This experience report describes a style of applying symbolic model checking developed over the course of four years at Amazon Web Services (AWS). Lessons learned are drawn from proving properties of numerous C-based systems, e.g., custom hypervisors, encryption code, boot loaders, and the FreeRTOS operating system. Using our methodology, we find that we can prove the correctness of industrial low-level C-based systems with reasonable effort and predictability. Furthermore, AWS developers are increasingly writing their own formal specifications. All proofs discussed in this paper are publicly available on GitHub.

This is a paper about making code-level proof via model checking a routine part of the software development workflow in a large industrial organization. Formal verification of source code can have a significant positive impact on the quality of industrial code. In particular, formally verified specifications of code provide precise, machine-checked documentation for developers and consumers of a code base. They improve code quality by ensuring that the program?s implementation reflects the developer?s intent. Unlike testing, which can only validate code against a set of concrete inputs, formal proof can assure that the code is both secure and correct for all possible inputs.

The key obstacle to rapid proof development is that proofs tend to be written by a separate specialized team and not the software developers themselves. The developer writing a piece of code typically has an internal mental model of their code that explains why, and under what conditions, it is correct. However, this model typically remains known only to the developer? at best, it may be partially captured through informal code comments and design documents. As a result, the proof team must spend significant effort to reconstruct the formal specification of the code they are verifying. This slows the process of developing proofs.

Over the course of four years developing code-level proofs in Amazon Web Services (AWS), we have developed a proof methodology that is resulting proofs with reasonable effort, and projects whose time-lengths can be reasonably predicted. For example, using these techniques, one full-time verification engineer, plus two interns, were able to specify and verify 171 entry points over key 9 modules in the AWS C Common library over a period of 24 weeks. Our prediction was that the effort would take 25 weeks. All specifications, proofs, and related artifacts (such as continuous integration reports), described in this paper have been integrated into the main AWS C Common repository on GitHub, and are publicly available at https://github.com/awslabs/aws-c-common.

Daniel Schwartz-Narbonne is a Senior Applied Scientist in the AWS Automated Reasoning Group. Prior to joining Amazon, he earned a PhD at Princeton, where he developed a software framework to debug parallel programs. As a postdoc at New York University, he designed a tool that automatically isolates and explains the cause of crashes in C programs. At Amazon, he has been focusing on integrating formal reasoning into the industrial workflow, enabling the continuous verification of key AWS software. When he?s not working, you might find Daniel in the kitchen making dinner for his family, in a tent camping, or volunteering as an EMT with a local ambulance squad.
Other available formats:

**Code-Level Model Checking in the Software Development Workflow**

Switch to normal viewer
Switch to experimental viewer

[Presentation HCSS 2020](#)