

# Formal Methods in Cyberphysical Systems

**James Bret Michael and Doron Drusinsky,**  
Naval Postgraduate School

**Duminda Wijesekera,**  
George Mason University

*To improve the state-of-the-art practice of applying formal methods to cyberphysical systems, we briefly discuss the evolution of these methods and also summarize four research efforts to close the current capability gaps in their application.*



The integration of computer systems into the interfaces of physical devices and social activities, such as the Internet of Things and Facebook, has grown by leaps and bounds and is now pervasive. However, there are gaps in our ability to construct cyberphysical systems (CPSs) with the built-in design and implementation to provide the assurance that systems will behave correctly under normal and abnormal conditions. A fundamental gap is that the definition of correct behavior is lacking. An example of a gap that impacts the practice of using formal methods in the development of CPSs is that existing methodologies and tools seem to be inadequate for representing and reasoning about the correct behavior of CPSs. Our own experience has been that the tools can also be difficult to use and are limited in their support for providing assurances that a model is not oversimplified, is consistent with physical principles, and is implemented as specified.

Formal methodologists, be they academics or practitioners, are interested in advancing the body of knowledge and practice of

- › integrating formal methods into lifecycle processes and toolchains
- › lowering the time to market and cost of using formal methods in the development of CPSs.

Formal methods entail the use of mathematical rigor to model and reason about systems. Formal methods have been used to reason about program correctness since the early days of programming languages. Early methods hierarchically decomposed system requirements and a corresponding software program into segments and collected the pre- and post-behavior of executing each segment into a single logic statement asserted to be implied by one or more requirements. Rules from mathematical logic or proof rules created for each construct of the language are utilized to prove such an implication.<sup>1</sup> Hand-proved correctness arguments were extended to use automata-theoretic models capable of capturing a system's evolution as a series of discrete state changes and then reasoning about those changes. Automata-theoretic models were extended to discrete, time-dependent concurrent systems, allowing rigorous mathematical proofs of properties concerning concurrency and timing-related specifications.<sup>4</sup> In addition, modal logics, such as temporal<sup>2</sup> and dynamic logics,<sup>3</sup> were introduced to support specifications written for shared-memory concurrent systems and reactive systems.

During the 1980s and early 1990s, automata-theoretic techniques were extended to model discrete control of systems that evolved based on continuous physics-based systems. This area of formal methods became known

as *hybrid systems modeling*.<sup>5</sup> In these formalisms, the state transitions of a hybrid system are governed by differential equations.

A CPS requires physics-based models (for example, nonlinear equations of motion to represent the motion of a vehicle and measure the accelerometer's sensed change in impedance). A CPS converts analog sensor data into digital data that are input to the CPS controllers. Automata-theoretic models are needed to represent the storage, transmission, and processing of the digital sensor data by the controller. The outputs of the controller are converted back to analog signals for use by physical components of the CPS, such as actuators (for example, a motor that adjusts the steering angle of a self-driving vehicle). The sensors and actuators have their own faults and vulnerabilities (captured by fault models and attack surfaces), and actuators have natural limitations based on their design. Some of these faults or limitations may be based on the operational contexts and may be affected by naturally occurring faults or human-created malevolent acts (either physical or cyber). CPSs are expected to adhere to specified behavioral constraints during their evolution while satisfying specifications. Consequently, applying formal methods to a CPS involves proving that the system evolves and traverses a trajectory that satisfies the stated constraints and requirements. An example is the driving automation of an autonomous vehicle, for which there are constraints related to stability, such as maintaining the vehicle's yaw, roll, and pitch within acceptable limits. A specification, on the other hand, would be to maintain a safe braking

- › modeling the interface between the cyber and physical partitions of systems
- › applying lightweight formal methods for verification and validation of CPSs
- › modeling uncertainty in CPSs and the environments in which they operate
- › managing levels of abstraction in modeling CPSs
- › applying compositional formal methods to composed CPSs

distance (which depends on the road geometry, surface friction, and the vehicle's speed profile) from static (like roadside barriers) and dynamic (like other vehicles) obstacles.<sup>13</sup>

Naturally, CPSs have been modeled as discrete, automata-theoretic, hybrid systems and/or their communicating counterparts. Consequently, many of the formal methods listed previously have been used to verify that CPSs satisfy stated constraints and formally stated requirements.

## IN THIS ISSUE

We thank everyone who submitted a manuscript for the special issue. In this issue you will find four of the 12 submissions in addition to a virtual roundtable on formal verification of CPSs.

In "Interpretable Fault Diagnosis for Cyberphysical Systems: A Learning Perspective," Ziquan Deng and Zhaodan Kong use signal temporal logic (STL)<sup>12</sup> for representing abnormalities in learned values of sensors. The authors make a connection between fault diagnosis systems and continuous learning systems to be modeled as timed automata. The authors use STL,<sup>12</sup> an extension of temporal logic, to do so. STL extends traditional temporal operators of possibility  $\diamond$  and necessity  $\square$  by having real-time time bounds and real-valued comparisons for expressions like  $\diamond_{[0,1]}(x > 0.4)$ . The intended meaning of the expression is that within the time interval  $[0,1]$  the value of sensor  $x$  would become larger than 0.4. In addition, the STL formulas also have a robustness interpretation that measures the minimum or maximum deviation of the signal value from the expected bound within the time interval. The key contribution here is a framework for adding details

about the environment when the model checker finds branches that are not closed and do not lead to a counterexample because of the inadequacy of the environmental details.

"Toward Formal Methods for Smart Cities," by Meiyi Ma, John Stankovic, and Lu Feng, highlights some of the challenges we face when formally modeling smart cities. The authors focus on formal specifications, runtime monitoring, and learning aspects of modeling smart cities. The authors argue that STL is insufficient to express the requirements of runtime monitoring of a state of a city, predicting the future states of a city, and ensuring that deep learning results satisfy the city's requirements. In response to these challenges, the authors introduce spatial aggregation signal temporal logic (SaSTL), which extends STL by including logical operators for spatial aggregation and counting. The authors demonstrate the use of SaSTL to specify points of interest (PoIs), the physical distance and spatial relations of the PoIs and sensors, aggregation of signals over locations, degree/percentage of satisfaction, and temporal elements. The authors also introduce a runtime monitoring tool for smart city requirements specified in SaSTL. Their article concludes with a discussion of how their tool can help city managers improve a city's livability measures.

Smart cities have many sensors, learning systems, and artificial intelligence systems that learn and adjust parameters in smart buildings and smart grids to dynamically provide resident comfort, power savings, and on-demand transportation, among many other objectives. The work described in this article extends STL to SaSTL, formulates some of the learning objectives

of smart cities, and creates a monitoring system for the learning systems.

The article "Environment Modeling During Model Checking of Cyberphysical Systems," by Guangyao Chen and Zhihao Jiang, addresses the issue of modeling environments and model checking of CPS requirements. They propose a domain-independent framework for environment model abstraction and refinement to provide interpretable counterexamples while ensuring coverage of environment behaviors. Their CPSs and environments are modeled using timed automata.<sup>4</sup> They provide a closed-loop method of creating a CPS model-checking system. At the first stage, they model an environment and the chosen system's specification as a temporal logic formula on timed automata, and they use the Uppaal model checker to find counterexamples or proofs. In real-world settings, the environment models that are chosen may not sufficiently capture all of the details needed to expose ways in which requirements can be violated. The proposed framework captures this situation by using a nonleaf node that has counterexamples while its children do not. At this stage, the framework proposed by the authors allows more details to be added to the base model and the abstraction tree, which are refinements of the model providing counterexamples.

In "A Case Study in the Formal Modeling of Safe and Secure Manufacturing Automation," Matthew Jablonski, Bo Yu, Gabriela Ciocarlie, and Paulo Costa formally specify a manufacturing system for producing aluminum cans. They use temporal logic to specify the usage scenarios of the factory workflow

### ABOUT THE AUTHORS

**JAMES BRET MICHAEL** is a professor in the Naval Postgraduate School's Department of Computer Science and Department of Electrical and Computer Engineering, Monterey, California, 93943, USA. Michael received a Ph.D. in information technology from George Mason University. He is an associate editor in chief of *Computer* and the magazine's column editor for "Cybertrust." He is a Senior Member of IEEE. Contact him at [bmichael@nps.edu](mailto:bmichael@nps.edu).


**DORON DRUSINSKY** is a professor in the Naval Postgraduate School's Department of Computer Science, Monterey, California, 93943, USA. Druinsky received a Ph.D. in computer science from the Weizmann Institute of Science, Rehovot, Israel. He is *Computer's* column editor for "Algorithms." Contact him at [ddrusins@nps.edu](mailto:ddrusins@nps.edu).

**DUMINDA WIJESEKERA** is a professor in the Department of Computer Science and chair of the Cyber Security Engineering Department at George Mason University, Fairfax, Virginia, 22030, USA. Wijesekera received a Ph.D. in computer science from the University of Minnesota and in mathematical logic from Cornell University. He is a Senior Member of IEEE. Contact him at [dwijesek@gmu.edu](mailto:dwijesek@gmu.edu).

process. They use the Architecture Analysis and Design Language (AADL) to specify architectural elements and their potential faults and attacks. The authors use model checkers to show potential attack paths and faulty scenarios. AADL provides a complete modeling system of the software, hardware, and their communication systems. AADL has tool support for modeling fault analysis and real-time constraint violations. The airline industry has used AADL to analyze and simulate aircraft control designs for many years. AADL can also be used to analyze the performance of distributed tightly synchronized industrial control systems (ICSs), with the aim of determining whether the ICS satisfies system constraints and specifications. The article by Jablonski et al. provides a first attempt at doing so. The described

work shows that a tightly synchronized distributed manufacturing plant has many stations that work in synchrony to ensure that the individual plants satisfy their strict real-time deadlines and have sufficient built-in fault handling. In addition, each manufacturing station is built on many layers, where the higher layers of the system design depend on the lower layers being able to satisfy their own fault handling and providing this information to adjacent stations and human operators, with the goals of minimizing energy and material waste and delivering manufactured products on time.

**T**hese selected articles provide a glimpse at the breadth and depth of formal methods in

CPS-related topics. Given our obvious space limitations, we could only select a few of the larger plurality of papers. 

### REFERENCES

1. E. W. Dijkstra, "Notes on structured programming," Technological Univ. Eindhoven, The Netherlands, T. H.-Rep. 70-WSK-03, Apr. 1970.
2. "Temporal logic," Stanford Encyclopedia of Philosophy, First published Nov. 29, 1999; substantive revision Feb. 7, 2020. Accessed: June 3, 2021. [Online]. Available: <https://plato.stanford.edu/entries/logic-temporal>
3. D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. Cambridge, MA: MIT Press, 2000.
4. R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. doi: 10.1016/0304-3975(94)90010-8.
5. J. Raskin, "An introduction to hybrid automata," in *Handbook of Networked and Embedded Control Systems*, D. Hristu-Varsakelis and W. S. Levine, Eds. Boston: Birkhäuser, 2005, pp. 491–518.
6. D. Prawits, *Natural Deduction: A Proof-Theoretical Study*. New York: Dover Publications, 1961.
7. A. Nanevski, and A. Banerjee, and D. Garg, "Dependent type theory for verification of information flow and access control policies," *ACM Trans. Program. Lang. Syst.*, vol. 35, no. 2, pp. 1–41, 2013. doi: 10.1145/2491522.2491523.
8. R. Constable et al., *Implementing Mathematics With the Nuprl Proof Development System*. Upper Saddle River, NJ, Prentice Hall, 1986.
9. L. Bachmair and H. Ganzinger, "Resolution theorem proving," in *Handbook of Automated Reasoning*, vol. 1, A. Robinson and A. Voronkov,

- Eds. Amsterdam: Elsevier Science, 2001, pp. 19–99.
10. E. M. Clarke Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*, 2nd ed. Cambridge, MA: MIT Press, 2018.
  11. “Architecture analysis and design language.” Carnegie Mellon Univ. [https://www.sei.cmu.edu/our-work/projects/display.cfm?customel\\_datapageid\\_4050=191439&customel\\_datapageid\\_4050=191439](https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439&customel_datapageid_4050=191439) (accessed June 10, 2021).
  12. O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, vol. 3253, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 152–166. doi: 10.1007/978-3-540-30206-3\_12.
  13. J. B. Michael, D. N. Godbole, J. Lygeros, and R. Sengupta, “Capacity analysis of traffic flow over a single-lane automated highway system,” *Intell. Transp. Syst. J.*, vol. 4, nos. 1–2, pp. 49–80, 1997. doi: 10.1080/10248079808903736.



# Call for Articles

## IEEE Pervasive Computing

seeks accessible, useful papers on the latest peer-reviewed developments in pervasive, mobile, and ubiquitous computing. Topics include hardware technology, software infrastructure, real-world sensing and interaction, human-computer interaction, and systems considerations, including deployment, scalability, security, and privacy.

**Author guidelines:**  
[www.computer.org/mcl/pervasive/author.htm](http://www.computer.org/mcl/pervasive/author.htm)

**Further details:**  
[pervasive@computer.org](mailto:pervasive@computer.org)  
[www.computer.org/pervasive](http://www.computer.org/pervasive)

**IEEE pervasive COMPUTING**  
 MOBILE AND UBIQUITOUS SYSTEMS