VYPe – Modern Compiler Design

Jan Kalina, xkalin03@stud.fit.vutbr.cz
Tomáš Trkal, xtrkal00@stud.fit.vutbr.cz

November 6, 2015

One of the modern compilation methods, which is applied at compiler writing time and which is targeted on obtaining better rewriting templates, is technique called *Supercompilation.* The main idea of supercompilation is creation of optimal code sequences for some simple but useful functions by trying combinations of instructions from a suitable set of machine instructions. At first, all combinations of two instructions are tried. If no combination performs the required function all combinations of three instructions are tried, and so on, until we find an appropriate combination that works. Supercompilation is still an experimental technique at the moment, but it is an example of original thinking and it yields surprising results.

Modern code generators often use tens and sometimes hundreds of optimization techniques and tricks, each of which can in principle interfere with each of the other optimizations. However, if we find out that a program fails when compiled with optimizations and runs correctly when compiled without them, it does not necessarily mean that the error is in the optimizer, because the program may be incorrect in a way that depends on details of the compilation. It is important to have a flag allowing the optimization to be performed or skipped, because this allows selective testing of the optimizations and any of their combinations.

Preprocessing the intermediate code involves *preprocessing of expressions*, *preprocessing of if-statements and goto statements* and *preprocessing of routines.* Preprocessing of expression techniques include optimizations *constant folding* and *arithmetic simplification*, which are performed during construction of an AST. Transformations that replace an operations by a simpler ones are called *strength reductions* and operations that can be removed completely are called *null sequences. Dead code elimination* belongs to methods of preprocessing of if-statements and goto statements. Preprocessing of routines include processes *in-lining* and *cloning.*

Unfortunately, even moderately sophisticated code generation techniques can produce pointless machine instruction sequences. These inefficient sequences of symbolic machine instructions may be replaced by more efficient sequences using the *peephole optimization.* Peephole optimizations consist in creating *replacement patterns* and locating and replacing suitable sequences of instructions by applying these replacement patterns.

The result of the compilation is usually a generated executable file. However, writing code for straightforward conversion of the machine code into the object file could be an error-prone work, therefore sometimes generating of symbolic assembly code is preferred. The assembly code is converted into object files by *assembler.* Assembler converts symbolic instructions to a binary machine code and symbolic data to binary data and this pack into the object file. Some non-imperative languages (functional, logical, distributed) compilers even use a higher-level language like C/C++ as the output code, which ensures platform independence.

References

[1] Dick Grune. Kees van Reeuwijk. Koen Langendoen. *Modern Compiler Design.* Springer-Verlag, New York, second edition, 2012.