The design and semantics of COOL

Matej Minárik (xminar29@stud.fit.vutbr.cz)
Petr Nechvátal (xnechv05@stud.fit.vutbr.cz)
06. 11. 2015

Compiler course is taken every year by many undergraduate students around the world. A substantial project, where students write a compiler, is typically part of every compiler course. This project teaches students some basics of language design, but also gives them a chance to implement a complex program. It is very time consuming and labour intensive for students to implement a compiler from scratch in one semester.

It is challenging also for course staff to make moderate changes every year, implement and test a reference program. It is very surprising that there is no framework to speed up creation and implementation of compiler project assignments as it is for example in teaching operating systems the nachos project.

Here comes the Cool, free available, portable compiler project. It has been used at Berkley for several years and now it can be used freely by others as well. Cool is also a programming language, which is object-oriented, statically typed with automatic memory management, similar to Java. Cool was designed to be easily compiled, rather than easily used by programmers, which is exactly what is needed. It is extremely modular, so that students that do a poor job on one assignment are not at disadvantage on other ones. Cool compiler is structured in 4 modules: lexical analyzer, parser, semantic analyzer and code generator. Students can implement, for example, their own parser and compile the whole compiler with reference modules and their parser, if they respect defined interfaces.

Cool project can be used on any Unix machine with standard gnu tools (flex, bison, gmake). Generated MIPS assembly code can run on a spim simulator included in the project. There is also supporting code with memory management and data structures that provides some level of abstraction, so students can focus on compiler implementation instead of investigating and fixing bugs not related to compiler construction. The whole project is well documented. Documentation has a formal part of the language and informal part. Supporting code with its interfaces and examples is also documented.