RESEARCH STATEMENT

Ronghui Gu (ronghui.gu@yale.edu)

My research goal is to make critical software systems truly reliable and secure through *formal verification*. As the backbone of modern software systems, operating system (OS) kernels can have the greatest impact on the reliability and security of today's computing host. OS kernels, however, are complex, highly concurrent, and prone to bugs. For the past several years, my research has focused on investigating compositional programming language theories and 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 real software systems.

My Current Work on CertiKOS

My research approach is to explore clean and concise programming language theories that reveal the fundamental insights of system design patterns and system feature models and to apply these theories to build practical systems that behave as required through formal verification. 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 encouraging and it seemed not too far away from building a fully verified concurrent kernel under 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 11 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 ignoring the *layered structure* in the programming language theories. Although modern systems are implemented with well-designed layers, this layered structure has not been exploited by the verification techniques like program logics: modules across different layers and multiple threads have to be reasoned about within the same abstraction level. It makes the system verification hard to untangle and costly to extend.

My thesis research is among the first to address these challenges by advocating and realizing a novel class of specifications, named *deep specifications*, through a layered approach. In theory, these layered deep specifications uncover the insights of layered design patterns and are rich enough to fully characterize the system's functionality. 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 is used in high-profile DARPA programs [CRASH, HACMS], is the core component of an NSF Expeditions in Computing project [DeepSpec], and has been widely considered "a real breakthrough" toward hacker-resistant systems [YaleNews, IBTimes, YDN].

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 *compiler transformation*. In this way, OS kernels can be verified by showing that every such transformation preserves the behavior of *arbitrary context programs* and all the compilation phases can be composed into a *compositional compiler*. Due to the contextual preservation property, these layer interfaces are named as deep specifications, which precisely capture the functionality of the implementation under any context.

As compilers enable program development in higher-level and machine-independent languages, this layer-based theory allows program verification to be done over some more abstract interfaces that are implementation independent. Each module is verified at its proper abstraction level by showing the contextual simulation 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 this layer-based theory for the verification of real software systems, we developed the CertiKOS framework to specify, verify, and compose certified abstraction layers in the Coq proof assistant. We also extended the CompCert verified compiler to compile certified C layers such that they can be linked with assembly 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 high-profiled DARPA program [HACMS].

Verifying Concurrent OS Kernels with CertiKOS

Moving from the sequential kernel verification to the concurrent one is not straightforward at all and requires a more powerful compositional theory. A concurrent kernel allows arbitrarily 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 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. A concurrent context corresponds to a particular execution of the kernel and is represented as a list of *events* that encapsulates the behaviors 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, and a new layer-based theory is developed to compose certified concurrent layers with valid concurrent contexts. Given a particular concurrent context, the interleaving is determined, and a concurrent layer is reduced to a sequential one, which allows us to apply sequential verification techniques for building 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., just before the invocations of synchronization primitives. The theory of this machine lifting enables the *local reasoning* of multicore (and multithreaded) programs, without worrying about the interleaving except for a few specific places. Furthermore, based on this machine-lifting theory, we introduce CompCertX, i.e., a thread-safe version of the CompCert compiler, to compile concurrent C layers into assembly ones and propagate the guarantees down to the assembly-level executions over the multicore hardware.

With this concurrent layer-based theory, 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 fully verified concurrent OS kernel with fine-grained locking. This work is recognized as "a milestone that the scientists say could lead to a new generation of reliable and secure systems software" [YaleNews].

Adapt CertiKOS to Verify Interruptible OS Kernels and Device Drivers

Besides the power to build certified software systems from scratch, our layer-based theory makes it efficient to introduce new system features. One convincing example is our CertiKOS extensions for verifying 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 extend the certified layer with 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. The strong isolation property leads to an abstract interrupt model such that most of the kernel execution is interrupt-unaware.

Thanks to the layer-based theory equipped with these new models, we successfully turned mCertiKOS into an interruptible kernel with verified device drivers, e.g., serial, IOAPIC, etc. The entire extension took seven person-months to implement in Coq. To the best of our knowledge, this is the first verified interruptible OS kernel with device drivers.

End-to-End Security Verification in CertiKOS

In the layer-based theory, richer properties of the whole system can be directly derived from the deep specifications. 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 contextual property of deep specifications propagates this security guarantee down to the level of the concrete implementation. This security guarantee 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 clean and concise programming language theories into the development of formally verified software systems and make them reliable and secure. 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 writing certified programs directly makes the certified software systems difficult to develop and maintain. I plan to work with *program synthesis* and *artificial intelligence* researchers to bridge this enormous gap between the low-level system programming and the high-level specification reasoning through a uniform framework that enables programming 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 write the layer specifications alone and generate the whole system from the layers automatically? 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. We have already synthesized a page allocator module consisting of four layers. I aim to exploit this uniform framework into more depth by attempting to generate a complete kernel from layer specifications following the line of the program synthesis work and link adjacent layers automatically by taking advantage of the recent progress in the artificial intelligence research.

Apply Verification Theories to Various Domains

OS kernel is not the only area that can benefit from the formal verification techniques. In the next five years, I will further investigate more powerful and more compositional verification theories and apply them in the following domains:

- I plan to work 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 may provide "customized" machine models by exposing an interface that abstracts a particular set of system features. It will enable a domain-specific approach to building certified layers for each system separately. The major issue is how to link together these layers with various reliability requirements and different system features with a single computational theory. I am looking forward to exploring this huge research opportunity.
- I plan to work 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 concept of our certified concurrent layers can scale the existing security proofs by instantiating the widely-used oracle techniques with the concurrent context. The theoretical challenge that I have to solve is how to preserve the security guarantees under the simulation with concurrent context.
- I plan to work with *cyber-physical system (CPS) researchers* to build high-confident CPS with the real-time assurance. 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 (a variant of mCertiKOS) that is proved to be functionally correct and can run on a real drone. But the most difficult problem left is how to formally model the time in theory. My proposal is to abstract the flow of time as a list of special events, which will be indicated by a timing oracle that is enriched from the concurrent context. With this model, we can apply our concurrent verification techniques to prove real-time properties. I hope to explore this timing theory in the near future.
- I plan to work with *program analysis* researchers to establish a general and efficient specification-based testing framework. For concurrent software (e.g., user-level apps) without high-reliability requirements, the complete formal verification is unnecessary and too expensive, while the random testing is difficult to detect concurrent-bugs caused by some certain interleaving. We plan to solve this research problem by utilizing deep specifications to generate the test cases smartly in the concurrent setting. Due to the "live" property, we may also run these tests over the deep specifications directly, without waiting for the responses from heavy-workload operations.

Pushing Towards an Industry-Scale Verification Framework

In the long term, I aim to scale our verification techniques to the industry level. To achieve that, we not only have to provide a practical and powerful set of standard certified libraries but also need to make a breakthrough in the theory of specifications. I aim to make the deep specifications not only rich and precise but also natural and straightforward 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)*.

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

[HACMS] High-Assurance Cyber Military Systems (HACMS). http://opencatalog.darpa.mil/HACMS.html.

[CRASH] Clean-slate design of Resilient, Adaptive, Secure Hosts (CRASH). http://www.darpa.mil/program/clean-slate-design-of-resilient-adaptive-secure-hosts.

[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.

[IBTimes] "CertiKOS: Yale develops world's first hacker-resistant operating system." *Internaltional Business Times*, 2016. http://www.ibtimes.co.uk/certikos-yale-develops-worlds-first-hacker-resistant-operating-system-1591712.

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