The CYK-Approach to Serial and Parallel Parsing

Anton Nijholt

語學硏究 第27卷 第2號

1991년 6월

Language Research Vol. 27, No. 2.
LANGUAGE RESEARCH INSTITUTE
SEOUL NATIONAL UNIVERSITY
SEOUL, KOREA

The CYK-Approach to Serial and Parallel Parsing

Anton Nijholt

Traditional parsing methods for general context-free grammars have been re-investigated in order to see whether they can be adapted to a parallel processing view. In this paper we discuss sequential and parallel versions of the Cocke-Younger-Kasami (CYK) parsing algorithm. The algorithms that are considered turn out to be suitable for implementations ranging from environments with a small number of asynchronous processors to environments consisting of a large array of highly synchronized processors.

1. Introduction

In this paper we discuss versions of the Cocke-Younger-Kasami parsing algorithm that are suitable for implementations ranging from environments with a small number of asynchronous processors to environments consisting of a large array of highly synchronized processors. Before discussing these versions it is useful to go back to the traditional, serial CYK-like parsing algorithms. In the early 1960s parsing theory flourished under the umbrella of machine translation research. Backtrack algorithms were used for parsing natural languages. Due to the 'exponential blow-up' no widespread application of these algorithms could be expected. It was even thought that a polynomial parsing algorithm for context-free languages was not possible. Greibach (1979) presents the following view on the situation in the early 1960s: *'We were very much aware of the problem of exponential blow-up in the number of iterations (or paths), though we felt that this did not happen in 'real' natural languages; I do not think we suspected that a polynomial parsing algorithm was possible.'*

However, at that time a method developed by John Cocke at Rand Corporation for parsing a context-free grammar for English was already avail-

able. The method required the grammar to be in Chomsky Normal Form (CNF). An analysis of the algorithm showing that it required $O(n^3)$ steps was presented some years later by D. H. Younger. An algorithm similar to that of Cocke was developed by T. Kasami and by some others (see the bibliographic notes at the end of this paper). Presently it has become customary to use the name Cocke-Younger-Kasami* algorithm.

Parsing methods for context-free languages require a bookkeeping structure which at each moment during parsing provides the information about the possible next actions of the parsing algorithm. The structure should give some account of previous actions in order to prevent the doing of unnecessary steps. In computational linguistics a simple and flexible bookkeeping structure called **chart** has been introduced in the early 1970s. A chart consists of two lists of 'items'. Both lists are continually updated. One list contains pending items, i.e. items that still have to be tried in order to recognize or parse a sentence. Because the items on this list will in general be processed in a particular order the list is often called an agenda. The second list—called the work area—contains the employed items, i.e. things that are in the process of being tried. The exact form of the lists, their contents and their use differ. Moreover, they can have different graphical representations. In the folowing sections CYK-versions of charts will be discussed. An almost completely analogous approach can be given to Earley-like chart parsing methods.

2. Cocke-Younger-Kasami-like Parsing

The usual condition of CYK parsing is that the input grammar is in CNF, hence, each rule is of the form $A \rightarrow BC$ or $A \rightarrow a$. It is possible to relax this condition. With slight modifications of the algorithms we can allow rules of the form $A \rightarrow XY$ and $A \rightarrow X$, where X and Y are in V. However, we choose to present the algorithms under the assumption that the grammars are in CNF. Some algorithms will be followed by remarks which make clear how to relax the CNF condition further and how this affects the time complexity of the algorithms. Our first versions of the CYK algorithm are rather unusual, but they fit quite naturally in the framework of algorithms that will be discussed in this section.

* Or some other permutation of these three names.

CYK Algorithm (Basic Chart Version)

The first algorithm we discuss uses a basic chart with items of the form $[i, A, j]$, where A is a nonterminal symbol and i and j are position markers such that if $a_1 \cdots a_n$ is the input string which has to be parsed, then $0 <= i < j \leq n$. Number i is the start (or left) and number j is the end (or right) position marker of the item.

For any input string $x = a_1 \cdots a_n$ a basic chart will be constructed. We start with an empty agenda and an empty list of employed items.

(1) Initially, items $[i-1, A, i]$, $1 \leq i \leq n$ with $A \rightarrow a_i$ in P are added to the agenda.

Now keep repeating the following step until the agenda is empty.

(2) Remove any member of the agenda and put it in the work area. Assume the chosen item is of the form $[j, Y, k]$. For any $[i, X, j]$ in the work area such that $A \rightarrow XY$ is in P add $[i, A, k]$ (provided it has not been added before) to the agenda. Similarly, for any $[K, Z, l]$ in the work area such that $A \rightarrow YZ$ is in P add $[j, A, l]$ (provided it has not been added before) to the agenda.

If $[0, S, n]$ appears in the work area then the input string has been recognized. Obviously, for any item $[i, A, j]$ which is introduced as a pending item we have $A \Rightarrow^* a_{i+1} \cdots a_j$

Notice that in the algorithm we have not prescribed the order in which items are removed from the agenda. This gives freedom to choose a particular strategy. For example, always take the oldest (first-in, first-out strategy) or always take the most recent item (last-in, first-out strategy) from the agenda. Another strategy may give priority to items $[i, X, j]$ on the agenda we have $l \leq j$. It is not difficult to see that with a properly chosen initialization and strategy we can give the algorithm a left-to-right or a right-to-left nature. It is also possible to give preference to items which contain 'interesting' nonterminals (comparable with the 'interesting corner' chart parsing method discussed in Allport (1988)) or to use the chart in such a way that so-called island and bi-directional parsing methods can be formulated (cf. Stock et al. (1988)).

As an example of the basic chart algorithm consider the example grammar with rules

S→NP VP | S PP

NP→*n | *det*n | NP PP

PP→*prep NP

VP→*v NP

and with input string *the man saw Mary*. In the algorithm the symbols **det*, **n*, **prep* and **v* will be viewed as nonterminal symbols. The rules of the form **det→a*, **det→the*, **n→man*, etc. are not shown. Because of rule *NP→*n* the grammar is not in CNF. Step (1) of the algorithm is therefore modified such that items $[i-1, A, i]$ for $1 \leq i \leq n$ and $A \Rightarrow {}^*a_i$ are added to the agenda. See Fig. 1 for a further explanation of this example.

AGENDA	WORK AREA	
[0, *det, 1] [1, *n, 2] [1, NP, 2] [2, *v, 3] [3, *n, 4] [3, NP, 4]		Initially, [0, *det, 1], [1, *n, 2], [1, NP, 2], [2, *v, 3], [3, *n, 4] and [3, NP, 4] are put on the agenda (first row of this figure) in this order.
[1, NP, 2] [2, *v, 3] [3, *n, 4] [3, NP, 4] [0, NP, 2]	[0, *det, 1] [1, *n, 2]	The strategy we choose consists of always moving the oldest item first from the agenda to the work area. Hence, [0, *det, 1] is removed and appears in the work area. It can not be combined with another item in the work area, therefore the next item [1, *n, 2] is removed from the agenda and put in the work area. It is possible to combine this item with a previously entered item in the work area. The result [0, NP, 2] is entered on the agenda. The present situation is shown in the second row.
[0, NP, 2] [2, VP, 4]	[0, *det, 1] [1, *n, 2] [1, NP, 2] [2, *v, 3] [3, *n, 4] [3, NP, 4]	
[1, S, 4] [0, S, 4]	[0, *det, 1] [1, *n, 2] [1, NP, 2] [2, *v, 3] [3, *n, 4] [3, NP, 4] [0, NP, 2] [2, VP, 4]	Item [1, NP, 2] is moved from the agenda and put in the work area. This does not lead to further changes. The same holds for the two next items, [2, *v, 3] and [3, *n, 4], that move from the agenda to the work area. With item [3, NP, 4] entered in the work area we can construct item [2, VP, 4] and enter it on the agenda. We now have the situation depicted in the third row. Et cetera.
	[0, *det, 1] [1, *n, 2] [1, *NP, 2] [2, *v, 3] [3, *n, 4] [3, NP, 4] [0, NP, 2] [2, VP, 4] [1, S, 4] [0, s, 4]	

Fig. 1. Basic Chart Parsing of *the man saw Mary*.

As soon as a new item is constructed information about its constituents disappears. This information can be carried along with the items. For the example it means that instead of the items in the left column below we have those of the right column.

[0, NP, 2]	[0, NP(*det *n), 2]
[2, VP, 4]	[2, VP(*v NP(*n)), 4]
[1, S, 4]	[1, S(NP(*n VP(*v NP(*n))), 4]
[0, S, 4]	[O, S(NP(*det *n)VP(*v NP(*n))), 4]

In this way the parse trees become part of the items and there is no need to construct them during a second pass through the items. Clearly, in this way partial parse tress will be constructed that will not become part of a full parse tree of the sentence. Worse, we can have an enormous blow-up in the number of items that are distinguished this way. Indeed, for long sentences the unbounded ambiguity of our example grammar will cause a Catalan explosion. In general the number of items that are syntactically constructed can be reduced considerably if, before entering these extended items on a chart (see also the next versions of the CYK algorithm)they are subjected to a semantic analysis.

The name *chart* is derived from the graphical representation which is usually associated with this method. This representation uses vertices to depict the positions between the words of a sentence and edges to depict the grammar symbols. With our example we can start with the simple chart of Fig. 2.a.

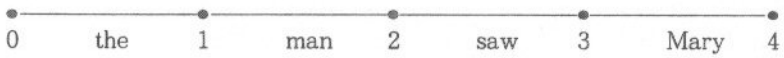


Fig. 2. a. The Initial CYK-like Chart.

By spanning edges from vertex to vertex according to the grammar rules and working in a bottom-up way, we end up with the chart of Fig. 2.b. Working in a 'bottom-up way' means that an edge with a particular label will be spanned only after the edges for its constituents have been spanned. Apart from this bottom-up order the basic chart version does not enforce a particular order of spanning edges. Any strategy can be chosen as long as we confine ourselves to the trammels of step (1) and step (2) of the algorithm.

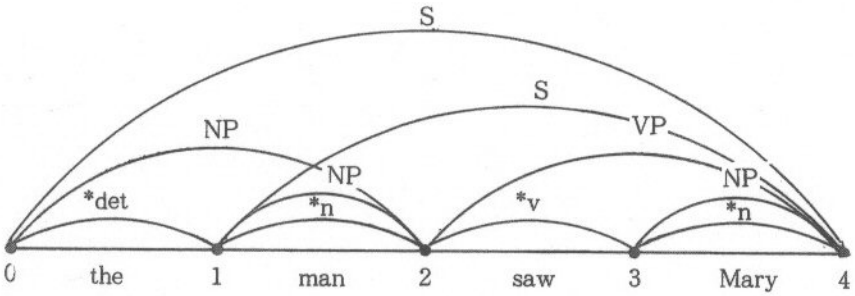


Fig. 2. b. The Complete CYK-like Chart for *the man saw Mary.*

In the literature on computational linguistics chart parsing is usually associated with a more elegant strategy of combining items or constructing spanning edges. This strategy will be discussed in a different paper on Earley-like parsing algorithms. It should be noted that although in the literature chart parsing is often explained with the help of these graphical representations, any practical implementation of a chart parser ('a spanning jenny') requires a program which manipulates items according to one of the methods presented in this paper.

We now give a global analysis of the (theoretical) time complexity of the algorithm. On the lists $O(n^2)$ items are introduced. Each item that is constructed with step (1) or (2) of the algorithm is entered on and removed from the agenda. For each of the $O(n^2)$ items that is removed from the agenda and entered on the list of employed items the following is done. The list of $O(n^2)$ employed items is checked to see whether a combination is possible. There are at most $O(n)$ items which allow a combination. For each of these possible combinations we have to check $O(n^2)$ pending items to see whether it has been introduced before. Hence, for each of the $O(n^2)$ items that are introduced $O(n^3)$ steps have to be taken. Therefore the time complexity of the algorithm is $O(n^5)$. The algorithm recognizes. More has to be done if a parse tree is requested. Starting with an item $[O, S, n]$ we can in a recursive top-down manner try to find items whose nonterminals can be concatenated to right-hand sides of productions. For each of the $O(n)$ nonterminals which are used in a parse tree for a sentence of a CNF grammar it will be necessary to do $O(n^3)$ comparisons. Therefore a parse tree of a recognized sentence can be obtained in $O(n^4)$time.

The algorithm which we introduced is simply adaptable to grammars

with right-hand sides of productions that have a length greater than two. However, since the number of possible combinations corresponding with a right-hand side increases with the length of the right-hand sides of the productions this has negative consequences for the time complexity of the algorithm. If ρ is the length of the longest right-hand side then recognition requires $O(n^{\rho+3})$ comparisons.

We are not particularly satisfied with the time bounds obtained here. Nevertheless, despite the naivity of the algorithm its bounds are polynomial. Improvements of the algorithm can be obtained by giving more structure to the basic chart. This will, however, decrease the flexibility of the method. It is rather unsatisfactory that each time a new item is created the algorithm cycles through all its items to see whether it is already present. Any programmer who is asked to implement this algorithm would try (and succeed) to give more structure to the chart in order to avoid this. Moreover, a more refined structure can make it unnecessary to consider all items when combinations corresponding to right-hand sides are tried. For example, when an item of the form $[j, Y, k]$ is removed from the agenda, then only items with a right position marker j and items with a left position marker k have to be considered for combination. Rather than having $O(n^3)$ comparisons for each item it could be sufficient to have $O(n^2)$comparisons for each item, giving the algorithm a time complexity of $O(n^4)$for grammars in CNF. Below such a version of the CYK algorithm is given. We will return to the algorithm given here when we discuss parallel implementations of the CYK algorithms.

CYK Algorithm (String Chart Version)

Let $a_1 \cdots\cdots a_n$ be a string of terminal symbols. The string will be read from left to right. For each symbol a_j an item set I_j will be constructed. Each item set consists of elements of the form $[i, A]$, where A is a nonterminal symbol and i, with $O \leq i \leq n-1$, is a (start) position marker.

For any input string $x = a_1 \cdots\cdots a_n$ item sets $I_1, \cdots\cdots, I_n$ will be constructed.

(1) $I_1 = \{[O, A] \mid A \rightarrow a_1 \text{ is in } P\}$.

(2) Having constructed $I_1, \cdots, I_{j-1}$, Construct I_j in the following way. Initially, $I_j = \phi$.

(2. 1) Add $[j-1, A]$ to I_j for any production $A \rightarrow a_j$ in P.

Now perform step (2. 2) until no new items will be added to I_j.

(2. 2) Let $[k, C]$ be an item in I_j. For each item of the form $[i, B]$ in I_k such that there is a production $A \rightarrow BC$ in P add $[i, A]$ (provided it has not been added before) to I_j.

It is not difficult to see that the algorithm satisfies the following property: $[i, A] \in I_j$ if and only if $A \Rightarrow^* a_{i+1} \cdots a_j$. A string will be recognized as soon as $[0, S]$ appears in the item set I_n. As an example consider the sentence *John saw Mary with Linda* generated by our example grammar. We allow again a slight modification of our algorithm in handling the rule $NP \rightarrow *n$. In Fig. 3 the five constructed item sets are displayed.

I_1	I_2	I_3	I_4	I_5
[0, *n]	[1, *v]	[2, *n]	[3, *prep]	[4, *n]
[0, NP]		[2, NP]		[4, NP]
		[1, VP]		[3, PP]
		[0, S]		[2, NP]
				[0, S]
				[1, VP]

Fig. 3. String Chart Parsing of *John saw Mary with Linda.*

Notice that each of the five item sets in Fig. 3 is in fact a basic chart, i. e., a data structure consisting of an agenda, a work area (the employed items) and rules for constructing items and moving items from the agenda to the work area. Except for item [3, *PP*], each application of step (2.2) in this example always yields one element only which is put in the item set and immediately employed in the next step. In general looking for a combination may produce several new items and therefore items in an item set may have to wait on an internal agenda before they are employed to construct new items. Therefore we can say that the algorithm uses a string of basic charts, which explains the name we have chosen for this CYK version. In a practical implementation it is possible to attach a tag to each item to mark whether it is waiting or not, or to put waiting items indeed on a separate list.

In the string chart version we have chosen a left-to-right order for constructing item sets. We could as well have chosen a right-to-left order corresponding with a right-to-left reading of the sentence. The reader will have no difficulties in writing down this string version of the CYK algorithm. With the sentence *John saw Mary with Linda* generated by our example grammar this version should produce the five item sets of Fig. 4 (first I_4, then I_3, etc.).

I_1	I_2	I_3	I_4	I_5
[*n, 1]	[*v, 2]	[*n, 3]	[*prep, 4]	[*n, 5]
[NP, 1]	[VP, 3]	[NP, 3]	[PP, 5]	[NP, 5]
[S, 3]	[VP, 5]	[NP, 5]		
[S, 5]				

Fig. 4. Right-to-Left Parsing of *John saw Mary with Linda.*

With the graphical representation usually associated with chart parsing the (left-to-right) string chart version of the CYK algorithm enforces an order on the construction of the spanning edges. In each item set I_j we collect all the items whose associated edges terminate in vertex j. Hence, a corresponding construction of the chart of Fig. 5 would start at vertex 1, where all edges which may come from the left and terminate in 1 are drawn, then we go to vertex 2 and draw all the edges coming from the left and terminating in 2, etc. As we noted above, whenever with step (2.2) of the algorithm an item $[i, A]$ in item set I_j leads to the introduction of more than one item in I_j, then there is no particular order prescribed for processing them. For the graphical representation of Fig. 5 this means that, when we draw an edge from vertex i to vertex j with label A, then there can be more than one edge terminating in i with which new spanning edges terminating in j can be drawn and the order in which these edges will be drawn is not yet prescribed.

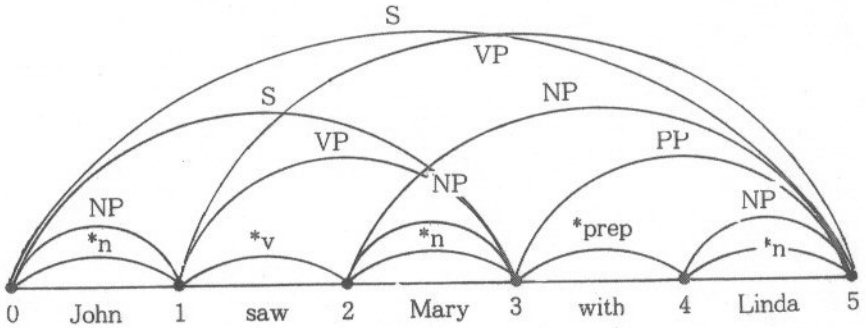


Fig. 5. The CYK Chart for *John saw Mary with Linda.*

Before discussing the time complexity of the algorithm we show some different ways of constructing the item sets. The string chart version discussed above can be said to be, in a modest way, 'item driven'. That is, in order to add a new item to an item set we consider an item which is already there and then we visit a previously constructed item set to look for a possible combination of items. The position marker of an item determines which of the previously constructed sets has to be visited. Without loss of correctness the string chart version can be presented such that in order to construct a new item the previously constructed item sets are visited in a strict right to left order. This will reduce the (slightly) available freedom in the algorithm to choose a particular item in the set that is under construction and see whether it gives rise to the introduction of a new item. Below this revised string chart version of the CYK algorithm is presented.

For any input string $x=a_1\cdots a_n$ item sets $I_1, \cdots, I_n$ will be constructed.

(1) $I_1=\{[0, A] \mid A\rightarrow a_1$ is in $P\}$.

(2) Having constructed $I_1, \cdots, I_{j-1}$, construct I_j in the following way. Initially, $I_j=\phi$.

(2.1) Add $[j-1, A]$ to I_j for any production $A\rightarrow a_j$ in P.
Now perform step (2.2), first for $k=j-1$ and then for decreasing k until this step has been performed for $k=1$.

(2.2) Let $[k, C]$ be an item in I_j. For each item of the form $[i, B]$ in I_k such that there is a production $A\rightarrow BC$ in P and $[i, A]$ (provided it has not been added before) to I_j.

A further reduction of the freedom which is still available in the algorithm is obtained if we demand that in step (2.2) the items of the form $[i, B]$ in I_k are tried in a particular order of i. In a decreasing order first the items with $i=k-1$ are tried, then those with $i=k-2$, and so on. A second alternative of the string chart version is obtained if we demand that in I_j first all items with position marker $j-1$, then all items with position marker $j-2$, etc., are computed. This leads to the following algorithm:

For any input string $x=a_1\cdots a_n$ item sets $I_1, \cdots, I_n$ will be constructed.

(1) $I_1=\{[0, A] \mid A\rightarrow a_1$ is in $P\}$.

(2) Having constructed $I_1, \cdots, I_{j-1}$, construct I_j in the following way. Initially, $I_j=\phi$.

(2.1) Add $[j-1, A]$ to I_j for any production $A\rightarrow a_j$ in P.
Now perform step (2.2), first for $i=j-2$ and then for decreasing i until this step has been performed for $i=0$.

(2.2) Add $[i, A]$ to I_j, if, for any k such that $i<k<j$, $[i, B]\in I_k$, $[k, C]\in I_j$ and $A\rightarrow BC$ is a production rule and $[i, A]$ is not already present in I_j.

We will return to this version when we introduce a tabular chart version of the CYK algorithm.

We turn to a global analysis of the time complexity of the algorithm. We confine ourselves to the first algorithm that has been presented in this subsection. The essential difference between this CYK version and the basic chart version is of course that rather than checking an unstructured set of $O(n^2)$ items we can now always restrict ourselves to sets of $O(n)$ items. That is, each item set has a number of items proportional to n. In order to add an item to an item set I_j it is necessary to consider $O(n)$ items in a previously constructed item set. Each previously constructed item set will be asked to contribute to I_j a bounded number of times (always less than or equal to the number of nonterminals of the grammar). Hence, the total number of combinations that are tried is $O(n^2)$. For each of these attempts to construct a new item we have to check $O(n)$ items that are available in I_j in order to see whether it has been added before. Hence, to construct an item set requires $O(n^3)$ steps and recognition of the input takes $O(n^4)$ steps. Parse trees of a recognized sentence can be constructed by starting with item $[0, S]$ in I_n and by checking I_1 and I_n for its constituents. This process can be repeated recursively for the constituents that are found. In this way each parse tree can be constructed in a top-down manner in $O(n^3)$ time. As demonstrated in the example, the CYK version discussed here is simply adaptable to grammars that do not fully satisfy the CNF condition or to grammars whose productions have right-hand sides with lengths greater than two.

Although the time bounds for this algorithm are better than for the previous CYK version we still can do better by further refining the data structure that is used. This can be done by introducing some list processing techniques in the algorithm, but it can be shown more clearly by introducing a tabular chart parsing version of the CYK method. This will be done in the next subsection. We will return to the string chart version when we discuss parallel implementations of the CYK algorithm.

CYK Algorithm (Tabular Chart Version)

It is usual to present the CYK algorithm in the following form (Graham and Harrison (1976), Harrison (1978)). For any string $x=a_1a_2\cdots a_n$ to be parsed an upper-triangular $(n+1)\times(n+1)$ recognition table T is constructed. Each table entry $t_{i,j}$ with $i<j$ will contain a subset of N (the set of nonterminal symbols) such that $A \in t_{i,j}$ if and only if $A \Rightarrow^* a_{i+1}\cdots a_j$. Hence, in this version the nonterminals are the items and each table entry is an item set. String x belongs to $L(G)$ if and only if S is in $t_{0,n}$ when the construction of the table is completed. Fig. 6 may be helpful in understanding the algorithm.

For any input string $x=a_1\cdots a_n$ an upper-triangular $(n+1)\times(n+1)$ table T will be constructed. Initially, table entries t_{ij} with $i<j$ are empty. Assume that the input string, if desired terminated with an endmarker, is available on the matrix diagonal (cf. Fig. 6).

(1) Compute $t_{i,i+1}$, as i ranges from 0 to $n-1$, by placing A in $t_{k,i+1}$ exactly when there is a production $A \rightarrow a_{i+1}$ in P.

(2) In order to compute $t_{i,j}$, $j-i>1$, assume that all entries $t_{k,l}$ with $l \leq j$, $k \geq i$ and $kl \neq ij$ have been computed. Add A to t_{ij} if, for any k such that $i<k<j$, $B \in t_{i,k}$, $C \in t_{k,j}$ and $A \rightarrow BC$ is a production rule and A is not already present in $t_{i,j}$.

Unlike the previous version of the CYK algorithm and unlike an algorithm of Yonezawa and Ohsawa (1988), which needed explicit representations of positional information, here this information is available in the indices of the table entries. For example, $t_{1,4}$ consists of all nonterminals that generate the substring of the input between positions 1 and 4. Notice that after step (1) the computation of the entries can be done diagonal by diagonal until entry $t_{0,n}$ has been completed. However, the exact order in which the entries are computed is not prescribed. Rather than proceeding diagonal by diagonal it is possible to proceed column by column such that each column is computed from the bottom to the top.

For each entry of a diagonal only entries of preceding diagonals are used to compute its value. More specifically, in order to compute whether a nonterminal should be included in an entry $t_{i,j}$ it is necessary to compare $t_{i,k}$ and $t_{k,j}$, for each k in between i and j. The order in which this may be done is, provided these entries are all available, arbitrary. Due to the systematic

a_1	0,1	0,2	0,3	0,4	0,5
	a_2	1,2	1,3	1,4	1,5
		a_3	2,3	2,4	2,5
			a_4	3,4	3,5
				a_5	4,5
					$

Fig. 6. The Upper-triangular CYK-table.

construction of each entry the need for an internal agenda has gone. The 'item driven' aspect from the previous versions has disappeared completely and what remains is a control structure in which even empty item sets have to be visited.

In Fig. 7 we have displayed the CYK table for example sentence *John saw Mary with Linda* produced by our running example grammar.

John	*n NP		*S*		*S*
	saw	*v	*VP*		*VP*
		Mary	*n NP		*NP*
			with	*prep	*PP*
				Linda	*n NP

Fig. 7. Tabular Chart Parsing of *John saw Mary with Linda*.

We leave it to the reader to provide a tabular chart version which computes the entries column by column (each column bottom-up) from right to left. Since the tabular CYK chart is a direct representation of the parse tree(s) for a sentence the strictly upper-triangular tables produced by this right-to-left version do not differ from those produced by the left-to-right version. Also left to the reader is the construction of the graphical represen-

tation according to the rules of this tabular chart version. For that construction it is necessary to distinguish between the column by column and the diagonal by diagonal computation of the entries.

In order to discuss the time complexity of this version observe that the number of elements in each entry is limited by the number of nonterminal symbols in the grammar and not by the length of the sentence. Therefore, the amount of storage that is required by this method is only proportional to n^2. Notice that for each of the $O(n^2)$ entries $O(n)$ comparisons have to be made, therefore the number of elementary operations is proportional to n^3. An implementation can be given which accesses the table entries in such a way that the time bound of the algorithm is indeed $O(n^2)$ time. This can be reduced to linear time if during the construction of the table with each nonterminal in $t_{i,j}$ pointers are added to the nonterminals in $t_{i,k}$ and $t_{k,j}$ that caused it to be placed in $t_{i,j}$. Provided that the grammar has a bounded ambiguity this can be done for each possible realization of a nonterminal in an entry without increasing the $O(n^3)$ time bound for recognition.

3. CYK-like Parsing in Parallel

In the previous section we gave several CYK-like parsing algorithms. Here we consider possible parallel implementations. A parallel implementation for the basic chart version can be given in the following way. In the algorithm there is one main loop: get an item from the agenda, put it on the list of employed items, try to combine it with items previously entered on this list and, if successful, put each newly created item on the agenda. If we have a number of processors at our disposal, then each processor can execute this loop. That is, we have a number of asynchronous processors and each processor picks up an item from the agenda, puts it on the list of employed items on the work area, cycles through this list and, after having verified that they have not been entered before, puts newly created items on the agenda. When it has finished the list it can pick up a new item from the agenda and start anew. More-fine-grained parallelism is possible by assigning two processors to each item which is picked from the agenda. If this item has the form $[j, Y, k]$, then one processor can search for combinations of the form $[i, X, j][j, Y, k]$, while the second processor searches for combinations of the form $[j, Y, k][k, Z, l]$. It is necessary to address the following problem in this parallel approach. Suppose processors P_1 and P_2 have entered items $[i, X, j]$ and $[j, Y, k]$ on the list of employed items and both start considering possible combinations. If there is a production $A \rightarrow XY$, then both processors may conclude that $[i, A, k]$ has not yet been entered on the agenda and both enter this new item. One possible way to avoid this is to assign increasing numbers to items that are entered on the agenda. When an item is entered on the list of employed items it is only allowed to make combinations with older (those with a smaller number) items. Another possibility is to reserve one processor for doing management tasks on the agenda and for scheduling tasks to free processors. This parallel implementation of the basic chart version resembles an implementation discussed in Grishman and Chitrao (1988).

Instead of discussing now the string chart version of the CYK algorithm we will first consider a parallel implementation of its tabular chart version. After that a parallel implementation of the string chart version will be derived. From the recognition table of the tabular chart version we can conclude a two-dimensional configuration of processors for a parallel implementation. For each entry $t_{i,j}$ of the strictly upper-triangular table there is a processors $P_{i,j}$ which receives table entries (i.e., sets of nonterminals) from processors $P_{i,j}$ and P_{j+1}. Processor $P_{i,j}$ transmits the table entries it receives from $P_{i,j-1}$ to $P_{i,j+1}$ and the entries it receives from $P_{i+1,j}$ to $P_{i-1,j}$. Proccessor $P_{i,j}$ transmits the table entry it has constructed to processors $P_{i-1,j}$ and $P_{i,j+1}$. Fig. 9 shows the interconnection structure for $n=5$.

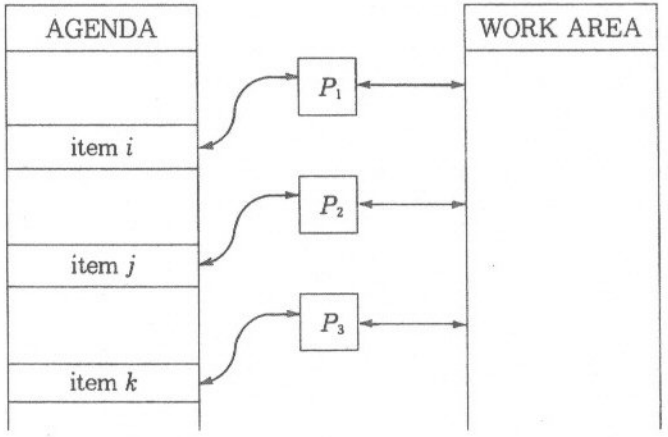


Fig. 8. Three Processors Working on the Basic Chart.

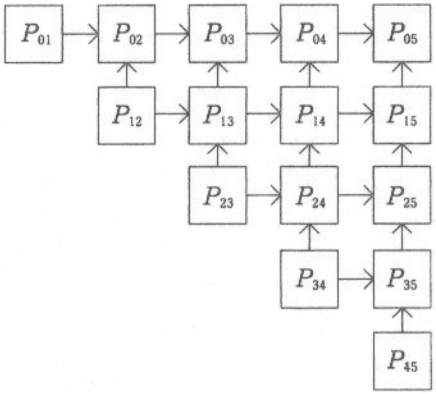


Fig. 9. Configuration for CYK's Tabular Chart Version.

More about the order of the communications between the processors will be said below. Each processor should be provided with a coding of the production rules of the grammar. Clearly, each processor requires $O(n)$ time. It is not difficult to see that like similar algorithms suitable for VLSI-implementation, e.g. systolic algorithms for matrix multiplication or transitive closure computation (see Guibas et al. (1979) and many others) the required total parsing time is also $O(n)$. In this particular case we can make the following observations. Consider processor $P_{i,j}$, with $j-i>1$. To compute $t_{i,j}$, $j-i-1$ matches have to be made. For reasons of symmetry we can assume that computation of table entries on the same diagonal will be finished in the same time. Therefore the matching pairs that are first available for a processor $P_{i,j}$ are $t_{i,k}$ and $t_{k,j}$ with k close to $(j-i)/2$. Let d be the largest natural number that satisfies the condition that it is less than or equal to $(j-i)/2$. Computation of entry $t_{i,j}$ by $P_{i,j}$ can then be done in the order shown in Fig. 10.

Now consider time steps such that in each step a processor can compute an entry $t_{i,j}$ with $j-i=1$ or can perform one matching and add the result to the table entry that is being constructed. With the given order of matchings each processor $P_{i,j}$ can start its computations at time step $P_{o,n}$ starts at time step n and is finished after $2n-2$ time steps.

The underlying assumption in this analysis is that at each time step each processor has at its disposal the table entries that have to be matched. One way to achieve this is to provide each processor $P_{i,j}$ with a local memory in

order	match entries $t_{i,k}$ and $t_{k,j}$w with	
	$j-i$ is even	$j-i$ is odd
1	$k=i+d$	$k=i+d$
2	$k=i+d-1$	$k=i+d+1$
3	$k=i+d+1$	$k=i+d-1$
4	$k=i+d-2$	$k=i+d+2$
.	.	.
.	.	.
.	.	.
$j-i-2$	$k=j+1$	$k=i+1$
$j-i-1$	$k=j-1$	$k=j-1$

Fig. 10. Computation Order of Processor $P_{i,j}$.

P_{38}		P_{28}		P_{39}		P_{38}	
ih	*iv*	*ih*	*iv*	*ih*	*iv*	*oh*	*ov*
35	58	25	58	36	69	36	58
36	68	24	48	35	59	35	48
34	48	26	68	37	79	37	68
37	78	23	38	34	49	34	38
		27	78	38	89	38	78

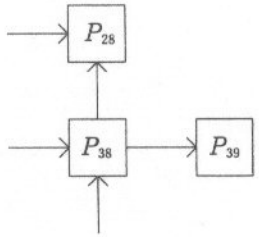
Fig. 11. Processor P_{38} and the Expectations of its Neighbors.

which the constructed table entries of row i and of column j will be stored. This requires $O(n^3)$ space. However, the order of communications can be chosen in such a way that only a few table entries have to be stored, while parsing time remains linear. To make this clear we need a few more observations. Assume, as in Fig. 9, that each processor has two input ports and two output ports. The input ports have the name *ih* (input port horizontal) and *iv* (input port vertical); the output ports are *oh* (output port horizontal) and *ov* (output port vertical). The order in which table entries should become available at the input ports of a processor $P_{i,j}$ is given by the computation scheme of Fig. 10. As a consequence, the output ports to its neighbors ($P_{i-1,j}$ and $P_{i,j+1}$) should transmit table entries in an order which allows them to make their matches in the proper order. Without going into details of a formal analysis it is possible to explain how this can be a-

chieved. Consider a processor P_{38}. In Fig. 11 we have given its scheme of matches and also the schemes for its upper and right neighbor.

From the input conditions of P_{28} and P_{39} it follows that P_{38} should transmit table entries to its neighbors on the next diagonal in the order displayed in the last table of Fig. 11. Hence, the *oh*-column of processor P_{38} becomes equal to the *ih*-column of P_{39} and the *ov*-column of processor P_{38} becomes equal to the *iv*-column of processor P_{28}. We can now merge the receive, the compute and the send tasks of processor P_{38} in the table of Fig. 12. Notice that after the final match table entry t_{38} has become available and can be sent to the neighbor processor.

Each processor $P_{i,j}$ finds $j-i-1$ times values at its input ports. It distributes, these entries together with the newly computed table entry $t_{i,j}$ to its neighbors.



P_{38}
ih 35 *iv* 58
match
oh− *ov*−
ih 36 *iv* 68
match
oh 36 *ov* 58
in 34 iv 48
mctch
oh 35 *ov* 48
match
oh 35 *ov* 48
ih 37 *iv* 78
match
oh 37 *ov* 68
ih− *iv*−
oh 34 *ov* 38
ih− *iv*−
oh 38 *ov* 78

Fig. 12. Compute and Communication Scheme of P_{38}.

step	P_{01}	P_{12}	P_{23}	P_{34}	P_{45}
1	ih−, iv− oh t_{01}, ov t_{01}	ih−, iv− oh t_{12}, ov t_{12}	ih−, iv− oh t_{23}, ov t_{23}	ih−, iv− oh $_{34}$, ov t_{34}	ih−, iv− oh t_{45}, ov t_{45}
	P_{02}	P_{13}	P_{24}	P_{35}	
2	ih t_{01}, iv $_{12}$ oh t_{01}, ov t_{02}	ih t_{12}, iv t_{23} oh t_{12}, ov t_{13}	ih t_{23}, iv t_{34} oh t_{23}, ov t_{24}	ih t_{34}, iv t_{45} oh t_{34}, ov t_{35}	
3	ih−, iv− oh t_{02}, ov t_{12}	ih−, iv− oh t_{13}, ov t_{23}	ih−, iv− oh t_{24}, ov t_{34}	ih−, iv− oh t_{35}, ov t_{45}	
	P_{03}	P_{14}	P_{25}		
3	ih t_{01}, iv t_{13} oh−, ov−	ih t_{12}, iv t_{24} oh−, ov−	ih t_{23}, iv t_{35} oh−, ov−		
4	ih t_{02}, iv t_{23} oh t_{02}, ov t_{13}	ih t_{13}, iv t_{34} oh t_{13}, ov t_{24}	ih t_{24}, iv t_{45} oh t_{24}, ov t_{35}		
5	ih−, iv− oh t_{01}, ov t_{03}	ih−, iv− oh t_{12}, ov t_{14}	ih−, iv− oh t_{23}, ov t_{25}		
6	ih−, iv− oh t_{03}, ov t_{23}	ih−, iv− oh t_{14}, ov t_{34}	ih−, iv− oh t_{25}, ov t_{45}		
	P_{04}	P_{15}			
5	ih t_{02}, iv t_{24} oh−, ov−	ih t_{13}, iv t_{35} oh−, ov−			
6	ih t_{01}, iv t_{14} oh t_{02}, ov t_{14}	ih t_{12}, iv t_{25} oh t_{13}, ov t_{25}			
7	ih t_{03}, iv t_{34} oh t_{03}, ov t_{24}	ih t_{14}, iv t_{45} oh t_{14}, ov t_{35}			
8	ih−, iv− oh t_{01}, ov t_{04}	ih−, iv− oh t_{12}, ov t_{15}			
9	ih−, iv− oh t_{05}, ov t_{34}	ih−, iv− oh t_{15}, ov t_{45}			
	P_{05}				
7	ih t_{02}, iv t_{25} oh−, ov−				
8	ih t_{03}, iv t_{35} oh t_{03}, ov t_{25}				
9	ih t_{01}, iv t_{15} oh t_{02}, ov t_{15}				
10	ih t_{04}, iv t_{45} oh t_{04}, ov t_{35}				
11	ih−, iv− oh t_{01}, ov t_{05}				
12	ih−, iv− oh t_{05}, ov t_{45}				

Fig. 13. The Order of Sending and Receiving Table Entries.

This is done in such a way that, in addition to table entry $t_{i,j}$, each processor needs to store four more table entries in order to achieve a delay necessary for transmitting them in the correct order. In this way each processor $P_{i,j}$ with $j-i>2$ performs the communications $(ih\ iv - -)(ih\ iv\ oh\ ov)^{j-i-2}(- -oh\ ov)^2$. Since each processor $P_{i,j}$ starts also at a time proportional with $j-i$ and since each processor takes $O(j-i)$ steps we may conclude that the algorithm takes $O(n)$ time. In Fig. 13 we present the receive and send scheme for parallel processing of sentences up to length 5. In this table the matches are not mentioned. In this figure it is assumed that when a processor sends an entry at step t it will be read at time $t+1$.

Finally we consider the string chart version of the CYK algorithm. Clearly, it has a left-to-right nature. An obvious parallel approach is the use of a pipeline configuration (Fig. 14), where each processor P_j computes a corresponding item set I_j. This can de done by distinguishing the same subsets of items that are used in the tabular chart version and by computing, for any processor P_j, first all items that would appear in $t_{j-1,j}$, then all items in $t_{j-2,j}$, etc., as we did in our last version of the string chart algorithm in the previous section. In this way processor P_j has to perform $O(j^2)$ matches and with a suitable list sutructure this translates into $O(j^2)$ elementary operations needed to compute item set I_j.

Each processor has an input port and an output port. Each time a processor receives a 'table entry' (i.e., a set of items) from its left neighbor it makes a match with a 'table entry' that is stored in its local memory and it stores the received 'table entry' until the right neighbor processor is ready to use it. Sending, receiving, and matching of 'entries' can be done such that total parsing time is $O(n^2)$ bounded and that each processor requires $O(n)$ memory. In Fig. 15 we have illustrated the sending, receiving and matching actions of two neighbor processors, P_5 and P_6. Notice that, in addition to the j 'table entries' that are computed by a processor P_j it has to store also at most j received 'table entries' until they can be transmitted to processor P_{j+1}.

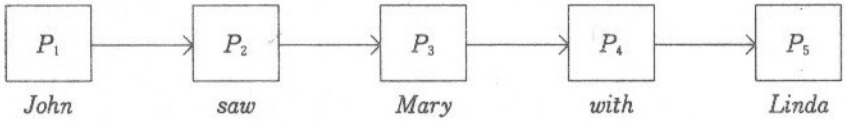


Fig. 14. Pipeline for CYK's String Version.

P_5			
step	in	match	out
1	—	t_{45}	t_{45}
2	t_{34}	t_{34} & $t_{45}(\Rightarrow t_{35})$	t_{34}
3	t_{23}	t_{23} & t_{35}	t_{35}
4	t_{24}	t_{24} & $t_{45}(\Rightarrow t_{25})$	t_{23}
5	t_{12}	t_{12} & t_{25}	t_{24}
6	t_{13}	t_{13} & t_{35}	t_{25}
7	t_{14}	t_{14} & $t_{45}(\Rightarrow t_{15})$	t_{12}
8	t_{01}	t_{01} & t_{15}	t_{13}
9	t_{02}	t_{02} & t_{25}	t_{14}
10	t_{03}	t_{03} & t_{35}	t_{15}
11	t_{04}	t_{04} & $t_{45}(\Rightarrow t_{05})$	t_{01}
12	—	—	t_{02}
13	—	—	t_{03}
14	—	—	t_{04}
15	—	—	t_{05}
16	—	—	—
17	—	—	—
18	—	—	—
19	—	—	—
20	—	—	—
21	—	—	—

Fig. 15. a. Pipeline Scheme for P_5.

Historical and Bibliographic Notes

Cocke's algorithm was first mentioned in Hays (1962). It was developed at RAND Corporation to become part of a system for the automatic syntactic analysis of English text. Algorithms similar to Cocke's algorithm were developed by Sakai (1962), Kasami (1965), and Kasami and Torii (1969). In Younger (1967) (based on an earlier report) an analysis of this algorithm was presented. Although reading these early papers is interesting it is more efficient to look in modern textbooks on formal language theory such as, e.g., Harrison (1978).

P_6			
step	in	match	out
1	—	t_{56}	t_{56}
2	t_{45}	t_{45} & $t_{56}(\Rightarrow t_{46})$	t_{45}
3	t_{34}	t_{34} & t_{46}	t_{46}
4	t_{35}	t_{35} & $t_{56}(\Rightarrow t_{36})$	t_{34}
5	t_{23}	t_{23} & t_{36}	t_{35}
6	t_{24}	t_{24} & t_{46}	t_{36}
7	t_{25}	t_{25} & $t_{56}(\Rightarrow t_{26})$	t_{23}
8	t_{12}	t_{12} & t_{26}	t_{24}
9	t_{13}	t_{13} & t_{36}	t_{25}
10	t_{14}	t_{14} & t_{46}	t_{26}
11	t_{15}	t_{15} & $t_{56}(\Rightarrow t_{16})$	t_{12}
12	t_{01}	t_{01} & t_{16}	t_{13}
13	t_{02}	t_{02} & t_{26}	t_{14}
14	t_{03}	t_{03} & t_{36}	t_{15}
15	t_{04}	t_{04} & t_{46}	t_{16}
16	t_{05}	t_{05} & $t_{56}(\Rightarrow t_{06})$	t_{01}
17	—	—	t_{02}
18	—	—	t_{03}
19	—	—	t_{04}
20	—	—	t_{05}
21	—	—	t_{06}

Fig. 15. b. Pipeline Scheme for P_6.

Chart parsing methods were introduced by Kaplan (1971, 1973) and Kay (1975, 1976, 1980). More recent explanations of chart parsing are much better to read. See e.g. Winograd (1984) and Thompson and Ritchie (1984).

In Chu and Fu (1982) and in Tan (1983) parallel aspects of the CYK algorithm are discussed and VLSI designs for this algorithm are presented. Similar methods have been introduced for Earley parsing Chiang and Fu (1982, 1984). These methods are in the tradition of algorithms for solving dynamic programming problems with arrays of processors (Guibas, Kung, and Thompson (1979)). Detailed accounts of parsing context-free languages with arrays of processors can be found in Chang, Ibarra and Palis (1987) and in Langlois (1988). The communication scheme which allowed us to store only a fixed number of table elements in each processor of the tabular chart version of the CYK algorithm is from Sijstermans (1986).

Chart parsing of unification grammars is discussed in Kay (1985) and in Haas (1987). Kay is concerned with Earley-like parsing while Haas in his paper considers parallel CYK-like parsing with the tabular chart and mentions a parallel implementation of the Earley-like tabular chart. Connectionist versions of the CYK algorithm and some extensions can be found in Fanty (1985), Nijholt (1990) and Sikkel & Nijholt (1990). In the latter paper an $O(\log n)$ time algorithm is presented.

References

Allport, D. (1988) 'The TICC: Parsing Interesting Text,' In: *Proceedings of the Second Conference on Applied Natural Language Processing*, Association for Computational Linguistics, Austin, Texas, 211-218.

Chang, J. H., O. H. Ibarra and M. A. Palis (1987) 'Parallel Parsing on a One-way Array of Finite State Machines,' *IEEE Transactions on Computers* 36, 64-75.

Chiang, Y. T. and K. S. Fu (1982) 'A VLSI Architecture for Fast Context-free Language Recognition (Earley's Algorithm),' *Proceedings Third International Conf. on Distributed Comp. Systems*, 864-869.

Chiang, Y. T. and K. S. Fu (1984) 'Parallel Parsing Algorithms and VLSI Implementations for Syntactic Pattern Recognition,' *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, 302-314.

Chu, K. H. and K. S. Fu (1982) 'VLSI Architectures for High-speed Recognition of Context-free Languages and Finite-state Languages,' Proceedings of the Ninth Annual Symposium on Computer Architectures, *SIGARCH Newsletter* 10, No. 3, 43-49.

Earley, J. (1968) 'An Efficient Context-free Parsing Algorithm,' Ph. D. Thesis, CMU, Pittsburgh, Also in: *CACM* 13 (1970) 94-102 and *CACM* 26 (1983) 57-61.

Fanty, M. A. (1985) 'Context-free Parsing in Connectionist Networks,' *Report RT* 174, Computer Science Department, University of Rochester, 508-519.

Graham, S. L. and M. A. Harrison (1976) 'Parsing of General Context-free Languages,' *Advances in Computers*, Vol. 14, M. Yovits and M. Rubinoff (eds.), Academic Press, New York, 76-185.

Graham, S. L., M. A. Harrison and W. L. Ruzzo (1980) 'An Improved Context-free Recognizer,' *ACM Trans. Progr. Lang. Syst. 2* (1980), 415-462.

Greibach, S. A. (1979) 'Formal Languages: Origins and Directions,' In: *20th Annual Symposium on Foundations of Computer Science*, 66-90, Also in: *Annals of the History of Computing* 3 (1981), 14-41.

Grishman, R. and M. Chitrao (1988) 'Evaluation of a Parallel Chart Parser,' In: *Proceedings of the Second Conference on Applied Natural Language Processing*, Association for Computational Linguistics, Austin, Texas, 71-76.

Guibas, L. J., H. T. Kung and C. D. Thompson (1979) 'Direct VLSI Implementation of Combinatorial Algorithms,' *Proc. Conf. on VLSI*, Caltech, 509-526.

Haas, A. (1987) 'Parallel Parsing for Unification Grammars,' In: *Proc. IJCAI-87*, 615-618.

Harrison, M. A. (1978) *Introduction to Formal Language Theory*, Addison-Wesley Publishing Company, Reading, Mass.

Hays, D. G. (1962) 'Automatic Language-data Processing,' In: *Computer Applications in the Behavioral Sciences*, H. Borho (ed.), Prentice-Hall, Englewood Cliffs, N. J.

Kaplan, R. M. (1971) *The MIND System: A Grammar Rule Language*, The Rand Corporation, Santa Monica.

Kaplan, R. (1973) 'A General Syntactic Processor,' In: *Natural Language Processing*, R. Rustin (ed.), Algorithmics Press, New York.

Kasami, T. (1965) 'An Efficient Recognition and Syntax Analysis Algorithm for Context-free Languages,' *Scientific Report AFCRL-65-758*, Air Force Cambridge Research Laboratory, Bedford, Mass.

Kasami, T. and K. Torii (1969) 'A Syntax Analysis Procedure for Unambiguous Context-free Grammars,'

Kay, M. (1976) 'Experiments with a Powerful Parse,' *American J. of Computational Linguistics*, Microfiche no. 43, Also: *2ème Conf. Internationale sur le Traitement Automatique des Langues, Grenoble, 1967*.

Kay, M. (1980) 'Algorithm Schemata and Data Structures in Syntactic Processing,' *CSL*-80-12, Xerox Palo Alto Research Center, Also in: *Proceedings of the Symposium on Text Processing*, Nobel Academy, 1984.

Kay, M. (1985) 'Parsing in Functional Unification Grammar,' In: *Natural Language Parsing*, D. R. Dowty et al. (eds.), Cambridge University Press, Cambridge, 251-278.

Langlois, L. C. (1988) '*Parallel Parsing of Context-free Languages on an Array of Processors*,' Ph. D. Thesis, *CST*-54-88, Department of Computer Science, University of Edinburgh.

Nijholt, A. (1989) 'Parallel Parsing Strategies in Natural Language Processing,' In: *Proceedings of the International Workshop on Parsing Technologies*, Carnegie Mellon University, 240-253.

Nijholt, A. (1990) 'Meta-parsing in Neural Networks,' In: *Cybernetics and Systems '90*, R. Trappl (ed.), Tenth European Meeting on Cybernetics and Systems Research, World Scientific Publishers, 969-976.

Sikkel, N. and A. Nijholt (1990) 'Connectionist Parsing of Context-free Grammars,' In: *Proceedings CSN '90 (Computing Science in the Netherlands)*, Utrecht.

Stock, O., R. Falcone and P. Insinnamo (1988) 'Island Parsing and Bidirectional Charts,' In: *Proceedings of the 12th Internationaal Conference on Computational Linguistics (COLING '88)*, Budapest, 636-641.

Sijstermans, F. W. (1986) 'Parallel Parsing of Context-free Languages,' Doc. No. 202, Esprit Project 415, Subproject A: Object-oriented Language Approach, Philips Research Laboratories, Eindhoven, The Netherlands.

Tan, H. D. A. (1983) 'VLSI-algoritmen voor herkenning van context-vrije talen in lineaire tijd. Rapport IN 24/83, Stichting Mathematisch Centrum, Amsterdam.

Thompson, H. and G. Ritchie (1984) 'Implementing Natural Language Parsers,' Chapter 9 in *Artificial Intelligence: Tools, Techniques, and Applications*, T. O'Shea and M. Eisenstadt (eds.), Harper & Row, Publishers, New York, 245-300.

Winograd, T. (1983) '*Language as a Cognitive Process*,' Vol. 1: Syntax, Addison-Wesley Publishing Company, Reading Mass.

Yonezawa, A. and I. Ohsawa (1988) 'Object-oriented Parallel Parsing for Context-free Grammars,' In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING '88)*, Budapest, 773-778.

Younger, D. H. (1967) 'Recognition and Parsing of Context-free Languages in Time n^3,' *Inform. and Contr.* 10, 189-208.

Faculty of Computer Science
University of Twente
P. O. Box 217, 7500 AE Enschede
The Netherlands