CHART PARSING AND THE EARLEY ALGORITHM

James Kilbury

Chart parsing and the algorithm of Earley are topics of considerable current interest, but they seldom are discussed explicitly in relation to each other. The aim of this paper is to present the Earley algorithm as a particular kind of chart parser and then to compare it with a modified algorithm that is closely related but apparently has not been presented elsewhere in the literature on parsing. In the course of this discussion special attention will be directed at a series of fundamental distinctions that constitute the basis for a new parser conception.

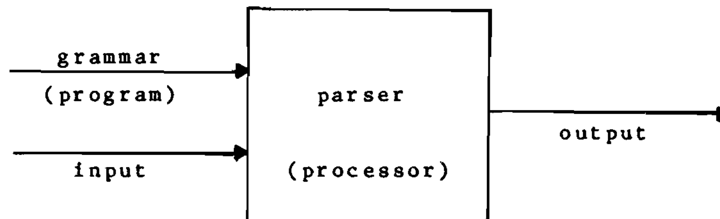
For workers in the area of computational linguistics the distinction between a grammar and a parser may well seem obvious, but we nevertheless can observe widespread confusion regarding these notions. The growing importance of natural language computer systems has brought people into contact with parsing who have little understanding of computational linguistics. Typical of the misunderstandings that arise is the question of the representative of a major computer company who wanted a new parser and wondered whether Generalized Phrase Structure Grammar or Lexical Functional Grammar would be faster. Such confusion is not surprising in the light of earlier ad hoc systems that indiscriminately mix linguistic data with algorithms.

Other misunderstandings are more subtle. One often hears people speak of "writing a Definite Clause Grammar parser," but aside from trivial syntactic conveniences, the PROLOG system itself is the parser, and what has been written is a grammar in a special formalism for PROLOG. In ATN parsers and the WASP parser of Marcus the distinction of grammar and parser is maintained but blurred in a curious way: categories of the parser itself are carried over into the grammar, so that the grammatical formalism resembles an assembler language in its orientation to the processor.

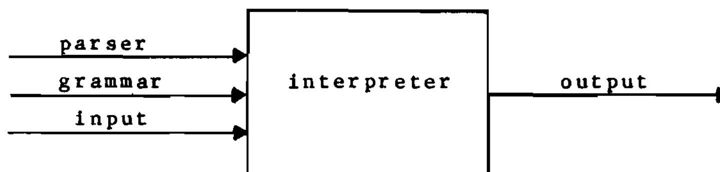
A clear conceptual distinction between grammar and parser can be achieved if the parser is "driven" by the grammar as its knowledge base. Obviously, this permits different grammars to be used with the same parser. Linguistic descriptions can be utilized better in developing a natural language system, and in particular, a practical division of labor is made possible, since linguists can write the grammar with no knowledge of programming and, in principle, little or no knowledge of the parser that is to use the grammar.

Such a conceptual modularization has not only the practical advantages mentioned above but also theoretical advantages. The grammar is independent of the parser in the sense that it can fully utilize categories of theoretical linguistics rather than categories dictated by the processor. On the other hand, if the mathematical properties of the grammatical formalism are sufficiently clear, then it is possible to see the interdependence of the grammar and parser in terms of the Chomsky hierarchy according to which each type of Chomsky grammar has a corresponding automaton which is just powerful enough to recognize or parse the corresponding language.

The viewpoint adopted here regards the parser as a processor which interprets the grammar as its program:



In practice, of course, we implement a virtual processor rather than constructing a specialized hardware parser:



The grammar and parser remain clearly distinct, however.

Given this viewpoint, the grammatical formalism is of central importance. Its role is described in an unpublished work by Gazdar et al.:

> A grammatical framework can and should be construed as a formal language for specifying grammars of a particular kind. The syntax and, more importantly, the semantics of that formal language constitute the substance of the theory or theories embodied in the framework.

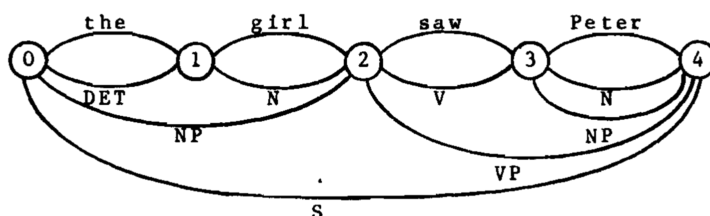
Stated in computational terms, a grammatical formalism amounts to a (higher) programming language in which programs (i.e. grammars) are written. It may show an arbitrary degree of abstraction from the processor and may be either interpreted or - with the expected gains in run-time efficiency - compiled.

To serve as the definition of a programming language the syntax and semantics of the grammatical formalism must be entirely explicit, as is proposed by Gazdar et al. This is an obvious prerequisite for implementation but also brings additional advantages: ad hoc interpretations of the theory can be avoided in the implementation, and - above all - the relation of the parser to a particular linguistic theory is made explicit.

The Earley algorithm is of interest here because it exemplifies a strict conceptual distinction between grammar and parser. Before we turn to the algorithm itself we should briefly consider what a parser does (cf VARILE 1983). A parser performs an incremental analysis of a sentence or text and produces a syntactic analysis as its output. At many points in the analysis the parser must deal with alternatives - either in the parts already analyzed or in those to follow, and a general mechanism to deal with the alternatives is required. One possibility is backtracking, in which the individual alternatives are explored sequentially. The main disadvantage of backtracking is that earlier results may be lost, so that a considerable amount of work may be repeated in exploring the successive alternatives. Another possibility is to keep a record of all alternatives as they appear in order to avoid repeating previous work. To keep such a record during the computational process a suitable data structure is required.

Essentially the same data structure was proposed for this purpose by different researchers around 1970. Within computational linguistics, Kay and Kaplan introduced the notion of a <u>chart</u> (cf WINOGRAD 1983, VARILE 1983, and THOMPSON 1981 for literature and an explanation of charts), while Earley (cf EARLEY 1970 and AHO/ULLMAN 1972) developed ideas current in the syntactic analysis of formal languages to formulate his notion of an <u>item</u>.

The chart (or <u>well-formed substring table</u>) is designed precisely to store intermediate results of the parsing process. It is a graph consisting of nodes (or <u>vertices</u>) connected by a set of arcs (or <u>edges</u>). The vertices are numbered and correspond to the positions before or after words of the input string; as ends of an edge they indicate the extent of a grammatical constituent. Edges are labelled and bear information about the constituent. The following figure may serve as a simple example:



In the approach of Earley an edge corresponds to an item and a chart to a set of items. Unfortunately, the notations used by EARLEY (1970) and AHO/ULLMAN (1972: 320ff) somewhat obscure the relation of edges to items. The notation of SHIEBER (1984) is clearer but also does not indicate the extent of an edge in the notation for the corresponding item. In the present paper an item is defined as a pentuple $[i, j, A, \alpha, \beta]$ where

- i and j are the numbers of the vertices that begin and end the edge, respectively,
- A is the LHS of a production with $\alpha\beta$ as corresponding RHS,
- α is the string of daughters of A that have been recognized, and
- β is the string of daughters of A that remain to be recognized.

The same item is represented as $[A \dashrightarrow \alpha.\beta, i] \in I_j$ by Aho/Ullman and as $[A, \alpha, \beta, i] \in I_j$ by Shieber.

In the general case the item $[i, j, A, \alpha, \beta]$ corresponds to an active edge. When $i = j$ and $\alpha = \varepsilon$ (the empty string) no daughter of A has as yet been recognized. When $\beta = \varepsilon$ all the daughters of A have been recognized, and the item corresponds to an inactive edge.

Strictly speaking, the items described here are for recognition rather than parsing. For the latter the items can be augmented with syntax trees or derivational information, semantic representations, etc.

Although the item or chart as data structure shapes the parser in a fundamental way, the data structure must be clearly distinguished from the parsing algorithm just as the parser is distinguished from the grammar. Some confusion may arise here because of the customary terminology, which allows parsers to be designated in various ways: the expressions "top-down parser" and "left-corner parser" refer to algorithmic strategies, while "chart parser" refers to the data structure. This must be stressed because distinct or even radically different algorithms can use the same chart or item as data structure, as the remainder of this paper will illustrate. Two algorithms will be presented that operate by constructing sets of items, and the first of these is the Earley algorithm itself.

The algorithm of Earley embodies a combination of top-down and bottom-up strategies with the chart-type data structure in an entirely novel way. It is considered to be the most efficient practical algorithm known for parsing context-free languages, although the efficiency calculation largely rests on the worst-case analysis that has been criticized by SLOCUM (1981) as misleading for natural-language parsing. The comparison of parsing algorithms with respect to their efficiency remains a largely open question, as we shall see below.

Earley's algorithm has recently received renewed attention in computational linguistics because of the present interest in restricted syntax formalisms with more or less context-free power (cf PULMAN 1983 and SAMPSON 1983). SHIEBER (1984) has adapted the Earley algorithm to the immediate dominance/linear precedence formalism of Generalized Phrase Structure Grammar.

The algorithm is driven by a context-free grammar G = <N, T, P, S>, where N, T, and P are finite, nonempty sets of nonterminal symbols, terminal symbols, and productions, respectively, and S is a start symbol in N. A production is a pair <A, α> (normally written A --> α) where A is a nonterminal symbol and α is a string of terminals and nonterminals.

The algorithm may be sketched informally. The item set is initialized with an item indicating that the start symbol is sought but that nothing has yet been identified; items for all the possible constituents that can immediately follow (i.e. all the "left corners" of S) are generated by the <u>predictor</u> using information from the grammar. The input string is then processed sequentially from left to right. In each step the next terminal symbol is read. Whenever a constituent - be it a terminal or nonterminal symbol - is identified the <u>completer</u> extends active edges (i.e. items) requiring the identified constituent, while the <u>predictor</u> creates new active edges for all the possible constituents that can immediately follow. An input of length n is recognized (or with a slight extension of the algorithm, parsed) if there is at least one inactive S-edge extending from 0 to n, i.e. if the item set contains an item of the form $[0, n, S, \alpha, \varepsilon]$ after the input has been processed.

Various formal statements of the algorithm exist. The version of EARLEY (1970: 97) is not generally quoted, while the widespread formulation of AHO/ULLMAN (1972: 321) is remarkably inelegant because of its lack of generality. The following formulation captures a generalization of Earley's <u>scanner</u> (completion with terminal symbols) and his <u>completer</u> (completion with nonterminal symbols). Note that ε is the empty string and that $A\alpha$ and αA are catenations of the string α with the terminal or nonterminal symbol A.

```
program PARSE a_1...a_n:
begin
CLOSURE for [0, 0, S, ε, S];
for i = 1 to n do CLOSURE for [(i-1), i, a_i, a_i, ε]
end;
```

```
procedure CLOSURE for [i, j, B, γ, δ]:
begin
if [i, j, B, γ, δ] ∉ I then
   begin
   add [i, j, B, γ, δ] to I;
   if δ = ε then COMPLETE with [i, j, B, γ, ε]
            else PREDICT  with [i, j, B, γ, δ]
   end
end;

procedure COMPLETE with [j, k, B, γ, ε]:
begin
for each [i, j, A, α, Bβ] ∈ I do CLOSURE for [i, k, A, αB, β]
end;

procedure PREDICT with [i, j, A, α, Bβ]:
begin
for each <B, γ> ∈ P do CLOSURE for [j, j, B, ε, γ]
end.
```

The operation of the algorithm can be illustrated with the grammar G = <N, T, P, S> with N = {S, NP, VP}, T = {det, n, adv, v}, and the following productions in P:

```
S  --> NP VP
NP --> det n
NP --> n
VP --> VP adv
VP --> v NP
VP --> v
```

For the input string w = det^n^v^adv the algorithm generates the following items. Numbers following some items indicate the items with which they arise through completion.

```
1  [0, 0, S, ε, S]
2  [0, 0, S, ε, NP^VP]
3  [0, 0, NP, ε, det^n]
4  [0, 0, NP, ε, n]
5  [0, 1, det, det, ε]
6  [0, 1, NP, det, n]       3 & 5
7  [1, 2, n, n, ε]
```

8 [0, 2, NP, det⌢n, ε] 6 & 7
9 [0, 2, S, NP, VP] 2 & 8
10 [2, 2, VP, ε, VP⌢adv]
11 [2, 2, VP, ε, v⌢NP]
12 [2, 2, VP, ε, v]
13 [2, 3, v, v, ε]
14 [2, 3, VP, v, NP] 11 & 13
15 [3, 3, NP, ε, det⌢n]
16 [3, 3, NP, ε, n]
17 [2, 3, VP, v, ε] 12 & 13
18 [0, 3, S, NP⌢VP, ε] 9 & 17
19 [0, 3, S, S, ε] 1 & 18
20 [2, 3, VP, VP, adv] 10 & 17
21 [3, 4, adv, adv, ε]
22 [2, 4, VP, VP⌢adv, ε] 20 & 21
23 [0, 4, S, NP⌢VP, ε] 9 & 22
24 [0, 4, S, S, ε] 1 & 23

Items of the form [i, j, S, S, ε] result from the initialization with [0, 0, S, ε, S]. Items 18 and 19 show that an S is recognized before the entire input has been processed. Note that the left recursion of the fourth production, which would be a problem for some algorithms, is neatly dealt with.

The test run also reveals difficulties. With items 4, 11, 14, 15, and 16 the predictor introduces active edges which are not extended in the rest of the analysis. For a natural language such as English or German this effect would lead to unacceptably large item sets. Before a single input word had been processed the predictor would have to introduce items for all possible constituents that could begin a sentence, and excessive numbers of predictions would also be made throughout the analysis. For practical chart parsing with grammars of natural languages an algorithm with weaker predictive power is necessary.

The next algorithm introduced here is intended to overcome precisely this defect. It employs a more strongly bottom-up strategy and essentially constitutes a left-corner chart parser (cf ROSS 1982 for left-corner parsing). Detailed discussions of the algorithm and its adaptation to the ID/LP formalism of GPSG are given in KILBURY (1984) and KILBURY (forthcoming).

This algorithm differs from Earley's primarily in the operation of its predictor. The input string is processed sequentially from left to right, but the item set is empty before the first terminal input symbol is read. As in the Earley algorithm, the completer extends active edges when a symbol is identified, but the new predictor enters an active edge for each phrase that can have the identified symbol as its left-most constituent. Earley's predictor works top-down and finds all the lower-level constituents that can immediately follow in a phrase, while this predictor works bottom-up and finds all the <u>next</u> higher-level phrases which a given constituent can begin.

A formal statement of the algorithm follows. Note that the algorithm does not permit grammars with identical recursion (i.e. derivations of $A \stackrel{*}{\longrightarrow} B$ and $B \stackrel{*}{\longrightarrow} A$) or deletion (i.e. productions of the form $A \longrightarrow \varepsilon$). This formally restricts the class of context-free grammars that can drive the algorithm but does so in a way that is linguistically interesting (cf KILBURY 1984: 34).

```
program PARSE' a1...an:
begin
for i = 1 to n do CLOSURE' for [(i-1), i, ai, ai, ε]
end;

procedure CLOSURE' for [i, j, B, γ, δ]:
begin
add [i, j, B, γ, δ] to I;
if δ = ε then
   begin
   COMPLETE' with [i, j, B, γ, ε];
   PREDICT'  with [i, j, B, γ, ε]
   end
end;

procedure COMPLETE' with [j, k, B, γ, ε]:
begin
for each [i, j, A, α, Bβ] ∈ I do CLOSURE' for [i, k, A, αB, β]
end;
```

```
procedure PREDICT′ with [i, j, B, γ, ε]:
begin
for each <A, Bβ> ∈ P do CLOSURE′ for [i, j, A, B, β]
end.
```

As in the Earley algorithm above, the procedures COMPLETE′ and PREDICT′ are stated separately for the sake of clarity; they could be incorporated directly in CLOSURE′ without procedure calls.

PREDICT′ requires searches through the entire grammar for productions of the form A --> Bβ. In the case of large grammars for natural languages, such searches can be relatively costly and thus slow down the parsing process. This problem can easily be solved by numbering the productions and then constructing a <u>first-relation</u> that specifies which productions can introduce a given symbol as their left-most daughter. The relation is defined as F = {<B, n>}, where B is a terminal or nonterminal symbol, n is the number of a production, and <n, A, Bβ> ∈ P′. For the test grammar given above the first-relation would be F = {<NP, 1>, <det, 2>, <n, 3>, <VP, 4>, <v, 5>, <v, 6>}. If the first-relation is used, then PREDICT′ is modified in the third line:

```
for each <B, n> ∈ F where <n, A, Bβ> ∈ P′ do ...
```

Whether or not the first-relation is used, the algorithm generates the following items for the test grammar and input string given above:

```
1  [0, 1, det, det, ε]
2  [0, 1, NP, det, n]
3  [1, 2, n, n, ε]
4  [0, 2, NP, det^n, ε]     2 & 3
5  [0, 2, S, NP, VP]
6  [1, 2, NP, n, ε]
7  [1, 2, S, NP, VP]
8  [2, 3, v, v, ε]
9  [2, 3, VP, v, NP]
10 [2, 3, VP, v, ε]
11 [0, 3, S, NP^VP, ε]      5 & 10
12 [1, 3, S, NP^VP, ε]      7 & 10
13 [2, 3, VP, VP, adv]
```

14 [3, 4, adv, adv, ε]
15 [2, 4, VP, VP^adv, ε]   13 & 14
16 [0, 4, S, NP^VP, ε]   5 & 15
17 [1, 4, S, NP^VP, ε]   7 & 15
18 [2, 4, VP, VP, adv]

Both algorithms generate superfluous items (i.e. active edges that are not extended later in the analysis), but the new algorithm produces fewer. This fact is a consequence of the different depths of the predictions made during analysis. Earley's algorithm predicts top-down until a terminal symbol is reached, e.g. from S to NP to det; this algorithm predicts bottom-up to the next nonterminal symbol, e.g. from det to NP without continuing to S until the entire NP is recognized. Earley's predictor generates superfluous items when a nonterminal can be expanded with more than one production, while the new predictor is less efficient when a given symbol is the left-most daughter in more than one production. A consequence of this difference is that the new algorithm produces fewer items than Earley's for languages with relatively free word order, i.e. for grammars with sets of productions like A --> a b and A --> b a. 1*

The new predictor encounters difficulties with sets of productions like A --> a b and A --> b, as items 6, 7, 12, and 17 of the second test run above demonstrate. Once det has been identified as the beginning of an NP, there is no sense in predicting a new NP beginning with the following n. It is hard to solve this problem in a way that allows for recursive rules like A --> a A, which most linguists postulate for natural languages. PULMAN (1983) discusses this question in detail, but his proposed solution unfortunately amounts to a radical restriction of the class of grammars that can drive the parser, which is not the point of this discussion. Note that the Earley

---

1* The problem of parsing with the ID/LP formalism of GPSG, which is directly relevant for free word order, is dealt with in SHIEBER (1984), KILBURY (1984), and KILBURY (forthcoming). The adaptation of the new algorithm here to the ID/LP formalism results in a further reduction in the number of superfluous items.

algorithm avoids this difficulty. 2*

The factors mentioned here involve properties of languages and their corresponding grammars. Clearly, different algorithms may be more or less advantageous relative to such particular properties. This fact shows the futility of comparing parsing algorithms without reference to special properties of the grammars that drive them. The theory of parsing badly needs a comparison of parsing algorithms, but the method for such a comparison and the factors to be taken into account are largely unclear despite the pioneering work of GRIFFITHS/PETRICK (1965).

Another difference between the algorithms discussed here lies in their robustness. For example, a parser based on the new algorithm would produce no analysis for an input sentence with a word not contained in the lexicon, while the predictor of an Earley-based parser could form hypotheses about the new word based on syntactic (and semantic) expectations, produce an analysis, and even simulate the learning of new lexemes from context. For normal analysis Earley's predictor is too powerful and too expensive, but this extra power is desirable when unfamiliar vocabulary is encountered.

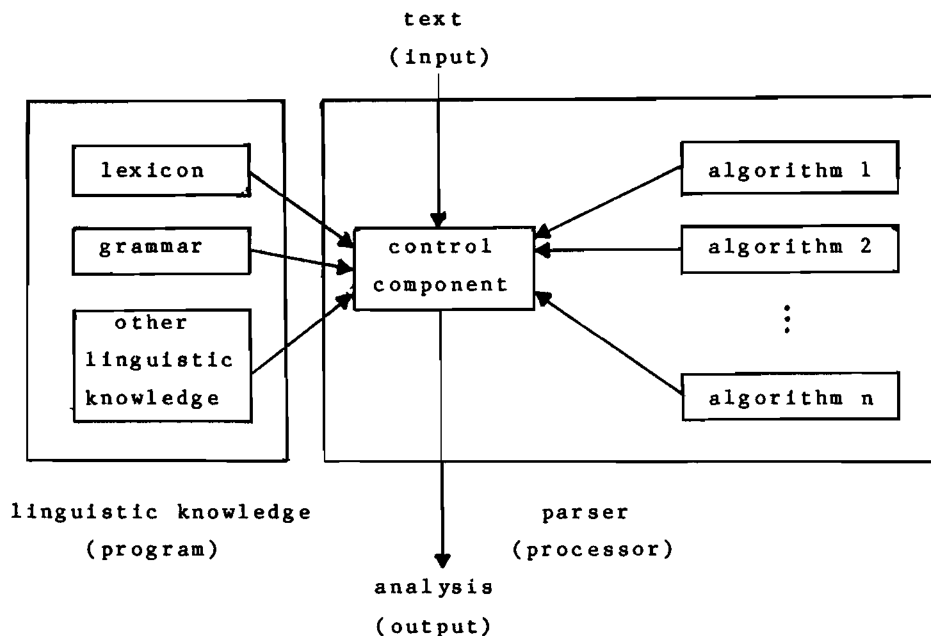
Here a distinction must be made between the parser components of software systems and parsing algorithms of the sort discussed in this paper. The latter are mathematical crystallizations of particular analysis strategies, while the former are practical programs dealing with a wide range of special problems. In view of all the distinctions presented above we might imagine a parser component with a repertoire of different parsing algorithms, i.e. a Multi-Algorithm Parser (MAP), operating with a common data structure and driven by a single grammar. A control component within the parser would call different algorithms depending on which was most advantageous at a given point in the analysis. Obviously, the design of such a control component raises major questions. It is unclear not only what factors make a particular algorithm advantageous but also how

---

2* Pulman must deal with this problem because his predictor, which is presented as that of Earley but in fact differs both from it and the new predictor here, uses a bottom-up strategy.

the control component should recognize that a particular factor arises at a given point. Furthermore, the relevant factors involve not only properties of the grammars but also other, nongrammatical linguistic knowledge.

The classical parsing algorithms are inefficient and psychologically implausible because they follow rigid strategies and make no use of knowledge about the communication situation. Developers of practical natural language systems are often criticized by computer linguists for mixing linguistic data with algorithms, but it is impossible to separate the two entirely at present. It must be seen as a challenge for linguists to establish what nongrammatical linguistic knowledge is relevant for parsing, how this knowledge is used in parsing, and how it can be formally represented. It then would be possible to drive a Multi-Algorithm Parser with a knowledge base containing different kinds of formally represented linguistic knowledge:



The diagram brings us back to the first figure of this paper and to the distinction of program and processor; a formal linguistic theory - now no longer restricted to grammar - is

again regarded as the definition of a programming language for a specialized processor (i.e. the parser). The ideas introduced here obviously require extensive elaboration before they can be realized in an actual system, but they may serve to suggest how such a system based on the distinctions stressed in this paper might appear.

References

AHO, Alfred V. / ULLMAN, Jefferey D. (1972): The Theory of Parsing, Translation, and Compiling. I: Parsing. Englewood Cliffs, N.J.: Prentice-Hall.

EARLEY, Jay (1970): "An efficient context-free parsing algorithm." Communications of the Association for Computing Machinery 13: 94-102.

GRIFFITHS, T. V. / PETRICK, S. R. (1965): "On the relative efficiencies of context-free grammar recognizers." Communications of the Association for Computing Machinery 8: 289-300.

KILBURY, James (1984): Earley-basierte Algorithmen fur direktes Parsen mit ID/LP-Grammatiken. KIT-Report 16. Berlin: Technische Universitat Berlin.

KILBURY, James (forthcoming): "A modification of the Earley-Shieber algorithm for direct parsing of ID/LP grammars." GWAI-84 Proceedings. Heidelberg: Springer.

KING, Margaret (ed.) (1983): Parsing Natural Language. London et al.: Academic Press.

PULMAN, S. G. (1983): "Generalised phrase structure grammar, Earley^s algorithm, and the minimisation of recursion." SPARCK JONES, Karen / WILKS, Yorick (eds.) (1983): Automatic Natural Language Parsing. Chichester: Harwood: 117-131.

ROSS, Kenneth M. (1982): "An improved left-corner parsing algorithm." Proceedings of the COLING 1982: 333-338.

SAMPSON, G. R. (1983): "Context free parsing and the adequacy of context-free grammars." KING (ed.): 151-170.

SHIEBER, Stuart M. (1984): "Direct parsing of ID/LP grammars." Linguistics and Philosophy 7: 135-154.

SLOCUM, Jonathan (1981): "A practical comparison of parsing strategies." Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, 1981: 1-6.

THOMPSON, Henry (1981): "Chart Parsing and rule schemata in PSG." Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, 1981: 167-172.

VARILE, G. B (1983): "Charts: a data structure for parsing." KING (ed.): 73-87.

WINOGRAD, Terry (1983): Language as a Cognitive Process. I: Syntax. Readings, Mass.: Addison-Wesley.

Kontextfreie Syntaxen und verwandte Systeme

Vorträge eines Kolloquiums in Ventron (Vogesen) im Oktober 1984

Herausgegeben von Ursula Klenk

Sonderdruck

Max Niemeyer Verlag
Tübingen 1985