Abstract

Compilers should not miscompile. Our work addresses problems in developing peephole optimizations that perform local rewriting to improve the efficiency of LLVM code. These optimizations are individually difficult to get right, particularly in the presence of undefined behavior; taken together they represent a persistent source of bugs. "Provably Correct Peephole Optimizations with Alive" presents a domain-specific language for writing optimizations and for automatically either proving them correct or else generating counterexamples. Furthermore, Alive can be automatically translated into C++ code that is suitable for inclusion in an LLVM optimization pass. Alive is based on an attempt to balance usability and formal methods; for example, it captures—but largely hides—the detailed semantics of three different kinds of undefined behavior in LLVM. We have translated more than 300 LLVM optimizations into Alive and, in the process, found that eight of them were wrong.

"Termination-Checking for LLVM Peephole Optimizations" identifies a new class of bugs with peephole optimizations: non-termination bugs. When a suite of peephole optimizations is executed until a fixed point, an optimization which undoes the effect of other optimizations will result in non-terminating compilation. This paper (1) proposes a methodology to detect non-termination bugs with peephole optimizations, (2) identifies the necessary condition to ensure termination while composing peephole optimizations, and (3) provides debugging support by generating concrete input programs that cause non-terminating compilation. We have discovered 184 optimization sequences, involving 38 optimizations, that cause non-terminating compilation in LLVM with Alive generated C++ code.

Building on these results, we hope to extend Alive with support for floating-point and vector instructions. We also plan to examine support for optimizations over different intermediate languages, and for classes of optimizations not presently expressible in Alive.