Formal Methods for the Informal Engineer: Workshop Recommendations

Workshop Leadership and Contributors

Program Committee

Gopal Sarma	Broad Institute of MIT and Harvard
Jimmy Koppel	MIT CSAIL
Gregory Malecha	BedRock Systems
Patrick Schultz	Broad Institute of MIT and Harvard
Eric Drexler	James Martin School of Oxford University
Ramana Kumar	DeepMind

Tutorial Leaders

Cody Roux	Draper Labs
Phil Zucker	Draper Labs

Speakers

Kathleen Fisher	Tufts University
Gregory Malecha	BedRock Systems
Hillel Wayne	Windy Coast Consulting
Matthew Mirman	ETH Zurich

Corresponding author: gsarma@broadinstitute.org

1 Introduction

In 2021, a workshop was convened at the Broad Institute of MIT and Harvard to explore potential applications of formal methods and programming language theory to software platforms being developed in the life sciences. The vision to host this workshop at the Broad Institute originated in conversations about economic incentives, the exponential growth of multi-modal data sources, and challenging biomedical problems that have resulted in the life sciences emerging as both key consumers and producers of software and AI/ML technologies [1–4]. We view this next decade as a critical growth phase for this process and an opportunity to shape the software engineering culture of the life sciences from the ground up. Safety and security, realized through both informal and formal methods, are central to this goal [5–7].

The result of these conversations was the event Formal Methods for the Informal Engineer (FMIE), a workshop aimed at highlighting recent successes in the development of verified software. Given the relative youth of industrial-scale software engineering and distributed AI/ML platforms in the life sciences, we decided to make the content domain agnostic. Our aim was to expose researchers and developers in the life sciences to what was possible using formal methods in order to shape their outlook and guide future decision making. The workshop was a venue for collecting the success stories of formal methods, regardless of their domain, exploring applications to real-world software development, and connecting researchers to aspiring users.

The event was organized as a "flipped workshop," and on the first two days, to accommodate an audience consisting of many interested non-experts, we organized hands-on tutorials with two different verification technologies, the Coq and Z3 theorem provers [8–10]. This provided many members of the audience with exposure to basic concepts of verification. The final day was organized as a traditional symposium with four sessions on General Topics, Verified Software Components, Distributed Systems, and Robust Machine Learning.

To more broadly communicate the lessons learned from organizing this workshop and the many conversations between a diverse group of life sciences researchers, physicians, software engineers, ML scientists, and programming language theorists, we make the following recommendations regarding the potential uses of and deployment strategies for formal methods. Although the workshop and these recommendations were conceived with a view towards organizations in the life sciences, they are domain agnostic and relevant to any organization that requires mission-critical or safety-critical software systems.

2 Recommendations

Recommendation 1: *Software leaders should investigate the use of formal methods as part of ongoing and future development.*
Formal methods are worth considering for many biomedical software projects, particularly those that require high assurances of data integrity, as in the case of patient data, that operate at scale and implement difficult concurrent algorithms, as in the case of aggregated genomic data, or that are safety-critical, such as medical devices in a hospital setting or automated, closed-loop biological laboratories.

Recommendation 2: *Software leaders should ensure that their projects are held to the highest standards of quality and robustness from the standpoint of traditional software engineering prior to considering the use of formal methods.*
Formal methods are neither a substitute nor a cure for otherwise haphazard development practices. Indeed, poor quality software foundations are one of the major hurdles to applying formal methods [11]. Formal methods should be considered as an additional set of tools and development practices that can give assurances beyond what traditional methods can provide, but that rely on well-built software as a starting point [12].

Recommendation 3: *Verification can be approached in an incremental manner, starting with the process of formalizing a specification.*
Investing in formal techniques does not need to be an all-or-nothing proposition. In particular, there is value in writing formal specifications for parts of a codebase, even if the results are never formally verified [13, 14]. The process of writing formal specifications alone can uncover bugs, especially architectural ones, early in the development process and result in many bugs being eliminated. Moreover, this process lays the foundation for a seamless transition to more rigorous verification when needed [12, 15, 16].

Recommendation 4: *Software leaders should consider opportunities to build on top of already secure foundations.*
There are a number of existing software components, ranging from compilers, to microkernels, to file systems that have undergone rigorous verification and that could form the basis of new software in biomedicine [6, 17–20]. The purpose of Recommendation 5, embedding experts within existing teams, can aid the process of identifying appropriate foundations.

Recommendation 5: *Software leaders should consider embedding experts from the formal methods community within their software teams to assess how and where verification might be beneficial.*
Organizations should strive to build a rich culture of outstanding software engineering practices with a close link to formal methods. Even in situations where full-scale verification is not employed, FM tools such as static analyzers should be used routinely and architectural decisions should be informed by formal methods. Early education, project scoping, and strategic investment are key to ensuring that mission and safety-critical systems are available when needed. Embedding formal methods experts within teams can play a critical role in identifying those tools, techniques, and training opportunities worth investing in and help to create a culture of safety that balances short and long-term needs.

3 Conclusion

The life sciences and medicine are poised to become leaders in the production of novel AI/ML technologies, many of which will be mission or safety-critical. Preparing for the widespread deployment of these systems is an urgent task, which demands long-term thinking and careful, deliberate attention to building bridges between communities with little previous interaction. Our experience with FMIE suggests that bringing together experts from the relevant fields of formal methods, programming language theory, and biomedicine can be done fruitfully and lead to concrete next actions to deepen the interaction between these spheres. We urge readers to consider the nexus of the life sciences and medicine with AI/ML and to work to ensure that the resulting technologies are developed not only with utmost regard to fairness and access, but also safety and security.

Acknowledgements

We would like to thank our sponsors Leaf Labs, Google DeepMind, CEA, and the Broad Institute of MIT and Harvard.

References

[1] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big Data: Astronomical or Genomical?," *PLoS Biology*, vol. 13, no. 7, p. e1002195, 2015.

[2] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, *et al.*, "Opportunities and obstacles for deep learning in biology and medicine," *Journal of The Royal Society Interface*, vol. 15, no. 141, p. 20170387, 2018.

[3] N. B. Benaich and I. Hogarth, "State of AI Report 2020." https://www.stateof.ai. Accessed: 2020-03-16.

[4] E. Schmidt, R. Work, S. Catz, S. Chien, M. Clyburn, C. Darby, K. Ford, J.-M. Griffiths, E. Horvitz, A. Jassy, G. Louie, W. Mark, J. Matheny, K. McFarland, and A. Moore, "Final Report: National Security Commission on Artificial Intelligence." https://www.nscai.gov/wp-content/uploads/2021/03/Full-Report-Digital-1.pdf. Accessed: 2020-03-16.

[5] National Academies of Sciences, Engineering, and Medicine, *Safeguarding the Bioeconomy*. Washington, DC: The National Academies Press, 2020.

[6] K. Fisher, J. Launchbury, and R. Richards, "The HACMS program: using formal methods to eliminate exploitable bugs," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 375, no. 2104, p. 20150401, 2017.

[7] S. Chong, J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, and N. Zeldovich, "Report on the NSF Workshop on Formal Methods for Security," *arXiv preprint arXiv:1608.00678*, 2016.

[8] The Coq Development Team, "The Coq Proof Assistant," Jan. 2021.

[9] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer Science & Business Media, 2013.

[10] L. De Moura and N. Bjørner, "Z3: An efficient SMT Solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, 2008.

[11] M. Hinchey, M. Jackson, P. Cousot, B. Cook, J. P. Bowen, and T. Margaria, "Software Engineering and Formal Methods," *Communications of the ACM*, vol. 51, no. 9, pp. 54–59, 2008.

[12] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How Amazon Web Services Uses Formal Methods," *Communications of the ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[13] S. Easterbrook and J. Callahan, "Formal Methods for Verification and Validation of Partial Specifications: A Case Study," *Journal of Systems and Software*, vol. 40, no. 3, pp. 199–210, 1998.

[14] D. Selsam, P. Liang, and D. L. Dill, "Developing Bug-Free Machine Learning Systems with Formal Mathematics," in *International Conference on Machine Learning*, pp. 3047–3056, PMLR, 2017.

[15] H. Wayne, *Practical TLA+: Planning Driven Development*. Apress, 2018.

[16] L. Lamport, *Introduction to TLA*. Digital Equipment Corporation Systems Research Center [SRC], 1994.

[17] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand, "CompCert - A Formally Verified Optimizing Compiler," in *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*, 2016.

[18] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, *et al.*, "seL4: Formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220, 2009.

[19] H. Chen, D. Ziegler, T. Chajed, A. Chlipala, M. F. Kaashoek, and N. Zeldovich, "Using Crash Hoare logic for certifying the FSCQ file system," in *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 18–37, 2015.

[20] Y. K. Tan, M. O. Myreen, R. Kumar, A. Fox, S. Owens, and M. Norrish, "A new verified compiler backend for cakeml," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pp. 60–73, 2016.