RESEARCH STATEMENT

Ronghui Gu (ronghui.gu@yale.edu)

My research goal is to make the software systems truly reliable and secure through *formal verification*. As the backbone of modern software systems, operating system (OS) kernels, on the one hand, can have the greatest impact on the reliability and security of today's computing host. On the other hand, OS kernels are complicated, highly-parallel, and prone to bugs. For the past several years, my research has focused on developing scalable tools to build verified sequential and concurrent OS kernels that are formally proved to be error-free and secure [POPL'15, OSDI'16, PLDI'16a, PLDI'16b]. Broadly speaking, my research falls into the subfield of programming languages that deals with the principles and practice of the formal verification of realistic software systems.

My Current Work on CertiKOS

While such mechanical "formal verification" can date back to the 1960s, complete formal proofs of sequential OS kernels only became feasible recently, demonstrated by seL4 in 2009. This result was so encouraging that it seemed only a mile away from a fully verified concurrent kernel with reasonable proof efforts. However, seven years have passed, this last mile is still insurmountable. Even in the sequential setting, the cost of such verification is quite prohibitive (seL4 took 12 person-years to develop), and it is unclear how to quickly adapt such a verified system to support new features and enforce richer properties. Furthermore, none of the previously verified systems have addressed the issues of concurrency.

We believe that these problems are caused by an overlook of the *layered structure* in the proofs. Although modern systems are implemented with multiple well-designed layers, this layered structure has not been exploited by the verification techniques like program logics: kernel modules across different layers and among multiple threads have to be reasoned about within the same abstraction level. It makes the system verification difficult to untangle and costly to extend.

My thesis research is among the first to address these challenges by exploring and realizing a novel class of specifications, named *deep specifications*, through layered approach. In theory, these layered deep specifications are rich enough to fully characterize the functionality of systems and uncover the insights of layered design patterns. In practice, they are "live" enough to connect with the actual implementation and provide a modular approach to building software system stacks that are entirely trustworthy. The advances in both dimensions have resulted in a comprehensive verification framework, named CertiKOS, and a series of fully verified sequential and concurrent OS kernels. This CertiKOS work wins an ACM Doctoral Award Nomination for my thesis research, inspires a decent research study of the science of deep specifications [DeepSpec], and was praised as "a real breakthrough" [YaleNews, YaleDailyNews].

Deep Specifications and Certified Abstraction Layers

One innovation of my thesis research is that OS kernels are treated as *run-time compilers* [POPL'15]. From this novel view, the OS kernel as a whole compiles the user programs that are understood using system call specifications to the programs that interact with the kernel implementation directly. We can view a layer in the kernel as a *compilation phase* and the kernel module between two layers as a *transformation*. In this way, OS kernels can be verified by showing that every such transformation preserves the behavior of *arbitrary context programs*. Due to the contextual preservation property, these layer interfaces are named as deep specifications, which captures the precise functionality of the implementation under any context.

As compilers enable program development in higher-level languages that are machine-independent, this layered approach allow program verification at some more abstract interfaces that are implementation-independent. Each kernel module is verified at its proper abstraction level by showing the contextual simulation relation between the implementation at that level and the deep specification at a higher level. Once this proof is done, the deep specification forms a new layer interface, which we call a *certified abstraction layer*. Any client program or any functional property of this module can be reasoned about using this more abstract layer specification alone, without going through the actual implementation.

To apply layer-based verification for real software systems, we develop the CertiKOS framework in the Coq proof assistant to specify, program, verify, and compose certified abstraction layers. With CertiKOS, the verification task of OS kernels can be mechanically decomposed into many small, simple, and independent tasks, which are fitting for manual proofs or automatable. To demonstrate the power of our framework, we have successfully developed multiple certified sequential OS kernels in Coq. The most realistic one is called mCertiKOS, which consists of 37 certified abstraction layers, took less than one person-year to develop, and can boot a version of Linux as a guest. An extended version of mCertiKOS was deployed on a military land vehicle in the context of a large DARPA-funded research project.

Verifying Concurrent OS Kernels with CertiKOS

Moving from the sequential kernel verification to the concurrent one is not straightforward at all. A concurrent kernel allows interleaved execution of kernel/user modules across different layers of abstraction. The complete formal verification of a non-trivial concurrent OS kernel is widely considered a grand challenge. Several researchers even believe that the combination of concurrency and the kernels' functional complexity makes the formal verification intractable, and even if it is possible, the cost would far exceed that of verifying a single-core sequential kernel.

We believe that the key to untangling this "intractable" complexity roots in the strong contextual property of deep specifications [OSDI'16]. A deep specification, in the concurrent setting, has to ensure the behavior preservation not only under any client context but also under any *concurrent context* with any interleaving. Each execution of the kernel corresponds to a concurrent context, which is represented as a list of *events* that encapsulates the behavior of the rest of CPUs (or threads), as well as the interference among them under this execution.

The *certified concurrent layer* is then parameterized over this concurrent context. Given a particular concurrent context, the interleaving is determined, and a concurrent layer is reduced to a sequential one, which allows us to apply standard techniques for verifying sequential programs to build new concurrent layers. A newly introduced concurrent layer becomes "deep" only after we show that its simulation proof holds for all valid concurrent contexts. To ease this step, we creatively lift the concurrent machine model (which allows arbitrary interleaving at any point) to an abstract local machine model, where all the impacts of the concurrent context are restricted at some certain points, i.e., the invocations of synchronization primitives. This machine-lifting idea enables the *local reasoning* of multicore (and multi-threaded) programs, without worrying about the concurrent interleaving except for a few specific places. Furthermore, this machine lifting also guarantees that the local reasoning of all the CPUs (and threads) can be composed together and then propagated down to the execution of actual implementations over the multicore hardware.

With this concurrent framework, we have also successfully developed and verified a practical concurrent OS kernel in Coq. Our certified kernel is written in 6,500 lines of C and x86 assembly and runs on stock x86 multicore machines. To our knowledge, this is the first formal verification of a complete, general-purpose concurrent OS kernel with fine-grained locking. As quoted from the Yale News, this is "a milestone that the scientists say could lead to a new generation of reliable and secure systems software" [YaleNews].

Adapt CertiKOS to Verify Interruptable OS Kernels and Device Drivers

Besides the power to build certified software systems from scratch, CertiKOS can also be quickly adapt to support new features. One convincing example is our CertiKOS extensions to verify device drivers [PLDI'16b]. In a monolithic kernel, device drivers are the majority of the code base, as well as the primary source of the system crashes. Although formal verification of device drivers is highly desirable, it is widely considered as challenging, due to the abundance of device features and the non-local effects of interrupts.

To address this issue, we introduce a general device model that can be instantiated with various hardware devices and a realistic interrupt model that scales the reasoning about interruptible code. The device drivers are modeled as if each of them were running on a separate *logical CPU*. This novel idea allows us to incrementally refine a raw device into more and more abstract devices by building certified layers of the relevant driver on its logical CPU. Meanwhile, this idea systematically enforces the isolation among different devices and the rest of the kernel. This strong isolation property leads to an abstract interrupt model such that most of the kernel execution is interrupt-unaware.

Thanks to these new models, we successfully turned mCertiKOS into a verified interruptible kernel with verified device drivers, e.g., serial, IOAPIC, etc. The entire extension is realized in Coq and took roughly seven person-months. To the best of our knowledge, this is the first verified interruptible operating system with device drivers.

End-to-End Security Verification in CertiKOS

Based on the contextually functional correctness guaranteed by CertiKOS, richer properties of the whole system can be derived with minimal costs. Take the security property as an example. Protecting the confidentiality of information manipulated by a software system is one of the critical challenges facing today's cybersecurity community. A promising step toward conquering this challenge is to verify the end-to-end information-flow security of the whole system formally. This step can be naturally established using our CertiKOS framework [PLDI'16a]. In CertiKOS, we only need to prove the noninterference between user processes at the top-most layer of the system, and the nature of deep specifications propagates this security guarantee down to the concrete implementation. The security guarantee derived using CertiKOS can be seen as end-to-end in the following two aspects: (1) it applies across all the layers, and (2) it ensures that the entire execution is secure from start to finish.

Future Research Agenda

The goal of my research is to integrate the efficient and scalable formal verification techniques into the development of real software systems and improve the software reliability and security. I plan to pursue this goal through the following research directions at different stages.

Programming Certified Software Systems Directly

Existing projects on real system verification, including CertiKOS, all require (manually) writing the actual implementation in a C-like language and a formal specification in a proof assistant language. Lacking the support of directly writing certified programs makes it difficult to develop and maintain certified software systems at scale. I aim to cooperate with *program synthesis* and *artificial intelligence* researchers to bridge this enormous gap between low-level system programming and high-level specification reasoning by providing a uniform way to program certified software directly.

This short-term goal is ambitious but still promising. Because the deep specification precisely captures the contextual functionality of the implementation, why not only write the layer specifications and then automatically generate the whole system from the layers? When focusing on a single verification task between two layers, the gap between the implementation and its deep specification, as well as the gap between two adjacent specifications are relatively small. I believe that we could generate the certified code from our deep specifications following the line of the program synthesis work and could link adjacent layers automatically by taking advantage of the recent progress in artificial intelligence. We have successfully synthesized a page allocator module consisting of four layers. But there is still a tremendous research opportunity for the complete functional synthesis and intelligent proof generator.

Apply Verification Techniques to Various Domains

OS kernel is not the only area that can benefit from our formal verification techniques. In the next five years, I aim to work on the reliability and security issues in the following domains:

- I plan to cooperate with *system researchers* to build a zero-vulnerability system stack consisting of verified components, such as device drivers, database system, file system, network stack, operating system, and distributed system. Although each of these components has been actively studied, most of them (e.g., file systems) only have sequential versions been verified, and there is a high demand to link all their guarantees together to form a trustworthy system stack. Our success on device driver extensions reveals a promising way to addressing these challenges. For different systems, we could provide "customized" machine models by exposing a particular set of hardware features and an abstract interface of the depending systems. It will enable a domain-specific approach to building certified layers for each system separately and all these layers can still be glued together.
- I plan to cooperate with *security researchers* to develop certified commercial-grade toolkit for security protocols. For those layer-based protocols (e.g., TLS and SSL), the layered verification approach will be a perfect fit. The certified concurrent layers can scale the existing security proofs by instantiating the widely-used oracle techniques with the concept of our concurrent context.
- I plan to cooperate with *cyber-physical system (CPS) researchers* to build high-confident CPS with real-time guarantee. Due to the physical consequences, it is highly desirable to prove that CPS behaves as required in terms of both functionality and timing. We have built an embedded system (an variant of mCertiKOS) that is proved to be functional correct and can run on a real drone. However, it is still a big challenge on how to formally model the time. We believe that the cure is still the event-based concurrent context. The flow of time can be represented as a list of special events, which will be indicated by a timing oracle that is enriched from the concurrent context. Thus, we can apply our concurrent verification techniques to prove real-time properties. I am looking forward to exploring this huge research opportunity when combining the formal verification with CPS.
- I plan to cooperate with *program analysis* researchers to establish a general and efficient specification-based testing framework. For non-critical software (e.g, user-level apps), random test is more applicable and productive than a complete formal verification. But how to generate test cases smartly is always challenging for concurrent programs. We believe that deep specifications can be utilized to inspire the test-case generation. Due to the "live" property, we can run these tests over the deep specifications directly without waiting for the responses from heavy workload operations. There is a huge research opportunity to exploring the testing framework based on deep specifications.

Pushing Towards an Industry-Scale Verification Framework

In the long term, I aim to extend our verification framework such that the zero-vulnerability system stack can scale to the industry level. We not only have to provide a practical and powerful set of standard certified libraries but also need to redesign the specification language such that they are not only rich and precise but also natural and simple for software

engineers. If successful, this research will make the real software systems truly reliable and secure, creating a profound impact on the software industry and the society in general.

References

[POPL'15] R. Gu, J. Koenig, T. Ramananandro, Z. Shao, X. Wu, S. Weng, H. Zhang, and Y. Guo. "Deep specifications and certified abstraction layers." In *42nd ACM Symposium on Principles of Programming Languages (POPL'15)*.

[OSDI'16] R. Gu, Z. Shao, H. Chen, X. Wu, J. Kim, V. Sjöberg, and D. Costanzo. "CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels." In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*.

[PLDI'16a] D. Costanzo, Z. Shao, and R. Gu. "End-to-end verification of information-flow security for C and assembly programs." In *37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'16)*.

[PLDI'16b] H. Chen, X. Wu, Z. Shao, J. Lockerman, and R. Gu. "Toward compositional verification of interruptible OS kernels and device drivers." In *37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'16)*.

[YaleNews] "CertiKOS: A breakthrough toward hacker-resistant operating systems." *Yale News*, 2016. http://news.yale.edu/2016/11/14/certikos-breakthrough-toward-hacker-resistant-operating-systems.

[YaleDailyNews] "Yale computer scientists unveil new OS." *Yale Daily News*, 2016. http://yaledailynews.com/blog/2016/11/18/yale-computer-scientists-unveil-new-os/.

[DeepSpec] DeepSpec: The science of deep specifications. http://deepspec.org/.