Selective increase in LL lookahead as an alternative to factorization

Jakub Kocalka, xkocal00@fit.vutbr.cz

Factorization is a commonly used technique for transforming context-free grammars to LL(1) grammars. This transformation introduces new nonterminals into the grammar. While the original grammar was (hopefully) designed in a readable way, with all nonterminals having clear semantic functions, these new nonterminals are obscure, and break rules apart. This can cause the parser code to become less readable, can make it more difficult to assign semantic actions to rules, and produces larger derivation trees. However, factorization is equivalent to increasing the number of tokens the parser looks at during parsing, and can be avoided by using a LL(k) parser.

Increasing the number of lookahead tokens an LL(k) parser needs to parse a language increases the size of the parsing table substantially. This is problematic both during the construction and during parsing when using the table. However, in the case of factorization, we can predict in which cases we will actually need to look at more than one token. Furthermore, recursive descent parsers don't actually need to hold a parsing table. Selectively requesting multiple tokens from the scanner instead of introducing obscure nonterminals (and therefore functions in the parser code) can make these parsers much cleaner.

In this article, we present an algorithm for constructing a partially LL(k) table by avoiding factorization. Next, we explore the effect this parsing algorithm will have on the size of the resulting parsing table compared to factorization. Finally, we provide an example recursive descent parser using this new parsing algorithm, and compare it to a parser for a factorized grammar.