

Adapser: an LALR(1) Adaptive Parser

Adam Carmi

Department of Computer Science
Technion - Israel Institute of Technology

15 May 2002

An adaptive parser is a parser that is capable of adapting its parse tables and parse stack to incremental and decremental grammar modifications while parsing. Adapser is a C++ template library that implements an adaptive LALR(1) parser; it is designed to be used as a framework for constructing compilers that support parse-time grammar modification.

Conventional compiler construction tools, such as YACC[1], provide compiler writers with an easy and straightforward way to perform syntactic and semantic analysis of programs. Users of such tools first create a grammar for the language to be compiled. The grammar is then processed by the tool that outputs a parser suitable for inclusion with other code to produce a compiler for the language. Whenever a need to change the grammar of the language arises, the entire process must be repeated in order to produce a completely new compiler.

Furthermore, the design of such tools (often referred to as *parser generators* or *compiler-compilers*) is usually based on the assumption that the generated parsers are used many times so that the time required to generate them is not a critical factor. As a result, inefficient parser generators are considered satisfactory as long as they generate efficient parsers.

There are, however, two major drawbacks to the approach sketched above. The first is that parsers generated using this strategy are inherently unable to parse extensible languages; that is, languages that provide constructs for extending their own grammar. Extensible languages offer the ability to reduce the ‘conceptual distance’ between the base general-purpose language and the application domain by allowing the extension of the language with concepts and constructs from the application domain. Although such languages are scarce, one may argue that the unavailability of construction tools for generating the corresponding compilers has made them such.

A second drawback stems from the fact that there are situations where the efficiency of the parser generator itself is crucial. For example, consider a language in development stages. After each change of the grammar a completely new parser must be generated and there is no guarantee that it will be used sufficiently often. It is clear that in such cases, most computational effort is wasted on repeated generation of parsers rather than on parsing.

To overcome these drawbacks, a different approach was taken in the development of Adapser. Unlike traditional compiler construction tools, Adapser is a hybrid of a parser and a parser generator. It starts off with an empty grammar and a corresponding empty parse table, and allows on the fly incremental and decremental changes to its grammar, which translate into parse-time correction of its parse table and parse stack.

In order to handle grammar changes efficiently, Adapser's LALR(1) parse table is generated in a lazy and incremental fashion. As Adapser parses its input, only those parts of its parse table that are necessary for parsing are generated according to its grammar. As a result, the overhead of computing those parts of the parse table that are never used while parsing a program is spared. It is important to note, however, that the efficiency of the parsing process itself remains unaffected, in the sense that once all needed parts of the parse-table have been generated, the parser will be as efficient as a conventionally generated parser.

In addition, Adapser handles grammar changes by incrementally correcting its parse table rather than by re-computing a new one from scratch. Those parts of the parse table that are unaffected by the modification in the grammar are reused, so that the effort put in generating them is not thrown away.

While adapting to grammar modifications, obsolete LR states may be removed and new states may be introduced. Consequently, the parse stack must be corrected to reflect the context of the current parse in terms of the new parse table. Adapser is capable of adapting to such grammar modifications without losing parse context. Therefore, it is capable of "reading" grammar modification directives directly from the parsed program and re-interpreting the current parse context according to the new grammar before the next token is read.

The idea of building a parser generator that accepts incremental/decremental changes to a grammar is not new. Most of the algorithms that are used in Adapser for lazy and incremental evaluation of the parse table are based on those described in IPG[2]. IPG (Incremental Parser Generator) is a full context-free grammar class parser that uses lazy and incremental evaluation of a LR(0) parse table to accept incremental and decremental changes to its grammar. Whenever a LR(0) conflict is detected, IPG creates separate tasks that explore each of the conflicting possibilities in parallel. Besides producing parsers for LALR(1) languages that are much less efficient than possible, IPG also fails to accept grammar modifications while parsing; that is, it is not an adaptive parser. It is important to point out that lazy and incremental generation of LALR(1) parse tables is much more difficult than that of LR(0) tables, as look-ahead sets have to be taken in account. As a consequence, LALR(1) table generation is slower than that of LR(0) tables.

Another implementation to which Adapser bears resemblance is ILALR[3]. ILALR is a fully interactive incremental generator of LALR(1) parsers, designed as a parser development and debug tool. Although ILALR is capable of accepting grammar modifications, its greedy approach for evaluating the LALR parse table forces (incremental) re-computation of the entire parse table after each grammar change. In addition, ILALR handles decremental grammar changes in a way that encourages redundant reduce actions that, in turn, cripple error recovery mechanisms. Also, as in the case of IPG, ILALR lacks the ability to accept grammar modifications while parsing.

In [4] an experimental implementation of a non-ambiguous context-free *dynamic parser* is described and applications of this parser for the resolution of some traditional semantic analysis problems, such as type checking, overloading resolution, derived types, and polymorphism, are described in detail. The above dynamic parser, although adaptive in the full sense of the word, is extremely complex and in the words of its author “should be improved in a much more descriptive manner”.

Finally, a recent commercial implementation of an adaptive parser called Meta-STM[5], which is part of a visual grammar development and debugging system, seems to offer all the capabilities provided by [4] and more. However, Meta-STM is a backtracking LL(k) parser, which implies the usual and well known disadvantages compared to LR parsers.

As a framework for constructing compilers, Adapser offers the following set of features:

- Translation scheme supporting embedded user-defined callback actions, inherited semantic attributes and operator precedence and associativity.
- Multiple parallel (thread safe) parsers.
- Recursive parser activation with alternative start-symbols.
- Error detection and recovery mechanism.
- Full LALR(1) conflicts detection with user-specified resolution.
- Trace mode parsing for parser debugging.
- Parser composition.
- Greedy evaluation mode for efficient parsing with a static grammar.

In order to demonstrate Adapser’s unique capabilities, a mini-interpreter was implemented using Adapser as a framework. The interpreted C-like language is strongly typed, supports user defined types, type inference, nested scopes and the usual control structures. However, all those features were implemented using only grammar modifications; that is, without using a symbol table or run-time stack.

It is the author’s intent to show that Adapser, although much simpler and more elegant than the dynamic parser described in [4], is capable of providing similar capabilities for practical applications.

References

- [1] J. R. Levine, T. Mason, D. Brown, Lex & YACC 2nd Edition. *O’reilly & Associates, Inc.* 1995.
- [2] J. Heering, P. Klint, J. Rekers, Incremental generation of parsers. *SIGPLAN Notices* 24/7, pp. 179-191, 1989.
- [3] R.N. Horspool, Incremental generation of LR parsers, Report DCS-79-IR, University of Victoria, Victoria, B.C., Canada, 1988.
- [4] Pierre Boullier, “Dynamic Grammars and Semantic Analysis”, INRIA Research report 2322, August 1994.
- [5] SandStone, “Meta-STM The Adaptive ParserTM”, <http://www.sandstone.com/Meta-S.html>