Generating Compiler Optimisations

Richard Warburton

May 29, 2008

Compilers are commonly used programs that translate source code into object code. It is possible for a compiler to introduce a bug into the program that it is compiling, if it changes the semantics of program during this translation. Ensuring that compilers preserve the semantics of the source code during this translation is a non trivial task, due to the difficulty of finding an effective testing approach. This difficulty is further exacerbated since modern compilers optimise the program whilst translating it.
Due to the well defined semantics of both input and output, and the common usage of compilers, a formal approach is well suited to the situation. Existing literature has demonstrated the effectiveness of formal methods in verifying simple compilers for trivial languages. Furthermore existing literature also describes the verification of some dataflow analyses that check the safety and applicability of compiler optimisations.
A common approach to specifying compiler optimisations is to use a domain specific language (DSL), with a formally defined semantics. When applying optimisations specified within a DSL one needs to consider two properties: efficiency, the time taken to apply an optimisation to a program, and effectiveness, to what extent is the performance of the program improved by the optimisation.
We use the TRANS language that can specify a large catalog of optimisations through the use of temporal logic and rewrite rules. Specifiable optimisations include dead code elimination, constant propagation, strength reduction, branch elimination, skip elimination, loop fusion, partial redundancy elimination, and lazy strength reduction and lazy code motion. A compilation strategy for the TRANS language is described that produces an optimisation phase for the Soot compiler framework for a given optimisation specification. An implementation is described and demonstrated that generates optimisation phases that can be efficiently applied to Java bytecode. Efficiency claims are examined up by empirical evaluation using the industry leading Spec JVM benchmark.