Proof Assistants at the Hardware-Software Interface (Invited Talk)

Adam Chlipala
Massachusetts Institute of Technology
USA
adamc@csail.mit.edu

Abstract

Some of the earliest applications of proof assistants were to correctness of digital hardware designs, but the subject doesn't come up too frequently today at venues like CPP. I will try to make the case that proof assistants are a crucial tool for resolving both classical problems and new ones at *the hardware-software interface*. That is, it is important to understand exactly what guarantees a processor exports to software, it is important to verify that hardware exports those guarantees correctly, and it is important to prove end-to-end theorems covering both hardware and software. A few social developments make this an exciting time to tackle these problems: open instruction sets and open-source hardware designs are growing in real-world relevance, and surprising new classes of security vulnerabilities have gotten more practitioners thinking about precise hardware-software contracts. I will sketch the state of the research area and go into detail on a few of my own related projects. An ancillary goal is to convey that programming or proving digital hardware is a lot like programming or proving software, with a few fun distinctions, so more of the CPP crowd might want to give it a try!

CCS Concepts • Hardware → Theorem proving and SAT solving; • Security and privacy → Side-channel analysis and countermeasures; • Computer systems organization → Architectures.

Keywords Proof assistants; formal verification; hardware-software interface; machine-language instruction sets; information leaks through side channels

ACM Reference Format:

Adam Chlipala. 2020. Proof Assistants at the Hardware-Software Interface (Invited Talk). In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '20)*, January 20–21, 2020, New Orleans, LA, USA. ACM, New York, NY, USA, 1 page. <https://doi.org/10.1145/3372885.3378575>

Biography

Started hacking on compilers & web-development tools in the late 1990s. Finished CS undergrad at Carnegie Mellon in 2003 and CS PhD at Berkeley in 2007, picking up mechanized proof of executable, decently practical systems with Coq as a main focus in between. Postdoc at Harvard through 2011, then faculty at MIT since. Author of *Certified Programming with Dependent Types*, a popular online & in-print introduction to using Coq at scale. Lately into building practical but clean-slate hardware-software stacks with end-to-end Coq proofs of everything digital, at the same time as developing a startup-company idea to trick ordinary people into using dependent types (with Ur/Web) to generate their business applications.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CPP '20, January 20–21, 2020, New Orleans, LA, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7097-4/20/01.

<https://doi.org/10.1145/3372885.3378575>