RESEARCH STATEMENT

Ronghui Gu (ronghui.gu@yale.edu)

My research goal is to make the software systems truly reliable and secure through *formal verification*. Operating System (OS) kernels form the backbone of modern software systems. On one hand, OS kernels can have the greatest impact on the reliability and security of today's software systems. On the other hand, they are large, complex, 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. Broadly speaking, my research falls into the subfield of programming languages that deals with the principles and practice of the formal verification of real software systems.

Background and My Current Work on CertiKOS

While such mechanical "formal verification" has a long history dating back to the 1960s, it only became feasible recently to formally prove the functional correctness of practical systems, e.g., OS kernels [7], file systems [8], etc. However, 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. Furthermore, none of these verified systems have addressed the important issues of concurrency [9]. These problems severely limit the applicability and power of todays formally verified software systems.

My thesis research is among the first to untangle 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 behaviors of the real system implementation and uncover the fundamental insights in the layered design pattern of modern software systems. In practice, they are "live" enough to connect directly with the actual implementation and provide a modular approach to build 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" [5]. This CertiKOS work composes the main body of my thesis research, which is nominated for the ACM doctoral award, and even inspires a decent research study of the science of deep specifications [6].

Deep Specifications and Certified Abstraction Layers

Modern software systems (e.g., OS kernels, hypervisors, device drivers, network protocols) are designed and constructed using a stack of *abstraction layers*, each of which defines an interface that hides the underlying implementation details. Client programs can be understood solely based on the interface, independent of the layer implementation. Despite their huge contributions to the computer industry, abstraction layers have mostly been treated as a system concept; prior to our work, they have almost never been formally specified or verified.

My thesis research proposes a novel language-based account of abstraction layers [2], whose power lies in the use of a rich class of specifications, which we call *deep specifications*. A deep specification captures the precise functionality of the implementation under any client context. Once a system module is proved to meet its deep specification, 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 the 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 [10] to formally specify, program, verify, and compose certified abstraction layers. CertiKOS enables users to decompose this otherwise prohibitive verification task into many simple and automatable ones, each of which can be verified at its proper abstraction level. 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

Complete formal verification of a non-trivial concurrent OS kernel is widely considered a grand challenge. Concurrency allows interleaved execution of kernel/user modules across different layers of abstraction. Several researchers[] even believe that the combination of concurrency and the kernels functional complexity makes formal verification of functional correctness 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 untangle this "intractable" complexity roots in the strong contextual property of deep specifications [1]. A deep specification, in the concurrent setting, has to enforce that the implementation meets its specification not only under any client context, but also under any *concurrent context* with any interleaving. Each

execution of the system 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 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 multithreaded) 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.

Besides the beautiful theory of CertiKOS, in practice, 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 proof of (contextual) functional correctness and progress property 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" [5].

Adapt CertiKOS to Verify Interruptable OS Kernels and Device Drivers

In a monolithic kernel, device drivers are not only the majority of the code base[], but also the main 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 [4]. 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 own 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.

We have successfully extended the CertiKOS framework with these new models and turned mCertiKOS into a verified interruptible kernel with device drivers, e.g., serial, IOAPIC, etc. The entire extension is realized in Coq and took roughly 7 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

Protecting the confidentiality of information manipulated by a software system is one of the most important challenges facing todays cybersecurity community. A promising step toward conquering this challenge is to formally verify the end-to-end information-flow security of the whole system.

This step can be easily established using our CertiKOS framework [3]. An example is our work verifying that mCertiKOS is end-to-end secure. We proved the noninterference between user processes only at the top-most layer of mCertiKOS but the nature of deep specifications propagates this security guarantee down to the concrete implementation. The security guarantee of mCertiKOS can be seen as end-to-end in the following two ways: (1) it applies across all the layers; and (2) it guarantees 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.

Short Term: 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 huge 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 along the line of the program synthesis work[] and could link layers at different levels automatically by taking advantage of the recent progress on artificial intelligence[]. We have successfully generated the page allocator module of mCertiKOS. But there is still a huge research opportunity for complete functional synthesis and creating programs that are intelligent enough to do formal proofs between layers.

Intermediate Term: Building a Zero-Vulnerability System Stack

Once the verification framework allows to write certified programs in a more direct and efficient way, I aim to utilize it to build a system stack that is totally 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 major challenges to build such a stack: (1) systems at different levels may have various reliability requirements, some of which are still unclear how to specify and verify (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 [4] shows that it is promising to address these challenges based upon the CertiKOS framework. For different systems, we could provide "customized" machine models by exposing a certain set of hardware features plus an abstract interface of the underlying systems. It allows us to build certified layers for each system in an isolate way and the composition rules of CertiKOS could be strengthen 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 UPenn 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.

Long Term: 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 easily 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

[1] 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).

[2] 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).

[3] 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).

[4] 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).

[5] "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`.

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

[7] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. "seL4: Formal verification of an OS kernel." In 22nd ACM Symposium on Operating Systems Principles (SOSP 09).

[8] H. Chen, D. Ziegler, T. Chajed, A. Chlipala, M. F. Kaashoek, and N. Zeldovich. "Using Crash Hoare logic for certifying the FSCQ file system." In 25th ACM Symposium on Operating System Principles (SOSP 15).

[9] S. Peters, A. Danis, K. Elphinstone, and G. Heiser. "For a microkernel, a big lock is fine." In Asia Pacific Workshop on Systems (APSys 15).

[10] The Coq proof assistant. The Coq development team. `http://coq.inria.fr`.