A Data Layout Description Language for COGENT

Christine Rizkallah
UNSW, Sydney, Australia

While purely functional languages allow for reasoning about code equationally and productively, they often do not give systems programmers sufficiently fine-grained control for achieving the level of efficiency they desire. Our aim is to reduce the effort required for producing reliable, efficient systems.

COGENT [4] is a restricted uniqueness-typed purely functional language for writing high-assurance systems code [1]. COGENT has a certifying compiler that generates efficient C code [4, 5] by making use of COGENT's uniqueness types.

COGENT programs do not exist in isolation. Typically, a COGENT program constitutes a component of a larger system, written in C, which is connected to the COGENT program using a foreign function interface (FFI). The aim, when building a system using COGENT, is to write as much of it in COGENT as possible, because the effort needed to verify low level imperative C code is significantly higher than that needed to verify COGENT code.

COGENT programs are defined as pure functions operating on *algebraic data types*. The exact layout of these data types in memory is determined by the COGENT compiler. Many of the data structures in operating systems such as Linux could be represented as algebraic types, however their exact memory layout differs from that used by our COGENT compiler.

Therefore, the systems written in COGENT must maintain a great deal of glue code to synchronise between two copies of the same conceptual data structure [1]. As COGENT code can only interact with the COGENT data representation, this glue code is currently written in C. This code is tedious to write, wasteful of memory, prone to bugs, has a significant performance cost, and requires cumbersome manual verification at a low level of abstraction. To solve these issues we want to be able to write this type of code in COGENT.

To do so, we propose a new framework that allows for data abstraction in COGENT programs. Rather than maintaining two copies of data, we define a data description language, DARGENT, to describe the correspondence between COGENT algebraic data types and the bits and bytes of kernel data structures — what we call the *layout* of the data. With DARGENT, the programmer can write code as usual, manipulating ordinary COGENT data types, and after compilation the generated C code will manipulate kernel data structures directly, without extensive copying and synchronisation at run-time.

This will improve performance by eliminating redundant code, simplifying the integration of C and COGENT code, and enabling users to write and verify more COGENT code rather than reasoning about cumbersome C code. DARGENT eliminates the need for a standalone language for marshalling and unmarshalling data (such as PADS [3], Nail [2]). Moreover, it allows programmers a level of fine-grained control over memory usage similar to that provided by C.

So far, we have designed DARGENT, implemented some of the compilation phases, and formalised our DARGENT prototype design in Agda. We have also implemented some essential extensions to the COGENT language to accommodate the data layout descriptions.

This talk will introduce DARGENT, a language that is still under development and that describes how a COGENT algebraic data type may be laid out in memory, down to the bit level. Data descriptions in DARGENT will influence the generated definitions and proofs that constitute the compilation certificate between COGENT and the generated C code.

References

[1] Sidney Amani, Alex Hixon, Zilin Chen, Christine Rizkallah, Peter Chubb, Liam O'Connor, Joel Beeren, Yutaka Nagashima, Japheth Lim, Thomas Sewell, Joseph Tuong, Gabriele Keller, Toby Murray, Gerwin Klein, and Gernot Heiser. 2016. Cogent: Verifying High-Assurance File System Implementations. In *International Conference on Architectural Support for Programming Languages and Operating Systems*. Atlanta, GA, USA, 175–188.

[2] Julian Bangert and Nickolai Zeldovich. 2014. Nail: A Practical Tool for Parsing and Generating Data Formats. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Broomfield, CO, 615–628. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/bangert

[3] Kathleen Fisher and David Walker. 2011. The PADS project: an overview. In *Proceedings of the 14th International Conference on Database Theory*. ACM, New York, NY, USA, 11–17. http://doi.acm.org/10.1145/1938551.1938556

[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.

[5] Christine Rizkallah, Japheth Lim, Yutaka Nagashima, Thomas Sewell, Zilin Chen, Liam O'Connor, Toby Murray, Gabriele Keller, and Gerwin Klein. 2016. A Framework for the Automatic Formal Verification of Refinement from Cogent to C. In *International Conference on Interactive Theorem Proving*. Nancy, France.

Sapling 2019, November 15, 2019, Sydney, Australia