Formal Methods-based Certification Frameworks for Scientific Computing Applications

Ariel Kellison [1,2], Geoff Hulette [2], John Bender [2], Samuel D. Pollard [2], and Heidi K. Thornquist [2]

[1]Cornell University, Ithaca, New York
[2]Sandia National Laboratories, Livermore, California

October 14, 2021

A significant barrier to certifiable, trustworthy computational science at the Department of Energy is the prevalent belief that proofs of accurate correspondence between an implementation and a desired conceptual model along with its solution are intractable. The current verification paradigm for computational science at the DOE therefore relies on the *absence of proof of incorrectness* [7]; this paradigm is woefully behind common verification practices employed in industry and academia, and ultimately means that national security and policy decisions based on computational models lack the highest assurances possible.

To address the absence of formal guarantees of correctness of critical computational models and simulations across DOE labs, we suggest the development of end-to-end formal methods-based certification frameworks for scientific computing applications. These frameworks can bridge the gap between conceptual models and their corresponding high-performance implementations.

Numerous examples from academia and industry provide strong evidence suggesting that the development of such frameworks is possible, and that these frameworks can be used to provide correctness guarantees for large and complex systems. Below, we provide some details on formal methods-based certification frameworks relevant to each of the three areas of verification [7] fundamental to trustworthy computational science.

Numerical algorithm verification

The current method employed for numerical algorithm verification across Department of Energy labs relies on the accumulation of evidence from test cases. The validity of this evidence rests on the assumption of a representative set of test cases and aims to prove that the implementation behaves according to an informal specification of correct behavior. In contrast, formal-methods based approaches that are extensively used in industry [6] enable the development of mathematically precise, verifiable program specifications. Furthermore, machine-checkable proof certificates of the correctness and numerical accuracy of imperative implementations of numerical algorithms have been demonstrated [1, 8].

Software Quality Assurance

The complexity of computational science models and simulations increases the likelihood of human errors and leaves the software development cycle vulnerable to adversarial agents. Before trusting the results of critical models and simulations, we need to establish extremely high confidence that each component of the system on which the simulation relies, from hardware to software, is correct. To that end, formal methods used in industry enable the precise definition of program behavior, and ensure that records of expected behavior are consistent throughout the software development stack [5, 6]. Fully verified software stacks that account for the behavior of underlying hardware have been demonstrated [4]. Furthermore, recent work has demonstrated that machine-checkable proof certificates of correctness for small system components can be composed to ensure the correctness of larger complex systems [3].

Solution Verification

Given a set of continuous ODEs or PDEs, solution verification entails a quantitative study of how accurately a discrete numerical implementation represents the original set of continuous equations and its qualitative behavior. Popular opinion in the computational science community is that formal mathematical proofs of compatibility between a numerical implementation and a continuous model are intractable in practice [7]. However, demonstrations of such proofs have appeared in the literature [2]. Notably these proofs come equipped with machine-checkable proof certificates that can be integrated with the methods proposed for numerical algorithm verification and software quality assurance.

References

[1] Andrew W. Appel and Yves Bertot. C floating-point proofs layered with VST and Flocq. *Journal of Formalized Reasoning*, 2020.

[2] Sylvie Boldo, François Clément, Jean-Christophe Filliítre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Trusting Computations: A Mechanized Proof from Partial Differential Equations to Actual Program. *Computers and Mathematics with Applications*, 2014.

[3] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI'16, 2016.

[4] Chris Hawblitzel, Jon Howell, Jacob R. Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. Ironclad Apps: End-to-End Security via Automated Full-System Verification. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI'14, 2014.

[5] K. Rustan M. Leino. Dafny: An Automatic Program Verifier for Functional Correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, 2010.

[6] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services Uses Formal Methods. *Commun. ACM*, 2015.

[7] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing.* Cambridge University Press, 2010.

[8] Tahina Ramananandro, Paul Mountcastle, Benoît Meister, and Richard Lethin. A Unified Coq Framework for Verifying C Programs with Floating-Point Computations. CPP 2016, 2016.