Combining Testing With Formal Verification in COGENT

(Talk Proposal)

Zilin Chen

CSIRO's Data61 and UNSW Sydney

Zilin.Chen@data61.csiro.au

COGENT [4] is a purely functional programming language for writing and formally verifying systems code. The certifying COGENT compiler takes a COGENT source program and produces an efficient C implementation (via a uniqueness type system), along with an Isabelle/HOL proof showing that the C code refines the semantics of the COGENT source program. COGENT is a restricted language: it does not have built-in support for loops or recursions, and cannot allocate or free memory dynamically. To compensate for expressiveness, a foreign function interface (FFI) to C is provided. The portion written in C will have to be manually verified against some functional correctness specification. Typically, systems programmers define their (usually recursive) abstract data types (e.g. linked lists, red-black trees) and glue code between COGENT and the operating system (OS) kernel in C. The former can usually be shared among multiple implementations, amortising the cost of manual verification. COGENT has been proved to be a viable approach for real-world systems programming. We used COGENT for developing file systems and device drivers.

Traditionally, testing is perhaps the most widely used method for ensuring the correntness of the programs. Even with formal methods emerging, testing should not be replaced by formal verification, espcially during the development phase. Testing is a useful complement to formal verification: (1) It is well known that formal verification is very costly (e.g. the formal proof of a $\sim$8.7k-line (in C) seL4 microkernel took about 11 person years [3]) and the work usually requires a long time span to finish. Before the full verification is complete, or when the specification changes, testing provides some degree of confidence in the correctness of the software in question. (2) For high-assurance applications, both formal verification and testing are required by many international standards (e.g. the Common Criteria for Information Technology Security Evaluation EAL7 [1]). Technically, there are always some aspects of the system which have to be assumed in formal proofs, and testing can potentially uncover bugs in the trusted portion of the system and tools used. (3) The systems implementations (even COGENT ones) are normally written by systems engineers, who have rich experience with the low-level behaviours of the OS, whereas the formal specification and the proofs are developed by formal verification experts, who do not necessarily read low-level programs a lot and understand systems programming idiomatics very much. Tests, if specified semi-formally, can serve as a communication protocol between these two groups of experts.

In this talk, I will give a high-level introduction on how we combine formal verification and testing in the context of COGENT [1], discuss various testing methodologies that are potentially beneficial, and discuss lessons learned from our own experience. In particular, I will show how property-based testing à la QuickCheck [2] enables an incremental approach to a fully verified systems and provides an effective interface between proof engineers and programmers that helps obtain a verification-friendly design and the technical challenges we encountered during this process.

References

[1] Zilin Chen, Liam O'Connor, Gabi Keller, Gerwin Klein, and Gernot Heiser. 2017. The Cogent Case for Property-Based Testing. In *Workshop on Programming Languages and Operating Systems (PLOS)*. ACM, Shanghai, China, 1–7.

[2] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the 5th International Conference on Functional Programming*. 268–279.

[3] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: Formal Verification of an OS Kernel. In *ACM Symposium on Operating Systems Principles*. ACM, Big Sky, MT, USA, 207–220.

[4] Liam O'Connor, Zilin Chen, Christine Rizkallah, Sidney Amani, Japheth Lim, Toby Murray, Yutaka Nagashima, Thomas Sewell, and Gerwin Klein. 2016. Refinement Through Restraint: Bringing Down the Cost of Verification. In *International Conference on Functional Programming*. Nara, Japan.

[1] `https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf`