically, will be in FV.

SOKOLSKY: It seems hard to separate the effects of physics and logic in CPS verification challenges. While physics verification seems harder, or at least less scalable than logic verification, it is the interaction between physics and logic that makes CPS verification so difficult. The balance between logical and statistical inference depends on the verification approach, with statistical inference becoming ever more prominent in recent years.

***COMPUTER*:** Should the FV of a CPS be conducted on the interface between the cyber and physical partitions instead of directly on them?

> Integration as an afterthought usually results in expensive redesigns and modifications late in the software development life cycle.

ÅKESSON: There is a need to do both. During early development phases, the components and their interfaces are defined, and the implementation and models might be missing or incomplete. During these phases, the interfaces' expectations and guarantees toward the environment can be defined and verified. Later in the development process, FV and other rigorous test methods, such as falsification, should be used to verify the closed-loop behavior.

GIANNAKOPOULOU: This falls under the standard topic of unit versus integration testing/verification. The answer is that it should be conducted at all levels. However, given the complexity of CPSs, it is worthwhile to invest in studying the interface between the cyber and physical partitions first. Understanding and specifying the intended interactions between the two provides a solid foundation for developing systems that will integrate seamlessly. Integration as an afterthought usually results in expensive redesigns and modifications late in the software development life cycle.

HAVELUND: As I pointed out, I think a large part of the verification of CPSs will still be the validation of traditional code bases. However, specifically monitoring techniques, also referred to as *runtime verification*, can be used to oversee the interface between the software and the physical system and potentially prevent the software from doing any harm, a subfield of runtime verification referred to as *runtime enforcement*. Here, the monitor will prevent the software from issuing harmful commands to the physical system.

MITRA: Carefully defining CPS model interfaces can help achieve a separation of concerns, for example, farming out the physics models or components and the software elements to different proof engines in such a way that their results can be soundly combined to verify the overall model. Our Koord[19] language and the CyPhyHouse[20] verification framework are tailored to address this issue in the context of distributed CPSs written using shared memory.

PASAREANU: Perhaps on both. Compositional reasoning can be helpful in putting together results from separate verifications.

SESHIA: Since CPSs are fundamentally about the intersection between cyber and physical worlds, some verification will always need to be on the interface between the two. For compositional analysis, some verification may need to be on individual cyber and physical "partitions." But the overall proof will always involve the interface. And if a counterexample is to be demonstrated, it must be a full CPS counterexample. Our experience working with industrial users in the automotive sector is that, first, integration testing is the biggest challenge, and second, people care much more about system-level counterexamples than "unit" counterexamples; for instance, see the work of Yamaguchi et al.[11]

SOKOLSKY: I try to avoid being prescriptive in the choice of verification approaches. Whatever works should be used. I would imagine that interface-based techniques may offer better scalability, in general, at the expense of more significant conservatism. A lot depends on the system design, and the verification engineer should be prepared to apply the whole range of available tools as needed.

***COMPUTER*:** Of the current impediments—technical or otherwise—that make it challenging to formally verify a CPS in an effective and efficient manner, which do you think is the most pressing to address and why?

ÅKESSON: Scalability and ease of use are limiting the industrial acceptance of the FV of CPSs. The limitations of formal and rigorous verification methods signify the importance of a modular approach, such as combining ML components with correct-by-construction approaches and software modules with manageable complexity. A significant challenge is combining the white-box approaches of FV with the black-box methods used in falsification to handle systems where parts are fully known while for others, only incomplete information is available.

GIANNAKOPOULOU: Regulatory bodies are pressed to come up with solutions for ensuring the safety of autonomous vehicles, which are expected to invade our lives in massive numbers in the near future. It is a great opportunity to exploit this pull for techniques that ensure trust in autonomy. In many respects, CPSs share

verification challenges with traditional large, complex distributed systems and can benefit from advances made in those domains. However, they place increased emphasis on AI. Within that domain, I believe it is most pressing to identify and formulate requirements for the correctness of adaptive and ML algorithms.

HAVELUND: The main problem, in my view, is the algorithmic challenge in verifying large systems. We are currently not able to automate this process sufficiently to make it broadly attractive. To this can be added the problem of writing specifications. However, I do believe that if the verification problem could be solved (highly automated) and if specifications really captured the details of interest (requiring expressive specification languages), there could be enough motivation for adopting FV. This is not to underestimate the problem of writing specifications. There is a need to support the formal specification and verification of programs written in programming languages and perhaps with specifications written in the programming language itself, for example, much like unit tests. Some programming languages are now being developed with built-in support for FV. The guaranteed short-term-winner approach is automated testing 24/7, in which a system is constantly bombarded with inputs and monitored as it executes with advanced test oracles. This requires trustworthy simulators of the physical systems, which can be rerun repeatedly on a normal desktop or laptop.

MITRA: We need a standardized, open development ecosystem for CPSs and related benchmarks. Open standards help identify problem definitions and attract talented researchers. They reduce friction in sharing solutions. Benchmarks and standards also help practitioners share hard instances across domains, and they give a yardstick for the community to measure progress.

PASAREANU: CPSs are increasingly built using ML components, such as neural networks, which are hard to specify and verify formally. I view that as the main challenge.

SESHIA: In a sense, the CPS verification challenge is the union of the difficulties of verifying hardware, software, and physical systems because CPSs integrate all of them. It is difficult to identify a single challenge that is the "most pressing." My top contenders include modeling the complex environments of CPSs, developing better theories of compositional reasoning for CPSs, verifying intelligent CPSs based on AI and ML, and creating a large and diverse repository of benchmarks to guide the community.

> Regulatory bodies are pressed to come up with solutions for ensuring the safety of autonomous vehicles, which are expected to invade our lives in massive numbers.

SOKOLSKY: A lot of challenges to FV, such as the computational complexity of verification algorithms and the rapidly growing scale of CPSs, are fundamental and thus cannot be really addressed, in my opinion. What can be addressed is the verifiability of CPSs. Systems can and should be designed in a way that makes them easier to verify, more modular, and better structured. To achieve that, we need better design approaches and techniques. But even more importantly, we need to change the mindset of designers. Most system designers are not experts in formal methods and do not need to be. But they need a better understanding, if only at a rule-of-thumb level, of what makes a system easier or harder to verify.

There is consensus among the panelists that the software industry is, indeed, slow to adopt FV, except for static analysis—which is arguably more of a compiler technology than FV—and some projects run by deep-pocket companies. The reasons include software's complexity, rate of change, and diverse correctness criteria. A key obstacle cited by multiple experts is the FV environment and ease of use. In contrast with the EDA market, in which FV is a first-class member of the development environment and tool chain, for software developers, FV is like a distant "nerd" cousin that speaks a different dialect and one that few first-class members pay attention to.

The panel members agree that as difficult as it is to successfully apply FV to software in general, it is as difficult or more so to apply it to CPSs. Some argue that the complexity, scale, and opaque nature of ML algorithms make the full application of FV for CPSs unrealistic. However, limited approaches, such as runtime verification (especially on the interface between the cyber and physical partitions of a CPS) can be used. The expectation that, inevitably, some correctness properties themselves will be machine learned only exacerbates the trust problem. Nevertheless, despite such mounting challenges, we recommend that the FV research community measure up by developing techniques for dealing with the difficult nature of building dependable CPSs. As an analogy, consider the various techniques theoretical computer scientists have developed for coping with intractable (NP-complete) problems, such as heuristics, Horn logic, and Boolean satisfiability-solving algorithms. Indeed, runtime verification is one such approach.

REFERENCES

1. M. B. W. Tent, ed., *Gottfried Wilhelm Leibniz: The Polymath Who Brought Us Calculus*, 1st ed. Boca Raton, FL: CRC Press, 2011.
2. E. W. Dijkstra, "On the reliability of mechanisms," in *Notes on Structured Programming, T. H.-Report 70-WSK-03*, 2nd ed. Eindhoven, The Netherlands: ersity, Apr. 1970, p. 7. Accessed: June 1, 2021. [Online]. Available: https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF
3. A. Edelman, "The mathematics of the Pentium division bug," *SIAM Rev.*, vol. 39, no. 1, pp. 54–67, 1997. doi: 10.1137/S0036144595293959
4. "Intel takes $475-million earnings hit: Computers: The charge for replacing flawed Pentium chips mars an otherwise stellar year," Los Angeles Times, Jan. 18, 1995. Available: https://www.latimes.com/archives/la-xpm-1995-01-18-fi-21424-story.html
5. *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Standard 1800-2017 (Revision of IEEE Standard 1800-2012), Feb. 22, 2018. doi: 10.1109/IEEESTD.2018.8299595.
6. J. B. Michael, G. W. Dinolt, and D. Drusinsky, "Open questions in formal methods," *Computer*, vol. 53, no. 5, pp. 81–84, 2020. doi: 10.1109/MC.2020.2978567.
7. "Letters: Another view on formal methods," *Computer*, vol. 53, no. 9, p. 8, 2020. doi: 10.1109/MC.2020.3001958.
8. C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspa, "Lessons from building static analysis tools at Google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, 2018. doi: 10.1145/3188720.
9. N. Chong et al., "Code-level model checking in the software development workflow," in *Proc. ACM/IEEE 42nd Int. Conf. Softw Eng.: Softw. Eng. Pract.*, 2020, pp. 11–20. doi: 10.1145/3377813.3381347.
10. P. W. O'Hearn, "Continuous reasoning: Scaling the impact of formal methods," in *Proc. 33rd Annu. ACM/IEEE Symp. Logic Comput Sci.*, 2018, pp. 13–25. doi: 10.1145/3209108.3209109.
11. T. Yamaguchi, T. Kaga, A. Donze, and S. A. Seshia, "Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems," in *Proc. IEEE Int. Conf. on Formal Methods Computer-Aided Design*, Oct. 2016, pp. 201–204. doi: 10.1109/FMCAD.2016.7886680.
12. C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "DryVR: Data-driven verification and compositional reasoning for automotive systems," in *Computer Aided Verification* (Lecture Notes in Computer Science), R. Majumdar and V. Kunčak, Eds. Berlin: Springer, pp. 441–461. 2017. doi: 10.1007/978-3-319-63387-9_22.
13. S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. Cambridge, MA: The MIT Press, 2021. ISBN-13: 978-0262044806.
14. X. Jin, A. Donze, J. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1704–1717, 2015. doi: 10.1109/TCAD.2015.2421907.
15. J. Kapinski et al., "ST-Lib: A library for specifying and classifying model behaviors," SAE Tech. Paper 2016-01-0621, 2016. doi: 10.4271/2016-01-0621.
16. S. A. Seshia, "Combining induction, deduction, and structure for verification and synthesis," *Proc. IEEE*, vol. 103, no. 11, pp. 2036–2051, 2015. doi: 10.1109/JPROC.2015.2471838.
17. S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," July 2016. [Online]. Available: https://arxiv.org/abs/1606.08514
18. E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA: The MIT Press, 2017.
19. R. Ghosh, C. Hsieh, S. Misailovic, and S. Mitra, "Koord: A language for programming and verifying distributed robotics application," *Proc. ACM Program. Lang.*, vol. 4, pp. 1–30, 2020. doi: 10.1145/3428300.
20. R. Ghosh et al., "CyPhyHouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination," in *Proc. IEEE Int. Conf. Robotics Automat.*, 2020, pp. 6654–6660. doi: 10.1109/ICRA40945.2020.9196513.

DISCLAIMER

The views and conclusions contained herein are those of the panelists and moderators and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of their employers. The U.S. government is authorized to reproduce and distribute reprints for government purposes, notwithstanding any copyright annotations thereon.

JAMES BRET MICHAEL is a professor in the Department of Computer Science and the Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, 93943, USA. Contact him at bmichael@nps.edu.

DORON DRUSINSKY is a professor in the Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943, USA, and the chief science officer at Aerendir, Mountain View, California, 94040, USA. Contact him at ddrusins@nps.edu.

DUMINDA WIJESEKERA is a professor in the Department of Cyber Security Engineering, George Mason University, Fairfax, Virginia, 22030, USA, where he is codirector of the Center for Assured Research. Contact him at dwijesek@gmu.edu.