From Hoare Logic to Owicki-Gries and Rely-Guarantee for Interruptible eChronos and Multicore seL4 (Extended Abstract)

June Andronick

Data61, CSIRO (formerly NICTA) and UNSW, Sydney, Australia
`june.andronick@data61.csiro.au`

In this talk we will be exploring the use of foundational proof techniques in the formal verification of real-world operating system (OS) kernels. We will focus on eChronos [2], a small interruptible real-time OS, and seL4 [7,8], the landmark verified microkernel, currently undergoing verification of its multicore version. Both are deployed in various safety- and security-critical areas, and present challenging complexities due to their performance constraints and concurrency behavior. Foundational techniques have been and are being used for their verification, ranging for standard Hoare logic [5], to concurrency logics like Owicki-Gries [10] and Rely-Guarantee [6]. We will describe their use and combination with theorem proving and automation techniques to achieve impact on large-scale software.

Hoare logic is well known to be the foundation of formal verification for mainstream programs. It is what is taught to university students to prove formally the correctness of programs. Hoare logic can also be the basis of large-scale, real-world software verification, such as the verified seL4 microkernel. seL4 is a very small OS kernel, the core and most critical part of any software system. It provides minimal hardware abstractions and communication mechanisms to applications. seL4 additionally enforces strong access control: applications can be configured to have precise rights to access memory or to communicate, and seL4 guarantees the absence of unauthorised accesses. seL4 has undergone extensive formal verification [7,8] when running on unicore hardware. The central piece of this verification is the proof of functional correctness: that seL4 source code satisfies its specification. This proof uses Hoare logic at its core, while the top-level theorem is a traditional refinement proof through forward simulation: we show that all behaviors of the source program are contained in the behaviors of the specification. For small programs, Hoare logic can be the central method to prove functional correctness, where the specification is defined as being the description of the state in the postcondition. For larger programs, and in particular for programs where further verification is desired (like seL4's further security proofs), having the specification as a separate standalone artifact saves significant overall effort. In this case a refinement proof links the concrete source code to the abstract specification, and often relies on global invariants to be maintained. In seL4 verification, invariant proofs represent the largest part of the effort [8]. They heavily use Hoare logic reasoning, combined with important use of automation in the Isabelle/HOL theorem prover [9], both to generate the required invariant statements for each of the hundreds of seL4 functions and to discharge as many as possible without need for human interaction.

Following this verification of a large and complex, but sequential program, we investigated the impact of concurrency in settings where interrupts cannot be avoided (seL4 runs with interrupts mostly disabled), or where running on multiple processors is desired.

Reasoning about interrupt-induced concurrency is motivated by our verification of the eChronos [2] embedded OS. In an eChronos-based system, the kernel runs with interrupts enabled, even during scheduling operations, to be able to satisfy stringent latency requirements. The additional challenge in its concurrency reasoning is that racy access to shared state between the scheduler and interrupt handlers is allowed, and can indeed occur.

The modelling and verification approach we chose for this fine-grained concurrency reasoning is Owicki-Gries [10], the simple extension on Hoare logic with parallel composition and *await* statements for synchronisation. Owicki-Gries provided the low-level of abstraction needed for the high-performance shared-variable system code we were verifying. We could conveniently identify localised Owicki-Gries assertions at the points of the racy accesses, and tune them to enforce the overall correctness invariant of eChronos scheduler. In contrast, the Rely-Guarantee (RG) approach [6] would have required identification of global interference conditions, which was challenging for such racy sharing with no clear interface, unless we made heavier use of auxiliary variables to identify racy sections of code, but this defeats the compositionality of the RG approach, one of its principal purposes. The explosion of verification conditions inherent in the Owicki-Gries approach has been minimized by the controlled nature of the interrupt-induced concurrency, and mitigated by proof-engineering techniques and automation of a modern theorem prover. We were able to develop an abstract model of eChronos scheduling behavior and prove its main scheduling property: that the running task is always the highest-priority runnable task [4,3]. Our models and proofs are available online [1].

We are currently exploring multicore-induced concurrency for seL4 in a setting where most but not all of the code is running under a big lock. Here we have explored the RG approach, on an abstracted model identifying the allowed interleaving between cores. In this setting, the relies and guarantees can express what shared state the lock is protecting, and what the conditions are under which shared state can be accessed without holding the lock. The main challenge is resource reuse. The kernel runs in privileged mode, and as such has access to everything; it can for instance delete objects on other cores to which critical registers point. This could create a system crash if later on in that core, the kernel code accesses these registers pointing to corrupted memory. Designs to solve this issue include forcing kernel operations on all other cores without holding the lock. The proof that this is sound needs to be expressed via relies and guarantees between cores. We proved, on our abstract model of the multicore seL4-system, that critical registers remain valid at all times.

The main challenge now, for both the eChronos verification and multicore seL4 one, is to transfer the verification down to the source code via refinement.

Acknowledgements.
The author would like to thank the people that have worked on the research presented in this paper: Sidney Amani, Maksym Bortin, Gerwin Klein, Corey Lewis, Daniel Matichuk, Carroll Morgan and Christine Rizkallah. The author also thanks Carroll Morgan and Gerwin Klein for their feedback on drafts of this paper.
Parts of the work presented are supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) and U.S. Army International Technology Center - Pacific under grant FA2386-15-1-4055.

References

1. eChronos model and proofs, `https://github.com/echronos/echronos-proofs`
2. The eChronos OS, `http://echronos.systems`
3. Andronick, J., Lewis, C., Matichuk, D., Morgan, C., Rizkallah, C.: Proof of OS scheduling behavior in the presence of interrupt-induced concurrency. In: Jasmin Christian Blanchette and Stephan Merz (ed.) ITP. pp. 52–68. Springer, Nancy, France (Aug 2016)
4. Andronick, J., Lewis, C., Morgan, C.: Controlled owicki-gries concurrency: Reasoning about the preemptible eChronos embedded operating system. In: Rob J. van Glabbeek and Jan Friso Groote and Peter Höfner (ed.) Workshop on Models for Formal Analysis of Real Systems (MARS 2015). pp. 10–24. Suva, Fiji (Nov 2015)
5. Hoare, C.A.R.: An axiomatic basis for computer programming. CACM 12, 576–580 (1969)
6. Jones, C.B.: Tentative steps towards a development method for interfering programs. Trans. Progr. Lang. & Syst. 5(4), 596–619 (1983)
7. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal verification of an operating-system kernel. CACM 53(6), 107–115 (Jun 2010)
8. Klein, G., Andronick, J., Elphinstone, K., Murray, T., Sewell, T., Kolanski, R., Heiser, G.: Comprehensive formal verification of an OS microkernel. Trans. Comp. Syst. 32(1), 2:1–2:70 (Feb 2014)
9. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
10. Owicki, S., Gries, D.: An axiomatic proof technique for parallel programs. Acta Informatica 6, 319–340 (1976)