The Role of Formal Methods in Computer Science Education

MAURICE H. TER BEEK, CNR–ISTI, Italy
MANFRED BROY, Technische Universität München, Germany
BRIJESH DONGOL, University of Surrey, UK
EMIL SEKERINSKI, McMaster University, Canada

In this paper, we advocate a prominent role of Formal Methods in the ACM CS 2023 curriculum and provide concrete suggestions for educators on incorporating Formal Methods into Computer Science (CS) education. To this aim, we proceed as follows. First, we underline the importance of Formal Methods *thinking* in CS education [11], since it provides the necessary rigor in reasoning about correctness—a fundamental skill for future software developers. Then, we argue that every computer scientist needs to *know* Formal Methods [4], since the skills and knowledge acquired from studying Formal Methods provide the indispensable solid foundation that forms the backbone of CS practice. This is confirmed by the increasing *use* of Formal Methods in Industry [2]—not limited to safety-critical domains. Finally, we underline that *teaching* Formal Methods need not come at the cost of displacing other engineering aspects of CS that are already widely accepted as essential [23]. On the contrary, Formal Methods have the potential to support and strengthen the knowledge in all these aspects.

CCS Concepts: • **Software and its engineering → Formal methods**; • **Social and professional topics → Computer science education**.

Additional Key Words and Phrases: formal methods, computer science education

ACM Reference Format:
Maurice H. ter Beek, Manfred Broy, Brijesh Dongol, and Emil Sekerinski. The Role of Formal Methods in Computer Science Education. November 2023, 8 pages.

1 INTRODUCTION

This paper summarizes discussion and evidence brought forward in four white papers on the role of Formal Methods in Computer Science (CS) education, which were written together with over 40 co-authors [2, 4, 11, 23]. Formal methods have multiple characterizations in the literature as languages and techniques (and tools) with rigorous mathematical foundations for the specification, development, and (manual or automated) analysis and verification of software and hardware systems [2, 4, 7, 11, 15, 26]. Formal Methods come in many shapes and sizes, ranging from light-weight static analysis to heavy-weight interactive theorem proving. We advocate for Formal Methods's role in complementing existing validation and verification techniques like testing and simulation. The distinguishing feature of Formal Methods is their capability to show the systematic usage of formal foundations in CS in engineering tasks such as mathematically guaranteeing the absence of errors in a given system.

Authors' addresses: Maurice H. ter Beek, maurice.terbeek@isti.cnr.it, CNR–ISTI, Pisa, Italy; Manfred Broy, broy@in.tum.de, Technische Universität München, Munich, Germany; Brijesh Dongol, b.dongol@surrey.ac.u, University of Surrey, Guildford, UK; Emil Sekerinski, emil@mcmaster.ca, McMaster University, Hamilton (ON), Canada.

Formal Methods do not yet appear in the current version of the ACM CS 2023 curriculum to the extent that reflects their fundamental role in CS and the benefits that a targeted education in Formal Methods can bring. In the above mentioned white papers, we make the following four key points.

(1) Every CS graduate needs to have an education in Formal Methods.
(2) Formal Methods support algorithmic problem solving, model-driven engineering, security, and many more.
(3) Formal Methods are applicable in numerous industrial domains, not limited to safety-critical applications.
(4) The current offering of Formal Methods in the CS curricula is inadequate.

Computer Science, namely the science of solving problems with software and software-intensive systems, provides the knowledge and skills to understand and capture precisely what a situation requires, and then develop a formal solution in a programming language. The most fundamental skill of a computer scientist, that of *abstraction*, is effectively addressed by Formal Methods. They provide the rigor for reasoning in precise terms about goals, such as validation and verification, thus guaranteeing adequacy, accuracy, and correctness of implementations.

At the least Formal Methods thinking, i.e., the ideas from Formal Methods applied in informal, light-weight, practical, and accessible ways, should be part of the recommended curriculum for every CS student. Even students who train only in that 'thinking' will become much better programmers. In addition, there are students who, exposed to those ideas, will be ideally positioned to study more: why the techniques work; how they can be automated; and how new ones can be developed. They could follow subsequently an optional path, including topics such as semantics, logics, and proof-automation techniques.

Formal Methods can assist in teaching programming to novices more effectively than by informal reasoning and testing. Formal Methods explain algorithmic problem solving, design patterns, model-driven engineering, software architecture, software product lines, requirements engineering, and security, i.e., are complementary to these CS fields. Formalisms can concisely and precisely express underlying fundamental design principles and equip programmers with a tool to handle related problems.

Formal Methods are becoming widely applied in Industry, from eliciting requirements and early design to deployment, configuration, and runtime monitoring. Evidence of successfully applying Formal Methods in Industry ranges from well-known stories in the safety-critical domain, such as railways and other transportation domains, to areas such as lithography manufacturing and cloud security in e-commerce, for example. Testimonies come from representatives who, either directly or indirectly, use or have used Formal Methods in their industrial project endeavours. Importantly, they are spread geographically, including Europe, Asia, North and South America.

ACM CS 2023 is the ideal time and place to adjust the way we teach CS. There are mature tools and proofs of concept available and the possibility of designing coherent teaching paths. Importantly, this can be done without displacing the other 'engineering' aspects of CS already widely accepted as essential. Support for teachers is available, for instance via Formal Methods Europe (FME)[1].

The next four sections are based on the above mentioned four white papers, each dealing with a different one of the aforementioned four key points: in Section 2, we emphasise the importance of Formal Methods *thinking* in CS education, followed by underlining the importance of *knowing* Formal Methods for CS graduates in Section 3, as witnessed by practitioners *using* Formal Methods in Industry, as testimonied in Section 4, after which we aim to convince the reader that *teaching* Formal Methods need not come at the cost of displacing other engineering aspects of CS in Section 5. Section 6 concludes the paper. All quotes reported hereafter stem from the four white papers, unless indicated otherwise.

[1] https://fmeurope.org/ (visited on 15 november 2023)

2 FORMAL METHODS THINKING

Formal Methods thinking offers distinct advantages over relying solely on program execution understanding and testing. Testing, for instance, has limitations: it may not guarantee correctness, subjective judgment is needed for some outputs, and untried cases can hide errors. A Formal Methods course provides students with an independent understanding of programs by proving their satisfaction of specifications for all inputs, surpassing testing capabilities. While testing remains crucial, Formal Methods augments it by reducing the likelihood of mistakes, improving specification precision, and fostering reflective design practices, thus instilling in students "a mindset of reflecting on our designs and checking (or verifying) that the intentions (or requirements) are met". Formal Methods enhance one's argumentation skills and lay the foundation for solid reasoning about software systems' behavior.

Most CS curricula initially cover mathematical foundations alongside introductions to programming, algorithms, and data structures. As students progress, they specialize in various applications and CS knowledge areas like databases, security, concurrency, networks, Artificial Intelligence (AI), etc. The level of foundational mathematics varies based on later specializations, encompassing topics such as discrete mathematics, logic, probability theory, and linear algebra. While the discussion of the appropriate level of mathematics for the average programmer is not the focus here, a reasonable argument could be made that "people should not be writing programs whose functionality requires discrete mathematics to describe it… unless they have some command of discrete mathematics themselves." We actually believe that Formal Methods thinking should extend beyond discrete mathematics to address also the dynamics and behavior of modern systems.

Obviously, Formal Methods thinking is particularly beneficial in programming, enhancing students' ability to model computational problems and comprehend their code, thereby improving their programming skills. However, Formal Methods thinking extends beyond programming and can be applied throughout the software life cycle phases:

Analysis: system modeling and requirements elicitation and specification;
Product Design: specification of functional requirements;
Software: Development establishing correctness, writing clean code and documentation;
Testing: complementing testing with property verification;
Maintenance: system correction and adaptation.

Moreover, Formal Methods are also commonly employed in Software Engineering, Cybersecurity, and Computer Networking—to name but a few. The central challenge addressed by Formal Methods is "the need for precision and rigour in modelling and analysing computer systems and software".

3 KNOWING FORMAL METHODS

Early computing pioneers like Turing and von Neumann recognized the importance of reasoning about programs, publishing papers in the 1940s that demonstrated the possibility of recording proofs for program correctness. Over the decades, researchers have enhanced the tractability of specification and reasoning, developing supporting software tools, which still require users to understand the underlying formalisms. Just like structural engineers do not always apply their most rigorous techniques, not all software needs to be developed formally. But just like structural engineers need to learn the foundational methods supporting documentation and reasoning about their designs, "software engineers must learn that precise specifications can be constructed and that their key design decisions are subject to rigorous justification." No structural engineer would be permitted to work on the design of a bridge without a solid understanding of the relevant mathematical concepts.

Building reliable systems necessitates rigorous development approaches rooted in abstract models, unambiguous specifications, thorough testing, and verification methods to ensure system requirements are met. While software systems do not endure wear and tear, changing environments introduce new requirements, necessitating long-term maintenance plans. CS is an independent engineering discipline, which draws from and interacts with other engineering disciplines, but relies heavily on Formal Methods. Computer programs interface with the physical world and impact the real world, from controlling machinery to managing energy distribution. In many such domains, testing is not sufficient but formal descriptions are needed to predict and prevent unintended consequences with significant responsibility. Creating these formal descriptions requires skills in abstraction, rigor, and a clear understanding of the model's semantics. "It seems irresponsible to let anyone design high impact computer control programs without suitable training in formal methods to mitigate potential risks." This applies not only to safety-critical applications.

Fortunately, programmers commonly use some (light-weight) Formal Methods (thinking) in their daily work, such as type systems for defining formal requirements on value expressions and checking compliance of the values produced by expressions with given types. This also aids them in program decomposition and structuring. Some programming languages have type systems that demand a formal understanding of types and the type-checking process, and the ability to apply abstraction. In practice, often formal techniques are not called "Formal Methods" since they are so smoothly integrated into the engineering. Formal methods can act as a bridge between pure mathematics and general software development. "Formal methods thinking consists of describing a system to be understood or designed in terms of fundamental discrete mathematical entities such as sets, lists, maps, relations, functions, and constraints." Such formalization typically unveils issues not seen otherwise and it moreover fosters an early shared vision among stakeholders. Abstraction plays a crucial role in this process [9, 19].

Computing Science is the discipline of the formal. Programs and software are formal entities. The steps from an idea and an informal problem description to a program or a piece of software are the steps from the informal to the formal. Formal Methods are thus in the heart of CS.

4 FORMAL METHODS IN INDUSTRY

While many well-known success stories of applying Formal Methods in Industry concern safety-critical systems [13, 16, 26], recent literature reports an uptake in the application of Formal Methods also outside safety-critical applications; e.g., to ensure the quality of cloud services at Amazon [1, 21], of cloud databases and weak memory models at Huawei [14, 22], and of mobile apps at Facebook [10]. Furthermore, in [2], a few representatives from Industry contributed testimonies concerning the use of Formal Methods in their projects, not limited to the safety-critical domain. We include some relevant parts of their contributions here.

Byron Cook (founder of the automated reasoning group at AWS): "Formal methods is transforming how Amazon Web Services (AWS) secures the cloud. Security has historically been a manual, high-judgement and thus un-scalable field; Automated formal reasoning is challenging that entire structure, changing both the quality of AWS products and the cost structure to support them. The key at AWS has been to avoid 'shiny-object syndrome' and instead build and apply tools that quietly but reliably change the behavior of engineers. Many leaders at AWS were skeptical of this type of work in 2016, but the success in areas such as cryptography, identity, storage and virtualization has changed minds."

Rod Chapman (AWS): "In late 2020, AWS announced the availability of strong read-after-write consistency in the S3 storage service. S3 operates at preposterous scale, storing over 100 Trillion objects and handling over 10 Million requests per second [25]. Strong consistency ensures that the same view of an object is available to all readers instantly

following a write operation to that object. Consistency properties were specified and verified using Dafny [20]"—a verification-aware programming language.

Ivo ter Horst (ASML): "To make ASML's lithography systems run reliably and consistently ASML needs software that sends unambiguous instructions in every situation to the carefully engineered hardware. One way that ASML ensures this is by formally verifying (model checking) the specified machine behaviour and automatically generating correct and semantically equivalent code from those models [3]."

5 TEACHING FORMAL METHODS

Fundamental Formal Methods in CS, comprising modeling, formal specification, refinement, and verification, constitute a core knowledge area with widespread relevance in many of today's innovative applications, like self-driving cars, in a society that increasingly relies on software systems. Currently, discrete mathematics courses are often perceived as early challenges in CS education, disconnected from modern programming languages, yet they are crucial springboards for introducing Formal Methods. "An additional core area directly focused on formal methods can help contextualize discrete mathematics courses for students, and can demonstrate why such courses are taught so early as a starting foundation for a computer science education."

Currently, CS 2023 envisions 17 knowledge areas[2], several of which we believe are, or rather should be, related to Formal Methods.

Algorithmic Foundations. The focus of this knowledge area is on teaching fundamental data structures, classical algorithms, algorithm construction strategies, and computational complexity and computability theory. While there are suggestions on addressing invariants, especially in loops and search algorithms, the curriculum lacks explicit competencies related to reasoning about algorithm correctness. "Students should learn from the beginning to reason (at least informally) about the correctness of their algorithms." A possible way to instil this type of reasoning in students could be to teach them the classical algorithms with arguments for correctness. Recently, a bug was found in the TimSort sorting algorithm of the Java standard library using Formal Methods [17].

Architecture and Organization. This knowledge area strives to enhance comprehension of the hardware environments that underpin nearly all computing and the corresponding interfaces provided to higher software layers. The scope of the hardware considered spans from low-end embedded system processors to high-end enterprise multiprocessors. Formal Methods are employed in this area to validate the accuracy of hardware designs and to guarantee that the combination of hardware and software components adheres to their specifications, for example to verify security requirements in hardware security architectures [12]. Formal modeling of application architectures by specifying the interface behavior of components is the backbone of architecture design and system integration.

Artificial Intelligence. This knowledge area prepares CS students to recognize when it is suitable to use an AI method (e.g., neural networks, machine learning) and how to apply it, taking the broader societal impacts and implications into account, including issues in AI ethics, fairness, trust, and explainability. Formal Methods are used to capture the assumptions of the designs of deep neural networks as used in large language models as well as their verification (with model-checking and interactive theorem proving techniques) or counterexample-based retraining [5, 18, 24].

[2] https://csed.acm.org/knowledge-areas/ (visited on 15 november 2023)

Parallel and Distributed Computing. This knowledge area encompasses various topics, ranging from parallelization and dependencies to progress, deadlocks, faults, safety, and liveness. Although Formal Methods are not explicitly mentioned, the suggested learning outcomes for core CS contain examples like "Write a program that correctly terminates when all of a set of concurrent tasks have completed", which obviously requires knowledge of correctness, termination, and rigorous reasoning about programs, as well as rigorous semantics of concurrency. "This area states as prerequisites logic, discrete maths, foundations of software engineering, but none of them in the current status of the ACM standard provides the ability to be able to understand and justify correctness of computational systems."

Security. This knowledge area focuses on instilling a security mindset in CS students by understanding vulnerabilities of—and threats against—software systems, ensuring that security (including concepts like privacy and cryptography) is inherent in all their output. Formal Methods can provide the assurance of security properties in algorithms and protocols, ensuring their resilience against attacks. Recently, Formal Methods were used at Amazon Web Services to reason about encryption properties [8].

Software Development Fundamentals. This knowledge area covers fundamental concepts and skills concerning programming, the use of data structures, and an understanding of how algorithms impact program performance. As mentioned earlier, it is our belief that algorithms should not be detached from reasoning about their correctness, yet Formal Methods are not mentioned at all. In general, however, Formal Methods concepts enter mainstream programming (think of contracts in C++ or mutexes in concurrent programming). We find it hard to imagine their effective use without knowing Formal Methods. "We teach children counting and algebra before giving them a pocket calculator for a very good reason."

Software Engineering. This knowledge area centers on appropriate means for software design, construction, and verification and validation, primarily through testing. A non-core Formal Methods module suggests learning outcomes like "describe the role of formal specification and analysis", while testing is the primary validation technique in other modules. However, Formal Methods and testing are not mutually exclusive: "Understanding correctness and reasoning about programs can greatly benefit effective testing." Formal Methods tools for static analysis, like Infer, are used by well-known companies such as Facebook [10].

According to a recent survey involving 130 Formal Methods experts (including 3 Turing Award winners[3], all 4 FME Fellowship Award winners[4], and 16 CAV Award winners[5]) the most suitable place for Formal Methods in a teaching curriculum is "in bachelor courses at the university", since this is what 79.2% of the respondents answered to the question *When and where should formal methods be taught?* [15, Sect. 5]. Furthermore, the fact that "engineers lack proper training in formal methods" is the key *limiting factor for a wider adoption of formal methods by industry*, according to 71.5% of the respondents [15, Sect. 5]. The survey concludes that "the current situation [of formal methods education] is very heterogeneous across universities, and many experts call for a standardisation of university curricula with respect to formal methods", which is confirmed by a recent white paper advocating "the inclusion of a compulsory formal methods course in computer science and software engineering curricula" based on the observation that "there is a lack of computer science graduates who are qualified to apply formal methods in industry" [6].

[3] https://amturing.acm.org/byyear.cfm (visited on 15 november 2023)
[4] https://www.fmeurope.org/awards/ (visited on 15 november 2023)
[5] http://i-cav.org/cav-award/ (visited on 15 november 2023)

6 CONCLUSION

We have presented discussion and evidence from four white papers that advocate a prominent role of Formal Methods in CS education. Currently, Formal Methods do not appear as a knowledge area in the ACM CS 2023 curriculum. We argumented that the current offering of Formal Methods in the CS education is inadequate because every CS graduate needs to have an education in Formal Methods, since they can support algorithmic problem solving, model-driven engineering, security, and many more areas of CS, and are moreover applicable in numerous industrial domains, not limited to safety-critical applications. The current revision of the ACM CS 2023 curriculum is the ideal moment to adjust the way CS is taught, incorporating Formal Methods as a knowledge area without displacing other widely accepted engineering aspects of CS. We have put forward to what extent seven of the 17 knowledge areas of CS 2023 are, or rather should be, related to Formal Methods. Four white papers [2, 4, 11, 23] reinforce the conclusion of an earlier white paper [6]—all together authored by more than 50 computer scientist and practitioners worldwide—in which it is argued that "formal methods need to be better rooted in higher education curricula for computer science and software engineering programmes of study."

ACKNOWLEDGMENTS

We thank our co-authors of the four white papers [2, 4, 11, 23] for their contributions: Achim Brucker, Rod Chapman, Marsha Chechik, Rance Cleaveland, Catherine Dubois, Alessandro Fantechi, João Ferreira, Hubert Garavel, Mario Gleirscher, Rong Gu, Stefan Hallerstede, John Hatcliff, Klaus Havelund, Eric Hehner, Michael Hicks, Ivo ter Horst, Daniel Jackson, Cliff Jones, Jeroen Keiren, Markus Kuppe, Kevin Lano, Thierry Lecomte, Michael Leuschel, Alexandra Mendes, Carroll Morgan, Peter Müller, André Platzer, Leila Ribeiro, Jan Oliver Ringert, Kristin Yvonne Rozier, Augusto Sampaio, Cristina Seceleanu, Alexandra Silva, Graeme Smith, Allison Sullivan, Martyn Thomas, Erik de Vink, Tim Willemse, and Lijun Zhang. We also thank Ana Cavalcanti and Luigia Petri for comments on an early draft which helped improve the paper.

REFERENCES

[1] John Backes, Pauline Bolignano, Byron Cook, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Martin Schäf, Cole Schlesinger, Rima Tanash, Carsten Varming, and Michael W. Whalen. 2019. One-Click Formal Methods. *IEEE Softw.* 36, 6 (2019), 61–65. https://doi.org/10.1109/MS.2019.2930609

[2] Maurice H. ter Beek, Rod Chapman, Rance Cleaveland, Hubert Garavel, Rong Gu, Ivo ter Horst, Jeroen J. A. Keiren, Thierry Lecomte, Michael Leuschel, Kristin Y. Rozier, Augusto Sampaio, Cristina Seceleanu, Martyn Thomas, Tim A. C. Willemse, and Lijun Zhang. 2023. Formal Methods in Industry. *Submitted to Form. Asp. Comput.* (2023).

[3] Lewis Binns. 2023. By computers, for computers: Improving scanner metrology software with generated code. https://www.linkedin.com/pulse/computers-improving-scanner-metrology-software-code-lewis/

[4] Manfred Broy, Achim D. Brucker, Alessandro Fantechi, Mario Gleirscher, Klaus Havelund, Cliff Jones, Markus Kuppe, Alexandra Mendes, André Platzer, Jan Oliver Ringert, and Allison Sullivan. 2023. Does Every Computer Scientist Need to Know Formal Methods? *Submitted to Form. Asp. Comput.* (2023).

[5] Achim D. Brucker and Amy Stell. 2023. Verifying Feedforward Neural Networks for Classification in Isabelle/HOL. In *Proceedings of the 25th International Symposium on Formal Methods (FM'23) (LNCS, Vol. 14000)*, Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker (Eds.). Springer, Germany, 427–444. https://doi.org/10.1007/978-3-031-27481-7_24

[6] Antonio Cerone, Markus Roggenbach, James Davenport, Casey Denner, Marie Farrell, Magne Haveraaen, Faron Moller, Philipp Körner, Sebastian Krings, Peter Csaba Ölveczky, Bernd-Holger Schlingloff, Nikolay Shilov, and Rustam Zhumagambetov. 2021. Rooting Formal Methods Within Higher Education Curricula for Computer Science and Software Engineering – A White Paper. In *Revised Selected Papers of the 1st International Workshop on Formal Methods – Fun for Everybody (FMFun'19) (CCIS, Vol. 1301)*, Antonio Cerone and Markus Roggenbach (Eds.). Springer, Germany, 1–26. https://doi.org/10.1007/978-3-030-71374-4_1

[7] Edmund M. Clarke, Jeannette M. Wing, et al. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* 28, 4 (1996), 626–643. https://doi.org/10.1145/242223.242257

[8] Byron Cook. 2018. Formal Reasoning About the Security of Amazon Web Services. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18) (LNCS, Vol. 10982)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, Germany, 38–47. https://doi.org/10.1007/978-3-319-96145-3_3

[9] Edsger W. Dijkstra. 1989. On the Cruelty of Really Teaching Computing Science. *Commun. ACM* 32, 12 (1989), 1398–1404. https://doi.org/10.1145/76380.76381 This is Dijkstra's Contribution to A Debate on Teaching Computing Science by Peter J. Denning, pp. 1397—1414.

[10] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (2019), 62–70. https://doi.org/10.1145/3338112

[11] Brijesh Dongol, Catherine Dubois, Stefan Hallerstede, Eric Hehner, Daniel Jackson, Carroll Morgan, Peter Müller, Leila Ribeiro, Alexandra Silva, Graeme Smith, and Erik de Vink. 2023. On Formal Methods Thinking in Computer Science Education. *Submitted to Form. Asp. Comput.* (2023).

[12] Andrew Ferraiuolo, Rui Xu, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. 2017. Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, USA, 555–568. https://doi.org/10.1145/3037697.3037739

[13] Alessio Ferrari and Maurice H. ter Beek. 2023. Formal Methods in Railways: a Systematic Mapping Study. *ACM Comput. Surv.* 55, 4 (2023), 69:1–69:37. https://doi.org/10.1145/3520480

[14] Song Gao, Bohua Zhan, Depeng Liu, Xuechao Sun, Yanan Zhi, David N. Jansen, and Lijun Zhang. 2021. Formal Verification of Consensus in the Taurus Distributed Database. In *Proceedings of the 24th International Symposium on Formal Methods (FM'21) (LNCS, Vol. 13047)*, Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, Germany, 741–751. https://doi.org/10.1007/978-3-030-90870-6_42

[15] Hubert Garavel, Maurice H. ter Beek, and Jaco van de Pol. 2020. The 2020 Expert Survey on Formal Methods. In *Proceedings of the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS'20) (LNCS, Vol. 12327)*, Maurice H. ter Beek and Dejan Ničković (Eds.). Springer, Germany, 3–69. https://doi.org/10.1007/978-3-030-58298-2_1

[16] Mario Gleirscher and Diego Marmsoler. 2020. Formal Methods in Dependable Systems Engineering: A Survey of Professionals from Europe and North America. *Empir. Softw. Eng.* 25, 6 (2020), 4473–4546. https://doi.org/10.1007/s10664-020-09836-5

[17] Stijn de Gouw, Frank S. de Boer, Richard Bubel, Reiner Hähnle, Jurriaan Rot, and Dominic Steinhöfel. 2019. Verifying OpenJDK's Sort Method for Generic Collections. *J. Autom. Reason.* 62, 1 (2019), 93–126. https://doi.org/10.1007/s10817-017-9426-4

[18] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19) (LNCS, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, Germany, 443–452. https://doi.org/10.1007/978-3-030-25540-4_26

[19] Jeff Kramer. 2007. Is Abstraction the Key to Computing? *Commun. ACM* 50, 4 (2007), 36–42. https://doi.org/10.1145/1232743.1232745

[20] K. Rustan M. Leino. 2023. *Program Proofs.* MIT Press, USA. https://mitpress.mit.edu/9780262546232/program-proofs/

[21] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services Uses Formal Methods. *Commun. ACM* 58, 4 (2015), 66–73. https://doi.org/10.1145/2699417

[22] Jonas Oberhauser, Rafael Lourenco de Lima Chehab, Diogo Behrens, Ming Fu, Antonio Paolillo, Lilith Oberhauser, Koustubha Bhat, Yuzhong Wen, Haibo Chen, Jaeho Kim, and Viktor Vafeiadis. 2021. VSync: Push-Button Verification and Optimization for Synchronization Primitives on Weak Memory Models. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*. ACM, USA, 530–545. https://doi.org/10.1145/3445814.3446748

[23] Emil Sekerinski, Marsha Chechik, João F. Ferreira, John Hatcliff, Michael Hicks, and Kevin Lano. 2023. Should We Teach Formal Methods or Algorithmic Problem Solving, Design Patterns, Model-Driven Engineering, Software Architecture, Software Product Lines, Requirements Engineering, and Security? *Submitted to Form. Asp. Comput.* (2023).

[24] Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. 2018. Formal Specification for Deep Neural Networks. In *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA'18) (LNCS, Vol. 11138)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, Germany, 20–34. https://doi.org/10.1007/978-3-030-01090-4_2

[25] Werner Vogels. 2021. Diving Deep on S3 Consistency. https://www.allthingsdistributed.com/2021/04/s3-strong-consistency.html

[26] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM Comput. Surv.* 41, 4 (2009), 19:1–19:36. https://doi.org/10.1145/1592434.1592436

Received 19 November 2023