Verifying C++ at Scale

Submitted by gmalecha on Fri, 08/14/2020 - 12:07pm. Contributor: Gregory Malecha

At BedRock Systems, we are developing and verifying a full virtualization stack from micro-hypervisor to user-land components developed on top of it using a formal-methods-first approach centered around modularity and automation. This ambitious task requires scaling in both the size and the complexity of the code. Formal-methods-first means that our Formal Methods team is working hand-in-hand with our systems developers to specify (and subsequently verify) the code. We have found that this process leads to more modular code that is easier to both specify and verify.

Applying formal methods to mainstream languages is crucial to expanding its adoption. Our verification infrastructure is built around modern C++ and leverages the clang compiler toolchain, allowing it to integrate with existing C++ code. Supporting C++ allows engineers to verify the code they write in a language that they are already familiar with, rather than forcing them to learn an entirely new, and often painfully minimalistic, programming language.

To support a modularity-first mentality, our tooling embraces separation logic, a logic built around disjointness of resources. Two crucial features of separation logic stand out as being particularly important for scaling verification.

• First, separation logic?s separate-by-default ?mentality? encourages thinking about modularity upfront leading to more verifiable code. In fact, we have found that many well-known patterns and anti-patterns can be justified by considering the specifications that hold on them. This lends credibility to the formal methods point of view when engaging with engineers that are new to formal methods.

• Second, separation logic?s expressivity allows us to apply it pervasively across all levels of our stack. The exact same principles for the user-land C++ code also apply to the kernel with page tables and embedded assembly. In the future, this will enable us to integrate with other languages, e.g. Rust or Java, that developers may wish to use on top of our stack.

While crucial, modularity is not a panacea for scale; automation is also essential. This is especially true in low-level languages such as C++ where code is often more explicit than it would be in higher-level languages. In conjunction with our verification efforts, we are building automation to reason about the more mundane aspects of verification (e.g. arithmetic reasoning) and have successfully used it to automatically verify some simple functions.

At the scale of our ambitions (which include weak memory concurrency), however, fully automated proofs are beyond the state of the art. To overcome this limitation, our tooling allows users to reason interactively about features that have not yet been automated such as loop invariants and complex concurrency. However, our automation is also highly customizable and we have already demonstrated some simple and reusable concurrent abstractions for automated proofs of concurrent code.

Gregory Malecha received his PhD in 2015 under the direction of Greg Morrisett and Adam Chlipala focusing on automation for program verification especially for separation logics. During his dissertation he worked on applying formal verification technology to verify infrastructure software such as a webserver and a simple database management system. After his degree, he worked on the verification of cyber-physical systems in the VeriDrone project with Sorin Lerner, proving safety properties in temporal logic. After a brief hiatus doing operations research and data science at Target Corp, he returned to the verification community and is currently developing verification technology and bringing academic insights on verification to the development of a full hypervisor stack as the director of formal methods at BedRock Systems.

License: Creative Commons 2.5