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" has a long history dating back to the 1960s, complete formal proofs of sequential OS kernels only became feasible recently, demonstrated by the seL4 verified microkernel in 2009. The result of seL4 was so encouraging that everyone believed it was only a mile away from a fully verified realistic and concurrent OS kernel under reasonable efforts. However, seven years have passed, and the situation is not likely to get better. The initial version of seL4 took 12 person-years to develop. Even until now, the cost of such verification is still quite prohibitive, and it is unclear how to quickly adapt such a verified system to support new features and enforce richer properties. Before our work, none of the existing verified systems have addressed the important issues of concurrency. This last mile once seemed insurmountable.

We believe that these verification challenges are caused by an overlook of the *layered structure* in the proofs. Although the system implementations consist of multiple well-designed layers, this layered structure is gone in the proof when using the verification techniques like program logics. Reasoning about the execution of kernel modules across different layers and among multiple threads is restricted within the same abstraction level. It makes the verification of real software systems 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 system implementations and uncover the fundamental insights of layered design patterns. In practice, they are "live" enough to connect with the actual implementation directly 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 that were praised as "a real breakthrough" [YaleNews, YaleDailyNews]. This CertiKOS work composes the main body of my thesis research, which is nominated for the ACM Doctoral Award, and inspires a decent research study of the science of deep specifications [DeepSpec].

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*. Since the preservation property holds for any context, these layer interfaces are named as deep specifications.

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 specification at a higher level. Once this proof is done, the 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 automatable or fitting for manual proofs. 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 the 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 kernel, 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 into the build of real software systems and dramatically improve the software reliability and security. I plan to pursue this goal through the following three research directions at different stages.

Programming Certified Software Systems with CertiKOS 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 fill in this enormous gap between low-level system programming and high-level specification reasoning by providing a uniform way to program certified software directly in CertiKOS.

This short term goal is ambitious but still promising. Due to the fact the all certified layers in CertiKOS are "deep", writing the layer specification alone is enough to express the full functionality. I plan to extend the CertiKOS framework such that the layer specification can be smartly "compiled" into C programs that meet the specification and the layers at different abstract level can be linked together in a mostly automated way. Thanks to the layered approach of CertiKOS, 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 the abstraction of two layer interfaces, are not that big. I believe that we could generate certified code from our deep specifications following the line of the program synthesis work and could link layers at different levels automatically by taking advantage of the recent progress in artificial intelligence[]. We have successfully generated the page allocator module of mCertiKOS. But there is still a tremendous research opportunity for complete functional synthesis and creating programs that are intelligent enough to do formal proofs between layers.

Building a Zero-Vulnerability System Stack

Once the verification framework allows writing certified programs in a more direct and efficient way, I aim to utilize it to build a system stack that is entirely trustworthy. Based on the hardware that supports cryptography primitives, I want to establish a reliable and secure stack that integrates the device drivers, database system, file system, network stacks, operating system, and trustable applications. This zero-vulnerability system stack will dramatically improve the reliability and security of computing hosts in the critical areas. However, there are two significant challenges in building such a stack: (1) systems at different levels may have various reliability requirements, some of which are still unclear how to they can be specified and verified (e.g., the network stacks); and (2) it is a big unknown how to link all the certified systems together in a single framework and provide the guarantee of the entire stack.

Our device driver verification work [PLDI'16b] shows that it is promising to address these challenges based on the CertiKOS framework. For different systems, we could provide "customized" machine models by exposing a particular set of hardware features plus an abstract interface of the underlying systems. It allows us to build certified layers for each system in an isolated way and the composition rules of CertiKOS could be strengthened to link all these layers. My Yale colleagues and I have already started to use CertiKOS to verify more systems in this stack, such as the file system and the network stacks. Meanwhile, I also work with a research group from University of Pennsylvania to set up a specification-based testing framework that enables the random test for user-level applications upon the topmost layer interface of mCertiKOS. I believe there is a huge research opportunity that lies in this zero-vulnerability system stack and I am looking forward to exploring it.

Pushing Towards an Industry-Scale Verification Framework

In the future, 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 it could precisely and efficiently describe the functionality and is acceptable by average engineers. If successful, this research will make the real software systems truly reliable and secure, making 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/.