Using Dependent Types to Port Type Systems to Low-Level Languages

George Necula

University of California, Berkeley, USA

A major difficulty when trying to apply high-level type systems to low-level languages is that we must reason about relationships between values. For example, in a low-level implementation of object-oriented dynamic dispatch we must ensure that the “self” argument passed to the method is the same object from whose virtual table we fetched the pointer to the method. Similarly, in low-level code using arrays we must relate the array address with the variables that store the bounds. We show for several examples that the high-level type system must be extended with dependent types in order to reason about low-level code. The novel feature in this use of dependent types is that they can be used in presence of pointers and mutation.

We discuss three case studies. First, we show a variant of bytecode verification that operates on the assembly language output of a native code compiler. Second, we show how to express and check at the assembly level the invariants enforced by CCured, a source-level instrumentation tool that guarantees type safety in legacy C programs. Finally, we show that dependent types are a natural specification mechanism for enforcing common safe programming practices in C programs. We have used this mechanism to efficiently enforce memory safety for several Linux device drivers.