(Generalized) LR Parsing: From Knuth to Tomita

Anton Nijholt

Computer Science Department, University of Twente,
P.O. Box 217,7500 AE Enschede, The Netherlands
anijholt@cs.utwente.nl

ABSTRACT

This paper is a short introduction to the research in LR parsing and its applications. It is concerned with the history of LR grammars and languages, LR parsing and parser optimization, generalizations of the LR grammar definition and parsing method, automatic parser construction, error handling and LR parsing for natural language. Moreover, it introduces the other papers on (generalized) LR parsing in these proceedings.

1 INTRODUCTION

These notes provide a short introduction to the field of LR parsing. That is, we mention the research topics that emerged in this field and some activities that have been going on since the early sixties of this century. These proceedings are devoted to generalized LR parsing, an extension of the LR parsing method. For computer scientists LR parsing was introduced in a paper by Donald Knuth in 1965 [Knuth65]. After the introduction of the method it became silent for a few years, until F.L. DeRemer presented more practical, i.e., efficient, versions of the algorithm and corresponding subclasses of the class of LR grammars, viz. the SLR and LALR grammars [DeRemer69]. Nevertheless, practical compilers and compiler writing systems for programming languages were still based on precedence or recursive descent parsing techniques. W.R. LaLonde wrote an influential paper in which he reported the construction of an LALR parser generator [LaLonde72]. It was shown that many time and space optimizations could be done on the constructed LR tables and empirical comparisons were made with other parsing techniques. LR parsing compared favourably with other methods and, moreover, it was claimed that existing 'natural' grammars for programming languages required little or no change in order to make them LALR. Some years later S.C. Johnson chose the LALR parsing technique for a compiler writing system that was made part of the UNIX operating system: Yacc, Yet another compiler-compiler [Johnson75]. Since then (programming) language designers had a simple tool to their disposal for generating LALR parsers. Due to these developments in computing science LR parsing has become the most commonly used parsing technique. Indeed, as mentioned by LaLonde in 1977, LR(k) grammars " ... are popular because the grammars describe a large class of programming languages, the parser constructing techniques take a 'reasonable' amount of time, the parsing tables take a 'reasonable' amount of space, the parsers are fast, i.e., operate in linear time, and they have excellent error detection properties." A clear exposition of the different LR parsing techniques can be found in [Aho86].

In the first years after the introduction of the Chomsky hierarchy similar parsing methods were used by compiler builders and builders of natural language parsers. The results obtained by researchers in this area were shared. Results obtained in formal language theory could be used for programming language and natural language parsing. However, in computing science the main interest was in unambiguous grammars and there was a strong need for efficient, linear time, parsing methods in order to reduce compile time of programs. In natural language parsing the grammars had to capture the syntactic ambiguities of natural language. Therefore, efficient methods for the full class of context-free grammars had to be found. Since it turned out that parsing with respect to a context-free grammar for a (subset of a) natural language would lead to an explosion of the number of syntactic parses other researchers turned to non-context-free for-

malisms for natural language description. Sometimes these formalisms and parsing methods remained close to those for context-free grammars. Two aspects were important: 1) the embedding of syntax in formalisms and methods in which semantic information could be used to reduce the number of syntactic parses, and 2) notations that were more suitable for computational linguists to display and manipulate syntactic and semantic information. For some time augmented transition networks (ATNs) were used and some domain-dependent natural language processing systems were build based on ATNs [Woods70]. Other researchers put more emphasis on semantics and introduced more semantically oriented parsing methods, for example conceptual dependency parsing [Riesbeck75]. Results on the full class of context-free grammars were obtained by formal language theorists. Hardly any interest was displayed by computer scientists or computational linguists. Polynomial-time algorithms were obtained by Younger (the Cocke-Younger-Kasami algorithm [Younger67]) and Earley ([Earley68]). Earley had already published on LR parsers in 1967 and work done in 1964 had influenced Knuth's definition of LR grammars. The papers by Woods, Younger and Earley all appeared in computing science journals. Although it can be considered as a notational variant of the Earley and CYK method, (active) chart parsing, introduced by Kay (see [Kay80]), became more popular in the computational linguistics community than other context-free parsing methods. Its main advantage was a clear visualization of the parsing process.

In the 1980s there has been a revival of context-free grammars and parsing methods in computational linguistics. New grammar formalisms were introduced (e.g., generalized phrase structure grammars) with explicit context-free components and discussions were started whether or not natural languages are context-free (see e.g. [Savitch87] for a collection of papers). Moreover, there was a decrease in a reluctance to study and use formal methods from formal language theory in computational linguistics. To mention another example, see a collection of papers on computational complexity and natural language [Barton87]. Illustrative is also a remark in [Pullum84]. Pullum first notes that there has been a strong resurgence of the idea that context-free structure grammars could be used for the description of natural languages and next observes: "Techniques straight out of programming language and compiler design may ... be of considerable interest in the context of natural language processing applications." In 1985 the book *Efficient Parsing for Natural Language* by M. Tomita appeared [Tomita85]. In this book a generalized LR parsing method was introduced for the (almost full) class of context-free grammars. It was argued that the algorithm performed more efficiently than any other existing parsing algorithms *in practice*. Since then the method has been used in natural language interfaces, machine translation systems and speech-to-speech translation systems. Hence, not only the computer science community, but also those researchers working on natural language processing have become familiar with LR parsing. Many papers on (generalized) LR parsing appeared in the proceedings of the first International Workshop on Parsing Technologies [Tomita89]. Presently the generalized LR parser developed by Tomita at Carnegie Mellon University has been made part of a software package for practical natural language processing projects. The algorithm is augmented with unification modules. The package is publicly available [Tomita90].

2 Properties of LR grammars and languages

LR(k) stands for 'parsing from **L**eft to right using **R**ightmost reductions and **k** symbols of lookahead'. LR(k) grammars are usually defined by introducing conditions on the rightmost derivations of context-free grammars. LR(k) grammars allow deterministic bottom-up parsing. The productions that have been used in a rightmost derivation of a sentence are recognized in their reversed order. This is done by reducing right-hand sides of productions that appear in the sentence and intermediate right sentential forms to their left-hand sides until the start symbol has been reached. The conditions in the LR(k) definition are such that each time the position of the right-hand side and the production to be used in the reduction are unique.

Many of the formal properties of LR(k) grammars have already been discussed in Knuth's original paper. Later papers have been concerned with practical approaches to the parsing problem and more formal treatments of the properties of LR grammars and parsing. Theoretical problems are e.g. the relationships with other classes of

grammars, decidability and equivalence problems and relationships with (deterministic) pushdown automata. There exist direct, but complicated, language preserving transformations from LR(k) to LR(1) grammars. The class of LR(1) languages coincides with the class of deterministic languages, i.e. the class of languages that can be accepted by deterministic pushdown automata. LR(k) grammars are unambiguous. Many subclasses of LR(k) grammars have been introduced. In many cases this was done to obtain even more efficient parsing methods.

3 LR PARSING AND PARSER CONSTRUCTION

Automatic LR parser construction is almost always based on the following idea of DeRemer. LR(0) grammars are not always sufficient to describe programming language constructs. However, if we give an LR(0) parser the possibility of a lookahead of one symbol then most constructs can be captured. Therefore automatic construction consists of first computing the LR(0) parser and then a method is used to incorporate a lookahead of one symbol in this parser. This lookahead computation has lead to the definitions of SLR(1) and LALR(1) grammars. The construction of an LR parser is in fact the construction of a finite automaton which is used as finite control of a deterministic pushdown automaton. The pushdown automaton acts as a shift-reduce parser. The finite automaton, i.e., its states and its transition function, is constructed from the production rules of the grammar. The automaton is mostly presented as a (parse) table. In the table it can be read which actions (shift a symbol from the input on the stack or pop symbols from the stack in order to effect a reduce action) and which state transitions have to be done. The actions are determined by the current state and the lookahead.

Methods have been introduced for increasing the parser efficiency. That is, increasing the speed of the parser and reducing its size. Especially for k¿1 an LR parser can require an unpractical amount of space. In fact, the number of states of an LR(k) parser can be exponential in the size of the grammar. Even for k=1 the construction of an LR(1) parser can result in disastrous space requirements. SLR(1) and LALR(1) parsers are less general than LR(1) parsers but they can be constructed from an LR(0) parser and they require much less space. Several techniques have been introduced to improve LR parser efficiency. Early research concentrated on techniques which were applied to the constructed parse table. Later research concentrated on methods which could be included in the parser construction algorithm. One method to reduce parsing time is to eliminate semantically irrelevant chain reductions from an LR parser. It is possible to do this in such a way that states are merged. Hence, also the parser size is reduced. Reduction of states in an LR parser can also be obtained by eliminating states where the only action is a reduction. The generalization of this idea has lead to the introduction of 'default reductions', reductions which are performed without checking the lookahead. There exist many techniques for reducing sparse matrices. These compression methods can be used for LR parse tables. An important question is always whether the optimization techniques leave the error detection capabilities undisturbed. In general, optimization techniques for LR parsers can lead to enormous size reductions and speed increase. Therefore they should always be applied or be included in the construction process of a parser. It is not unusual that an optimized parser is twice as fast and requires only fifty percent of the space of the original parser. Recent books in which many results on LR parsing, parser construction and optimization can be found are [Sippu88] and [Sippu90].

Although optimization and error detection and recovery issues have drawn most of the interest, there are other issues which have received attention. There are results on parallel parsing, parsing ambiguous grammars (with the help of disambiguating rules), on-line parsing, unparsing and parsing of incomplete sentences. Moreover, incremental LR parsing (e.g., [Agrawal83] and [Gafter87]) and incremental and lazy parser construction (e.g., [Fischer80] and [Horspool90]) have been studied. An incremental parser makes it possible to reshape a parse tree when its corresponding string is modified without reparsing the complete input string. Incremental parser construction can be useful if a grammar is often changed, for example during the design phase of a language. After each change a new parser must be generated. In incremental parser construction the parts of the parse table that are not affected by the grammar change are re-used. In lazy parser generation the parser is generated by need while parsing the input string. Only those parts of the parser are generated that are needed by the par-

ticular input string.

4 LR GENERALIZATIONS

Many generalizations of LR(k) grammars have been explored. Knuth mentioned the LR(k,t) grammars for which the condition is relaxed that exactly the production that has been used in a rightmost derivation should be reduced in the process of converting intermediate sentential forms until the start symbol has been reached. Others have allowed unbounded lookahead. In this case the set of all strings over the alphabet is partitioned into regular sets and by determining to which set a certain lookahead belongs parsing decisions are made. Although parsing time remains linear it becomes possible to parse nondeterministic languages. LR type grammars and parsing methods have also been introduced for non-context-free classes of grammars. Methods have been presented for subsets of the indexed grammars [Sebesta78], context-sensitive grammars [Walters70], type-0 grammars [Turnbull79], [Harris85] and two-level grammars (Van Wijngaarden grammars, W-grammars) [Wegner80]. Moreover, it has been investigated for attribute and affix grammars whether attribute evaluation can be done during LR parsing and whether attribute values can be used in guiding the parsing process [opdenAkker88].

Various authors have explored the possibility to extend the LR parsing method such that a larger subset of the context-free grammars can be parsed while retaining the simplicity and the efficiency of the LR parsing method and the parser construction. The general idea is as follows. If a grammar fails to be LR the constructed LR parse table contains multiple entries. There are shift/reduce or reduce/reduce conflicts. Nevertheless the table can be used as a nondeterministic finite control of the pushdown automaton that acts as a shift-reduce parser. This is called nondeterministic LR parsing. However, if we want to parse a general context-free grammar, rather than making a nondeterministic choice during parsing, each possible choice should be explored. Some paths in the computation will fail, other paths (more than one when the grammar is ambiguous) will be successful. The possible choices that can be made during parsing of a sentence can be displayed in a computation tree. In order to walk the paths of the computation tree different techniques can be used. In a computational environment with more than one processor different paths can be followed in parallel. For example, whenever a nondeterministic choice has to be made we can introduce new processors and each processor continues with the current contents of the parse stack, the parse table and the remaining input. Observations like these have been given in the parsing literature. They have hardly been explored. In a more general setting this nondeterministic table driven parsing has been mentioned in [Lang74]. Computer scientists were not interested in general context-free grammars and especially not in such a method since it was much less efficient than existing methods. In order to obtain an efficient version of nondeterministic LR parsing some authors introduced a bound on the nondeterminism. For example, in [Frank79] bounded nondeterministic LR parsing is studied in order to allow efficient parallel parsing. If, for a given grammar, each computation tree for any input has at most m leaves, where m is a fixed number dependent on the grammar only, then m processors can be used to parse the grammar's sentences in parallel. It can be shown that there exists an infinite hierarchy of languages, starting with the deterministic context-free languages, based on the number m. Hence, by increasing the number of processors more context-free languages can be parsed with this model. Independently from Frank nondeterministic and bounded nondeterministic LR parsing has been studied in [Schäfer80] and [Schäfer82]. Theoretical observations on classes of bounded nondeterministic LR languages have been given in [Kintala78].

In [Tomita85] multiple entries in the parse table are handled with the help of a graph-structured stack, where the graph is directed and acyclic. As we mentioned above, for each path in the computation tree it is possible to introduce separate parsers, each performing actions on its own stack. Hence, in a breadth-first search of the computation tree a list of stacks has to be maintained. Tomita's contribution to nondeterministic LR parsing is to make it deterministic by the introduction of stack combination techniques. They are based on the following observations. From a certain point, due to the situation on top of the stacks, different stacks may start to behave exactly the same. If this happens, the tops of the stacks can be unified and these stacks can be combined into a tree. From its root only one continuation needs to be represented (if the stack grows). Similarly, when a multiple entry is encountered there is no need to make copies of the

stacks. We can as well keep the bottom portion of the stack and split it according to the multiple entries. The result of the combination of these two ideas is Tomita's graph-structured stack. In order to improve the space efficiency of the algorithm it is also necessary to have an efficient representation of all the parse trees that are constructed. This is done by introducing techniques for the sharing of subtrees and the merging of top nodes of certain subtrees. The algorithm can be based on any type of LR parse table (LR(0), SLR(1), LALR(1), LR(1), etc.). The main properties of the algorithm thus obtained are [Tomita85]:

- It is more efficient if the grammar is 'close' to LR.
- It is less efficient if the grammar is 'densely' ambiguous.
- It is not able to handle cyclic grammars.
- Empirical results show that in practical applications the algorithm is significantly more efficient than Earley's algorithm.

The similarities between Earley's algorithm and the generalized LR algorithm have been made explicit in [Sikkel90]. Tomita's ideas about the graph-structured stack were independently developed by Van der Steen [vanderSteen87]. Van der Steen extended the technique to type-1 (an extension of Walters' parsing algorithm) and type-0 grammars (an extension of Turnbull's algorithm). An application of Tomita's algorithm in computer science is reported in [Rekers92]. The application concerns the derivation of programming environments from formal language definitions. Generalized LR parsing was selected as the basis for the syntactic tools. The system allows the interactive development of syntax definitions. Grammar writers are given maximal freedom. They are not forced to ensure beforehand that the grammar parts with which they are experimenting do not lead to ambiguity. However, if a grammar is LR the constructed parser should be comparable in speed with LR parsers. The generalized LR parser which is described is based on an LR(0) parse table. Due to the application of the algorithm it became necessary to develop techniques for, among others, lazy and incremental generalized LR parser construction. Tomita's method is slightly adapted so that it can handle cyclic grammars.

5 About the papers

In these proceedings eight invited papers presented at the first Twente Workshop on Language Technology (march 1991) can be found. The first paper, *Recursive Ascent Parsing* by **René Leermakers**, throws a completely new light on deterministic and nondeterministic LR parsing by giving a functional description of LR parsers. This formulation allows relatively simple correctness proofs for LR parsers and by using a technique from functional programming known as memo-functions this functional LR parser can parse non-LR grammars with a cubic time complexity. Due to its formulation there is no reference to a push-down or a graph-structured stack. However, they appear implicitly as a procedure stack induced by functional programming language concepts. The results on LR parsing in this paper are extended to grammars with productions with regular expressions in the right-hand sides (extended context-free grammars) and to a functional definition of the Marcus lookahead parsers. The author claims that for non-LR grammars there is no reason to use Tomita's algorithm since his algorithm has the possible advantages of Tomita's parser for natural language grammars, while being polynomial.

In the last chapter of Tomita's book some suggestions for future work can be found. One of the suggestions concerns the extension of the generalized LR algorithm to context-sensitive grammars. Reference is made to Walters' deterministic shift-reduce parse algorithm for a subset of the context-sensitive grammars, utilizing a parse table similar to an LR parse table. Extension of the algorithm should make it possible to handle multiple entries in this parse table. In the paper *A Parsing Algorithm for Nondeterministic Context-Sensitive Languages* by **Henk Harkema** and **Masaru Tomita** the extension to context-sensitive grammars is explored. Walters' algorithm uses two stacks: a parse stack to keep track of the parsing process and an input buffer on which, among others, results of reduce actions are pushed. In the extended version both stacks are replaced by graph-structured stacks. The authors give a clear exposition of the method and discuss improvements of its efficiency.

In the paper *Unrestricted On-line Parsing and Transduction with Graph-Structured Stacks* by **Gert van der Steen** a comprehensive program-generator is discussed for parsing and transduc-

tion based on the generalized LR technique and suitable for grammars in the full Chomsky hierarchy and for grammar formalisms like, among others, augmented transition networks, transformational grammars and augmented context-free grammars (attribute grammars). It is expected that the linguist writes a grammar in a so-called unifying formalism, a formalism in which we recognize type-0 grammar rules and the possibility to assign variables (attributes) to nonterminal symbols. The program-generator converts the grammar into a program written in a code for an abstract machine, the Parallel Transduction Automaton (PTA), designed by the author. It can be considered as an extension of a two-stack machine, where the stacks are directed acyclic graphs in order to allow generalized LR parsing for context-free, context-sensitive and type-0 grammars. The system and its underlying LR theory has been developed in the early eighties. Recently it has been extended with an error-repair facility. Among the applications are the parsing of several natural languages, the recognition of compound words and the transformation of graphemes into phonemes.

In the next paper, *Substring Parsing for Arbitrary Context-Free Grammars* by **Jan Rekers** and **Wilco Koorn**, we return to context-free language parsing. The task of a substring recognizer is to determine whether a string is a substring of a sentence. A substring parser generates trees for the possible completions of the substring. In this paper a substring recognizer and parser are introduced based on the generalized LR algorithm. The important difference with a few other algorithms for substring parsing is that it works for the full class of context-free grammars and that there is no need for specially generated parse tables. The same tables as used in the original parser are used for the substring parser. Applications of substring parsing include syntax error recovery, syntax-directed editing and incremental parsing. It is not clear whether the present algorithm is sufficiently efficient to be used for practical applications.

The next two papers deal with (generalized) LR parsing algorithms in practical natural language applications. In *Detection and Correction of Morpho-Syntactic Errors in Shift-Reduce Parsing* **Theo Vosse** describes grammar-independent error detection in a shift-reduce parser for augmented context-free grammars. The errors that are considered are morpho-syntactic errors, errors that are related to the derivation and inflection of words (agreement violations, spelling and typing errors, punctuation errors, etc.). In general these errors can not be detected by spelling checking mechanisms. What is needed is a parser that handles ungrammatical input. In the paper a standard shift-reduce parser based on an LR parse table is extended so that it can handle augmented context-free grammars and it is shown how the parser helps in detecting and sometimes correcting the different types of errors. It is also discussed which adaptations have to be made to the generalized LR algorithm in order to parse augmented context-free grammars and to obtain similar error detection capabilities. The parser has been implemented in a Dutch grammar checker.

In *Tomita's Algorithm in Practical Applications* **Rob Heemels** discusses the advantages of the generalized LR algorithm for natural language applications. The paper is based on his experiences with the algorithm in a natural language parser developed by the language and speech technology group of Océ. One important requirement of a natural language processing system is the possibility to handle ungrammatical input. In the paper a classification of errors in ungrammatical sentences is given. Most of the errors caused by the user are grammatical ones, followed by style, typing and spelling mistakes. The author presents some requirements of a robust parsing algorithm, discusses properties of Tomita's generalized LR algorithm and concludes that it is not only one of the fastest and most efficient algorithms, but it has also the potential to meet most of the requirements for a robust parser.

In An *Empirical Comparison of Generalized LR Tables* **Marc Lankhorst** compares the efficiency of generalized LR parsing for LR(0), SLR(1), LALR(1) and LR(1) parse tables, respectively. The comparison is made on the basis of empirical data. Use is made of the grammars and the sentence sets of [Tomita85] which were used by Tomita to compare Earley and generalized LR parsing. From the grammars the different parse tables were constructed and their sizes were compared. With the sentence sets parsing efficiency of the different parsers was compared. It turned out that for one of the larger grammars, due to a lack of memory, the LR(1) parse table could not even be constructed. From the comparisons with respect to space the conclusions are that LR(1) tables are unsuitable, LALR(1) tables are the best choice, but if space is a constraint or

ease of construction is important, then LR(0) tables are advised. As mentioned earlier in this introduction, in [Rekers92] generalized LR parsing is based on LR(0) parse tables exactly for the reason of ease of construction. Surprisingly, perhaps, using LR(1) tables turns out to be *slower* than LALR(1), SLR(1) and even LR(0). This confirms a remark of Billot and Lang [Billot89], who found (in a somewhat different context) that adding sophistication to chart parsing schemata may harm, rather than improve the efficiency. Lankhorst clearly illustrates and explains this phenomenon in the context of a Tomita parser.

The final paper, *Bottom-Up Parallelization of Tomita's Algorithm*, is by **Klaas Sikkel**. A parallel version of the generalized LR algorithm is presented. The algorithm uses a pipeline of processors, one for each word in the sentence. Each processor (or process) runs an adapted version of the Tomita parser. Symbols are passed along the pipeline from right to left. The leftmost processor will deliver a parse of the sentence. In the paper ordering requirements for the passing of symbols are discussed. In order to prevent communication bottle-necks communication saving rules are introduced. These rules can be used by the parser to detect whether a symbol will be used by a processor further down the pipeline. If not, there is no need to pass it from one processor to another. In the paper it is announced that the algorithm will be tested against some natural language grammars.

REFERENCES

[Agrawal83] R. Agrawal and K.D. Detro. An efficient incremental LR parser for grammars with epsilon productions. *Acta Informatica* **19** (1983), 369-376.

[Aho86] A.V. Aho, R. Sethi and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.

[opdenAkker88] H.J.A. op den Akker. *Parsing Attribute Grammars*. Ph.D. Thesis, University of Twente, The Netherlands, 1988.

[Barton87] G.E. Barton, R.C. Berwick and E.S. Ristad. *Computational Complexity and Natural Language*. The MIT Press, Cambridge, Mass., 1987.

[Billot89] S. Billot and B. Lang. The Structure of Shared Forests in Ambiguous Parsing. *27th Ann. Meeting of the ACL*, Vancouver, 143-151, 1989.

[DeRemer69] F.L. DeRemer. *Practical translators for LR(k) languages*. Ph.D. Thesis, Project MAC TR-65, MIT, 1969. See also: F.L. DeRemer. Simple LR(k) grammars. Comm. ACM 14 (1971), 453-460.

[Earley68] J. Earley. An efficient context-free parsing algorithm. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, 1968. See also: *Comm. ACM* **13** (1970), 94-102.

[Fischer80] G. Fischer. *Incremental LR(1) Parser Construction as an Aid to Syntactical Extensibility*. Ph.D. Thesis, Universität Dortmund, Abteilung Informatik, 1980.

[Frank79] P.D. Frank. *Bounded nondeterminism and the parallel parsing of context-free languages*. Ph.D. Thesis, University of Washington, Seattle, WA, 1979.

[Gafter87] N.M. Gafter. Algorithms and data structures for parallel incremental parsing. *Proc. Int. Conf. on Parallel Processing*, 577-584, 1987.

[Harris85] L.A. Harris. *SLR(1) and LALR(1) parsing for unrestricted grammars*. Mathematics Department, University of Kentucky, Lexington, Kentucky 40506, 1985.

[Horspool90] R.N. Horspool. Incremental generation of LR parsers. *Computer Languages* **15** (1990), 205-233.

[Johnson75] S.C. Johnson. *Yacc — Yet another compiler-compiler*. CSTR 32, Bell Laboratories, Murray Hill, N.J.,1975.

[Kay80] M. Kay. *Algorithm Schemata and Data Structures in Syntactic Processing*. Report CSL-80-12, Xerox PARC, Palo Alto, Ca., 1980.

[Kintala78] C.M.R. Kintala. Refining nondeterminism in context-free languages. *Mathematical Systems Theory* **12** (1978), 1-8.

[Knuth65] D.E. Knuth. On the translation of languages from left to right. *Information and Control* **8** (6): 607-639, 1965.

[LaLonde72] W.R. LaLonde, E.S. Lee and J.J. Horning. An LALR(k) parser generator. *Proc. IFIP Congress '71*, North-Holland Publ. Co., Amsterdam, 1971, 153-157.

[Lang74] B. Lang. Deterministic techniques for efficient nondeterministic parsers. Proc. Second Colloquium on Automata, Languages and Programming. J. Loeckx (ed.), *Lecture Notes in Computer Science* **14**, Springer-Verlag, 1974, 255-269.

[Pullum84] G.K. Pullum. Syntactic and semantic parsability. *Proc. COLING'84*, Stanford University, 1984, 112-122.

[Rekers92] J. Rekers. *Parser Generation for Interactive Environments*. Ph.D. Thesis, University of Amsterdam, 1992, to appear.

[Riesbeck75] Conceptual analysis. In: *Conceptual Information Processing*. R.C. Schank (ed.), North-Holland Publ. Co., Amsterdam, 1975, 83-156.

[Savitch87] W.J. Savitch et al (ed.). *The Formal Complexity of Natural Language*. D. Reidel Publishing Company, Dordrecht, 1987.

[Schäfer80] W. Schäfer. *Kontextfreie LR(k) Analyse unter Verwendung mehrdeutiger Grammatiken*. Diplomarbeit, Universität Dortmund, Juni 1980.

[Schäfer82] W. Schäfer. *Nondeterministic LR(k) parsing*. Osnabrücker Schriften zur Mathematik I-7, Fachbereich Mathematik, Universität Osnabrück, 1982.

[Sebesta78] R.W. Sebesta and N.D. Jones. Parsers for indexed grammars. *Int. J. Comput. Inform. Sci.* **7** (1978), 345-359.

[Sikkel90] K. Sikkel. *Cross-fertilization of Earley and Tomita*. Memoranda Informatica 90-69, University of Twente, The Netherlands, 1990.

[Sippu88] S. Sippu and E. Soisalon-Soikinen. Parsing Theory, Vol. 1: Language and Parsing. *EATCS Monographs on Theoretical Computer Science* **15**, Springer-Verlag, Berlin, 1988.

[Sippu90] S. Sippu and E. Soisalon-Soikinen. Parsing Theory, Vol. 2: LR(k) and LL(k) parsing. *EATCS Monographs on Theoretical Computer Science* **20**, Springer-Verlag, Berlin, 1990.

[vanderSteen87] G.J. van der Steen. *A Program Generator for Recognition, Parsing and Transduction with Syntactic Patterns*. Ph.D. Thesis, University of Utrecht, 1987. Also: CWI Tract 55, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[Tomita84] M. Tomita. LR parsers for natural languages. *Proc. COLING'84*, Association for Computational Linguistics, Stanford University, 1984, 354-357.

[Tomita85] M. Tomita. *Efficient Parsing for Natural Language*. A fast algorithm for practical systems. Kluwer Academic Publishers, Boston, 1985.

[Tomita89] M. Tomita (ed.). *Proc. Int. Workshop on Parsing Technologies* (IWPT-89). Carnegie Mellon University, Pittsburgh, 1989.

[Tomita90] M. Tomita. The generalized LR parser/compiler V8-4: A software package for practical NL projects. *Proc. COLING'90*, Association for Computational Linguistics, Helsinki, 1990, 50-63.

[Turnbull79] C.J.M. Turnbull and E.S. Lee. Generalized deterministic left to right parsing. *Acta Informatica* **12** (1979), 187-207.

[Walters70] D.A. Walters. Deterministic context-sensitive languages. *Information and Control* **17** (1970), 14-61.

[Wegner80] L.M. Wegner. On parsing two-level grammars. *Acta Informatica* **14** (1980), 175-193.

[Woods70] W.A. Woods. Transition networks for natural language analysis. *Comm. ACM* **13** (1970), 591-606.

[Younger67] D.H. Younger. Recognition and Parsing of Context-Free Languages in Time n^3, *Information and Control* **10**, 2 (1967) 189-208.