

1. The number of all the possible structures as a function of the length of the sentence

As soon as practical applications are considered the efficiency of the parsing method is of fundamental importance whether natural or programming languages are to be processed. The problem of efficiency arises because the relationship between the length of a string of symbols and the number of structures that may in theory be assigned to the string is far from linear, the growing number of symbols entails a much more rapidly growing number of possible structures.

For CF grammars it is comparatively easy to determine how the number of structures depends on the length of the string. Considering binary branchings only and excluding the possibilities that arise from having different labels attached to one node

$$f(n) = \frac{1}{2n-1} \binom{2n-1}{n-1}$$

is the number of different trees that can be assigned to a linear string of n elements [1]. This means that for a 10 element string the number of different trees is slightly less than 5000, for a 20 element string this number

becomes more than 1.75 milliard.

To include non-binary branchings as well I suggest the following recursive formula

$$\begin{aligned}
 g/1/ &= g/2/ = 1 \\
 g/n/ &= 2 \left[g/2/ \sum_{j=1}^{n-2} g/j/ + g/3/ \sum_{j=1}^{n-3} g/j/ + \dots \right. \\
 &\quad \left. \dots + g/n-2/ \sum_{j=1}^2 g/j/ + g/n-1/ g/1/ \right] + 1
 \end{aligned}$$

where n is the number of elements in the string.

Accordingly, more than 100 000 different structures can be assigned to a 10 element string, and 1.6×10^{12} different structures to a 20 element string.

Let us stress again that what we have calculated here is the number of the essentially different derivations, i.e. the number of those yielding different results. The number of possible derivational paths for 10 elements is 18 times larger than the number of the different results, for 20 elements the number of paths is 750 times larger than that of the different structures $\left(\frac{20!}{2} \quad 1.2 \times 10^{18} \right)$.

2. Syntactic ambiguities

Natural languages utilize but a small fraction of these possibilities. As to the number of possible structures of concrete sentences, the syntactic restrictions are very strong yet far from sufficient to yield information for unambiguous assignment. The number of structures allowed by the formal syntactic rules is in most cases definitely larger than the number of structures a human being becomes aware of in the course of speech.

A well-known point is that unambiguity cannot always be ensured by grammatical means even for artificial languages whose structure is immensely less complicated [2]. It is worth mentioning that the authors of ALGOL-68 decided to let some ambiguities remain in the language as it could have been eliminated but by making the grammar a lot more complicated [3].

Where does the majority of syntactic ambiguities in natural languages come from?

1. There is a number of words with varying scopes and vice versa: some words may fall within the scope of several different words and it cannot be determined by formal syntactic means - nor yet

by semantic ones at times - whose scope they really fall within. These two things often combine, especially in complex genitive constructions.

A fine Russian specimen of which is as follows:

...вследствие других законов сохранения и особенностей взаимодействия частиц...
 The corresponding string of symbols:

$Pr^G \quad A_g \quad N_g \quad N_g \quad E \quad N_g \quad N_g \quad N_g$

The rules of reductions:

$$/i/ \quad A_g + \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} = NP_g$$

$$/ii/ \quad \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} + \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} = NP_g$$

$$/iii/ \quad \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} + E + \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} = NP_g$$

$$/iv/ \quad Pr^G + \left\{ \begin{array}{c} N_g \\ NP_g \end{array} \right\} = C$$

Apparently a number of different structures can be determined by changing the order of rule application.

2. Another source of syntactic ambiguities is that not even the string of symbols /categories/ can always be unambiguously assigned to the sentence, i.e. homonymy may often appear on the morphological level. Homonymy arises either because formal differentiation between parts of speech is absent /e.g. in English/ or because the correspondence of the functions of words and the morphological means of expressing them is ambiguous, the morphological functions are not unambiguously expressed/e.g. in Russian/. Completely independent words with or without affixes too can of course agree in form.

Still one seldom comes across a sentence that could be assigned several entirely different structures. Sentences of this type are usually puns or grammatical examples /cf. "Time flies like an arrow"/. It is the so-called local syntactic ambiguity that normally troubles us, i.e. a part of the sentence that can be assigned several different part-structures without influencing the remainder of the sentence-structure. Now if there are several locally ambiguous parts in the sentence and they are independent from each other, the number of ambiguities for the whole sentence will considerably increase: it will be the arithmetic product of the numbers of independent local ambiguities.

3. Questions of tactics

The above numeric data clearly show how hopeless it is to simply proceed by checking on all the theoretically possible structures. But it is also apparent that syntax-directed parsing systems will fail in a considerable number of cases just because the sentence structure is syntactically undetermined [4]. The development of an effective analyzer is at least as much a mathematical as a linguistic problem.

The most important demands a parsing algorithm should meet are as follows:

/i/ It should be able to determine all the conceivable parsings that a given sentence is assigned by a particular grammar.

/ii/ It should be consistent in the sense that one parsing could not be arrived at but in one single way.
/It should be a 'one-to-one algorithm'./

/iii/ In some way or other it should counterbalance the immense growth of the number of possible structures. The purpose to be strived for is a linear relationship between the steps to be taken and the length of the sentence.

The efficiency of the algorithm depends considerably on factors that are independent of the particular method one has chosen to apply. These problems arise with any algorithm even if in different forms. The most important 'tactical' questions of this type are as follows:

/i/ Assuming a large set of rules how does the algorithm select the rules that are /possibly/ to be applied?

/ii/ How does it check for the conditions of applying them?

/iii/ How does it recognize 'blind alleys' i.e. illegal paths /if any/?

/iv/ How does it return from the illegal path to the legal one /or to the one that has not proved to be illegal as yet/?

Some of the well-known methods for solving /i/ are as follows:

/a/ Each rule is explicitly assigned the set of rules by which it could be continued. But choosing this method for a complicated grammar with a large number of possibilities one might face troubles.

/b/ The rules are divided into several groups on the basis of different characteristics such as the number or the character of the symbols within the rule etc. Searching is then carried out within a comparatively small set of rules.

/c/ Each symbol is assigned a set of all the rules this particular symbol appears in. Assignment can be done according to the position numbers within the rules. The so called initial symbols, i.e. symbols in first position, play then a distinguished role in the rule selection.

Whatever method one applies one may choose one of the several possible ways of practical realization. In case of /c/ the choice made will be of immense importance /e.g. rules arranged in matrix form, chainlike representation etc./.

Problems /i/ and /ii/ are strongly interconnected. How are we to decide whether the conditions of applying a rule are met?

In the case of CF grammars checking could be carried out quite easily. For top-to-bottom analysis all we have to do is the identification of the left-hand side symbol of the rule. For bottom-to-top analysis based on normal form CF rules /i.e. binary branchings/ only, once again it is not too difficult to check a twodimensional table for the possibilities of connecting a pair of symbols.

If general form CF or CS grammars are applied, the problem is not trivial at all, it turns out to be that of identifying strings of symbols. It could of course be solved in a trivial way but this would require an awful lot of work to do. B. Dömölki has developed a most

elegant method that would examine a whole series of rules at once. The checking is performed on Boolean vectors, and the point Dömölki has made an excellent use of is that computers carry out logical operations on all the bits of a machine word at the same time [5].

Two subproblems connected with checking rules should be discussed:

/a/ When should it start at all? Suppose that the symbol string is processed in sequential order /left-to-right or right-to-left/ and a possibly applicable rule or a given context should be checked for. Then we could either go back to symbols that have already been examined /and check them repeatedly when checking for the applicability of various rules/ or have already begun and completed certain examinations so that we finished checking by the time its result is needed. /The second solution could of course be applied only if an appropriate mechanism automatically provides the checking for the conceivable conditions and the /gradual/ cancelling of the non-realizable possibilities./

/b/ Is some kind of an additional examination necessary before the checking is completed? Namely it might turn out that the whole checking was superfluous because its result cannot be used later on or it will not lead to a correct result.

We have come very near to /iii/, i.e. to how the occasional impasses /blind alleys/ could be recognized in the course of the analysis? This is a cardinal problem concerning the efficiency of automatic analysis. The growing length of the sentence /symbol string/ entails not only a growing number of possible structures but the number of inappropriate part-structures growing as well. These 'torsoes' correspond to certain parts of the sentence but are incompatible with the remainder of it. What is more, the longer a sentence the more levels it may have i.e. the deeper its structure can be. This holds for the blind alleys as well: the longer the sentence the deeper the blind alley can be, the more branches and the more valid elements it may contain. Sentences that are monosemantic though syntactically ambiguous could be thought of as bottomless blind alleys not yet explored whose exploration needs either a wider context or the use of interrelationships not contained in the text.

The problem once again becomes twofold:

a/ What is the criterium of having got into a blind alley?

b/ How could we prevent getting into a blind alley at least in some cases?

The answer to these questions may be different, of course, for each algorithm and plays a subordinate though extremely important role regarding the "strategy" applied.

Just to give an example I would like to mention a most elegant method of defining and "calculating" the criterium of blind alleys using an algorithm built up in terms of logical vectors. Dömölki [5] -- who condenses the information related to the hypothetically accepted part structure and to the given symbol string under processing into a state vector defined recursively -- applies the following criteria to determine the impossibility of continuing the analysis along the given line

$$(T(Q_t) \vee B) \wedge H [x_{t+1}] = 0$$

Accordingly the new symbol x_{t+1} to be processed may neither continue the paths the previous vector of state contained that have proved possible so far, i.e.

$$T(Q_t) \wedge H [x_{t+1}] = 0, \text{ nor begin a new rule, i.e.}$$

$$B \wedge H [x_{t+1}] = 0.$$

The only handicap of Dömölki's method is that impasses can be recognized only after the algorithm has got into them -- the algorithm cannot pick out the paths that will lead into an impasse later on. So we have modified the algorithm and instead of using Dömölki's vector B - that would 'activate' the first position of each of the rules - we let only those of the rules become active that provide /direct or indirect/ continuation of the paths that have already proved to be legal [6].

Experience so far shows three practical methods of at least partial avoidance of impasses: /i/ taking into consideration the context; /ii/ making use of the transitive connectivity of the rules; /iii/ checking ahead the number of symbols not yet processed.

Taking into consideration the context means making use - if possible -- of only one direction of the context to avoid the repetition of the tests performed. Today such analyzing grammars play an important role in the analysis of artificial languages [7].

In my opinion making use of the transitive joining of rules has yet many important possibilities to offer. P. Z. Ingerman's analysis is a good example of experiments in this direction [8].

Taking into consideration the number of symbols not yet processed, saves the analysis many unnecessary tests. There have been attempts at doing a preliminary global analysis of the complete symbol string on this basis to assess in advance the possibilities of each path of the analysis [9].

Finally let us mention the question as the last of the questions of tactics:

/iv/ How to find the way from an illegal path back to a legal one?

This is the task that must somehow be solved by the parsing algorithm. So it is not enough to give a sign or "flag" at the points where the decision may perhaps be a failure. /i/ It must be ensured that the state prior to committing the error is reconstructed. /ii/ It would be advantageous to return to the state immediately prior to committing the error thus avoiding unnecessary delays.

(Nevertheless, there exist fine algorithms with no assurance that every error could be corrected. One of them is the well-known 'compiler compiler' that would never reinterpret a part of the symbol string if the part has once been accepted, consequently it is unable to recognize certain structures.)

One of the possible solutions to the problem in question is to have the "current state" of the analysis stored whilst proceeding so that it could be accessed later on. What we have termed "current state" here may include all the half-finished and abandoned rule applications that could be continued only after other rules have been applied. Whenever reaching back for a previous "current state" the possibilities that have ceased to exist in the meantime can always be cancelled. /The techniques followed for practical realization may vary depending on the amount of information to be stored, on the memory area available for the working fields, etc. (In most cases some kind of a push down store is applied.)

4. The strategy of analysis I.

The problems mentioned so far are common in varying degrees for all parsing systems, the ways they are solved have no decisive influence on the whole flow of analysis /though they are of decisive importance as far as efficiency is concerned/.

T.V. Griffiths and S.R. Petrick base the determination of the types of parsing systems on two considerations [10] /whilst stressing that 'some procedures are described in these terms only with difficulty' and 'others seem to allow no such classification'/:

/i/ In what direction does the parsing proceed - is it a top-to-bottom or a bottom-to-top analysis? /The third type mentioned - 'direct substitution algorithms' - is a subclass of the bottom-to-top algorithms./

/ii/ Does the algorithm apply any means of a preventive reduction of the number of blind alleys, i.e. for increasing the 'selectivity' of the algorithm?

Their most important findings concerning the efficiency of the different types of algorithms are as follows:

/a/ Algorithms proceeding from top to bottom - especially those of the direct substitution type - are the more efficient ones.

/b/ Methods of increasing selectivity are of no special importance in the case of top-to-bottom

analyses but they do considerably increase the efficiency in the case of bottom-to-top analyses. /c/ Efficiency is demonstratably influenced by the asymmetry /left-branching or right-branching/ of the structure to be analyzed. In the case of analysis proceeding from top to bottom it is influenced in the reverse direction if compared with the analysis proceeding from the bottom upwards. /We assume that the analysis proceeds either from right to left in both cases or from left to right in both cases./

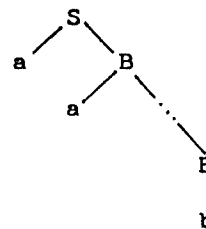
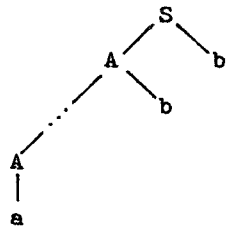
They considered the parsing time of the following sentence types:

ab^n	$a^n b$	$a^n b^n$	$ab^n cd$
left- branching	right- branching	embedding	compound
/'regressive' in Yngve's term/	/'progressive' in Yngve's term/		/left-branching with respect to recursivity/ ⁺

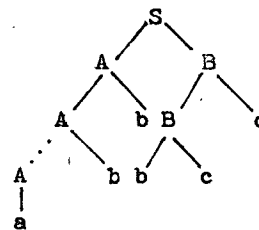
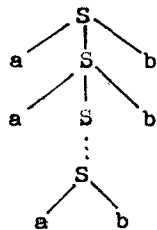
Parsing time as a function of sentence length increases - according to Griffiths' and Patrick's data - as follows:

⁺The grammar given would have allowed right recursivity as well / $ab^n cd^m$ / but in the measurements only the above restrictions of grammar are dealt with.

		ab^n		$a^n b$	
		non-select.	selective	non-select.	selective
top-to-bottom		q u a d r a t i c		l i n e a r	
bottom-to-top		l i n e a r		exponential	linear



		$a^n b^n$		$ab^n cd$	
		non-select.	selective	non-select.	selective
top-to-bottom		l i n e a r		e x p o n e n t i a l	
bottom-to-top		exp.	linear	exp.	cubic



According to Griffiths's and Patrick's data it is the bottom-to-top selective parser alone that is able to analyze sentences of the last, comparatively simple type grammar with a better than exponential efficiency.

What are the underlying reasons for the results obtained by Griffiths and Patrick?

/i/ Bottom-to-top algorithms are characterized by the fact that they take their start from what actually exists instead of looking for what "could be" [11].

In the case of exceedingly extensive grammars the top-to-bottom analysis must work with a huge number of potential possibilities and the elements of the symbol string to be analyzed will but slowly filter out the possibilities that may not be realized.

/ii/ Selectivity, in the sense Griffiths and Patrick use the term, does not influence all this to any degree as the filtering on the basis of a precedence-matrix extends only to testing the first element. It will be shown later on that selectivity can be considerably increased and, going even further, it could be made the basis of the strategy of the analysis.

/iii/ In the case of bottom-to-top analysis the situation is entirely different. Here the seemingly identical apparatus works with a much greater efficiency. But /a/ the "look ahead" condition suggested by Bastian [12]

/i.e. the possibility of the resultant symbol achieving its aim checks the compatibility/ one level higher up and the distance from the top is so much less. /b/ Here only such rules are to be realized in which all the components can be found, the others are omitted in the course of the rule controls. It is out of the question therefor to regard this selectivity as analogous with the top-to-bottom selectivity that is based on the first symbol of the lowest level.

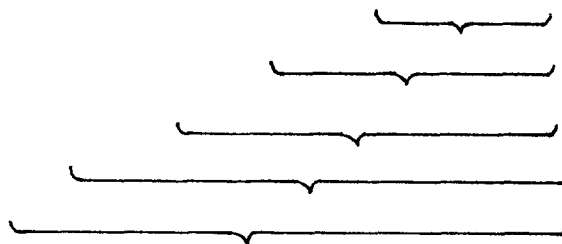
/iv/ Griffiths's and Patrick's measurements of the effect of the asymmetry of sentences on the efficiency of the analysis are a practical justification of an observation I made in 1964. In an article about Yngve's hypothesis [13] I developed the idea that for languages that have mostly "progressive" /right-branching/ structures it is the right-to-left analysis that is more effective in the case of analysis from bottom to top. /The right-to-left analysis is equivalent of course with a left-to-right analysis in a system that is a mirror image of the original./

In case of pure structures the explanation of the phenomenon is simple: In a right-branching structure the number of erroneous linkings is started at the end of a sentence. Let us take the example from the above mentioned article of mine:

Вы знаете много теорем о пределах.

Its processing from right to left is very simple:

Вы знаете много теорем о пределах



If, however, the analysis is started from the beginning of the sentence we get erroneous /or incomplete/ linkages again and again:

Вы знаете
знаете много
много теорем

In the case of complex structures the situation is more complicated. In this case the effectivity greatly depends on the method used for eliminating the impasses.

/On the disadvantages of vertical analysis see the next paragraph./

5. The strategy of analysis II.

When determining the type of analysis apart from its starting point it is also very important to know along what paths the analysis proceeds towards its goal, or in other words in what sequence the tests are carried out together with the inseparable question of in what form or structure the part-results are stored.

On the basis of these considerations there are two basic types of **parsers**. In theory this classification is independent of the fact whether the analysis proceeds from the bottom upwards or from the top downwards.

/i/ Those **parsers** that proceed with "maximum width" from level to level working on the full symbol string, first produce all the reductions that may be achieved by applying a single rule, then those that may be obtained by applying two rules and so on until the part-structures thus obtained are gradually linked. /In the analysis that proceeds from top downwards, these correspond to the derivations produced by applying two, three, ... rules, followed by the comparison of the terminal symbols thus obtained with the symbol string being analysed./

/ii/ The **parser**s that proceed with a "minimum width" and the "steepest slope", while gradually extending the

elements of the symbol string take the first opportunity to apply a rule and will not extend the analysis to a new symbol until there are new rules that could be built on the rules applied so far.

We could mention as an example for the first method the Sakai-Nagao algorithm [14] [15] the Cocke algorithm [16] or its application by Kuno to context sensitive languages [17] /the same strategy is applied by Vauquois in his analysis of Russian/. The algorithms by Woods [18] by Boracsev [19] and the Dömölki-Varga algorithms [5] [6] are examples of the second method.

Both methods have their advantages and disadvantages; perhaps it may be useful to draw the attention to them.

The great advantage of the analysis that proceeds from level to level is the ease with which in case of appropriate storage the part-analyses that could be continued along the same line, are contracted /see Griffiths-Petrick: "Merging similar sections of different TM [Turing Machine] paths"/.

Its disadvantage is the fact that

- a/ relatively large number of independent part structures has to be stored,
- b/ it needs relatively lengthy tests to determine whether the individual part structures are compatible.

The strategy of "maximal hierarchization" is more advantageous beyond doubt as far as economy in storage is concerned because in this case a single push down store will suffice to store the results and all the paths that have proved incorrect may be removed once and for all from the push down store together with all the derivations. This principle may be formalized as follows.

Let us denote according to inverse Polish notation the result of the rule applied to the elements

$a_k a_{k+1} \dots a_{k+r}$ with the result B_m as $a_k a_{k+1} \dots a_{k+r} B_m^r$. In other words let the elements of the symbol string that we applied the rule remain in the symbol string and let us simply add to the end of the string the symbol obtained as the result of the rule application.

Accordingly the resulting symbol string will be

$$\min_i a_1 \dots a_i B_i^{r_1} \quad r_1 \leq i$$

after applying the first applicable rule.

Let us suppose that there are at most $m-1$ more applicable rules following the first one while no new symbol is read $/m \geq 0/$

The symbol string will become

$$\max_m \min_i a_1 \dots a_i B_i^{r_1} \dots B_m^{r_m} \quad r_j \leq i$$

While continuing the application of this principle the symbol string will be increased by new terminal and non-terminal symbols:

$$\min_j \max_m \min_i a_1 \dots a_i B_1^{r_1} \dots B_m^{r_m} a_{i+1} \dots a_j^{r_j};$$

$$\max_n \min_j \max_m \min_i a_1 \dots a_i B_1^{r_1} \dots B_m^{r_m} a_{i+1} \dots a_j^{r_j} B_{m+1}^{r_{m+1}} B_n^{r_n}$$

.....

If the analysis gets into an impasse and cannot continue, then we have to return to the symbol B_s^r last applied, remove it and continue the analysis applying the above principle. /First an attempt is made at applying another permissible rule in the same place and only if this fails shall we take a new a_t symbol and continue the analysis./

The return from an impasse always means the deletion of the last non-terminal symbol and the reconstruction of the symbol string following it. /We would like to mention that this principle of analysis may be quite easily adopted to analyze context sensitive languages as well/.

This undoubtedly elegant principle of application produces the first possible analysis relatively rapidly, in its canonic form.

The increased selectivity of the analysis gives us a procedure that could be very well used in practical applications. Going further, having obtained the first

analysis if the analysis is continued on the same principles /just as if the first correct analysis were in an impasse/ all the other analysis may be likewise produced.

The disadvantages of the applied strategy of analysis are as follows:

/a/ If right at the beginning of the analysis we have taken an incorrect path, then the correction of this error may only be done after all the following and in part independent applications of the rules **have been deleted**. This means that the correct, or perhaps the only possible part-results are lost: after putting the error right they have to be **re-generated**.

/b/ The position is somewhat similar as far as the erroneous part-results are concerned: the analysis may get into a "local" impasse several times.

/c/ A new, different system of storage and searching must be provided if we wish to ensure a newer generation of the identical continuations -- supposing that previously some kind of a change took place in the determined structure.

6. A new strategy suggested for analyzing CF languages

The exponential increase in the time of analysis in various systems of analysis is obviously due to the increase in the number and depth of impasses, to their various branches -- in short to their dangerousness increasing with the length of the symbol string.

This is the dangerous point I tried to dodge by elaborating a parsing system that applies selectivity not as an additional device for increasing the efficiency of some method but as an independent method itself.

The linearity of the increase in the process of analysis may be best achieved if the symbol string to be analyzed can be segmented in accordance with the highest level rules applicable and these parts could be analyzed separately. If several parsings can be assigned to any of these segments /cf. what we have said about homonymy on p.5/ the structures corresponding to the whole sentence can be produced from the local part-results by combinatorical means.

Segmentation requires the following apparatus:

/i/ the transitive **initial** matrix of the rules

/B/a,n//

/ii/ the transitive continuation matrix of the rules

/C/a,n//

/iii/ the transitive end matrix of the rules $/E/a,n/$
 /iv_i/ the transitive initial matrix of the ith
 rule component $/B_i/a,n/$
 /v_i/ the transitive continuation matrix of the ith
 component $/C_i/a,n/$
 /vi_i/ the transitive end matrix of the ith component
 $/E_i/a,n/$
 /vii/ the matrix of the number of rule components
 $/N/i,n/$

The structure of the transitive initial matrix of
 the rules is almost the same as that of the so called
 precedence /or complete connectivity/ matrix. The
 differences show up in two facts, namely

- a/ the lines correspond to the terminal elements only
and not to all the elements of the vocabulary V;
- b/ the columns are assigned to rules of the grammar
and not to the symbols.

Thus it is a Boolean matrix $E/a,n/$; its element
 $E/a,n/$ is a truth function whose value is t if and only
 if the grammar allows the terminal symbol a to be the first
 element of the terminal rewriting of the nth rule.

The transitive continuation matrix of the rules
 $C/a,n/$ is a Boolean matrix whose element $C/a,n/$ is t if and
 only if the terminal symbol a is whichever but not the
 first element of the terminal strings of the nth rule.

The value of an element $E/a,n/$ of the transitive end

matrix of the rules is t if and only if the terminal symbol a can be the last element of the terminal strings of the n^{th} rule.

It follows from the definition that

$E/a_p, n/ = E/a_p, n/$ and $C/a_q, n/ = E/a_q, n/$
may occur but $E/a_p, n/ = E/a_p, n/ = C/a_p, n/$ may not.

The initial, continuation and end matrices of the rule components can be defined in a similar way, so it will be sufficient to give the definition of the initial matrix of the i^{th} rule component:

The value of an element $B_i /a, n/$ of the initial matrix of the i^{th} rule component $B_i /a, n/$ is t if and only if the non-terminal symbol a may be first element of the terminal strings of the i^{th} direct component of the n^{th} rule.

The line of thought of the algorithm is as follows:

Tests are carried out on two levels: on the level of inter-rule linkages /from top downwards/ and on the level of inter-terminal-symbol linkages /from left to right/. In each successive step of the test the individual components of the rules are made to correspond in the sequence of the components to a certain series of the terminal symbols of which the given component may be built up. By continuing this process finally either we arrive at the terminal ending in case of all components or the given segmentation is found

to be incorrect.

In case of incorrect segmentation first the permissible branches of the latest segmentation are tested by the algorithm. In our experience the selectivity of the system is considerable. Therefore even the storage of relatively small quantity of information allows a rapid examination of all the possibilities.

During segmentation we apply a "principle of segmentation" that is analogous to the principle discussed in connection with the "maximum hierarchization": the shortest component that is nearest to the beginning of the segment or to the end of the previous component, is taken and used until it becomes evident that for some reason the given segmentation is not applicable. In this case an attempt is made at solving the situation by shifting the last border of segmentation to the right: only if this leads to no result, is the previous border of segmentation changed. The outstanding effectivity of the method applied is due to

- a/ making best use of the bottle-neck for the reduction in analyzing time;
- b/ the fact that the tests for the possibilities of various part-segmentations can be quickly performed;
- c/ the possibility of testing each segment in complete separation from all the other segments;
- d/ the fact that the twosided approach leads to much fewer unnecessary part results than either Cock's or the well-known top-to-bottom algorithms.

Bibliography

- 1 Berge, C. Théorie des graphes et ses applications, Dunod, Paris, 1958.
- 2 Ginsburg, S. The Mathematical Theory of Context-Free Languages, McGraw Hill, New York, 1966.
- 3 Algol-68. MR 95. /mimeographed/
- 4 Shrejder, Ju. A. Teorija tolerantnosti, Nauchno-
Texniceskaja Informacija, Ser. 2
- 5 Dömölki, B. Voprosy sintaksicheskogo analiza dlja
formal'nyx jazykov, Computational Linguistics 5,
pp. 41-93.
- 6 Varga, D. Problems of Machine Analysis, Linguistica
Antverpiensia II. pp. 415-428.
- 7 Knuth, D.E. On the Translation of Languages from
Left to Right, Information and Control Vol. 8,
No 6, pp. 607-639.
Kaufman, V. Sh. O raspoznavanii nekotoryx svojstv
kontekstno-svobodnyx grammatik, I-ya Vsesojuznaja
konferencija po programirovaniju, Kiev, 1968.
- 8 Ingerman, P.Z. A Syntax-Oriented Translator, Academic
Press, New York, 1966.
- 9 Unger, S.H. A Global Parser for Context-Free Phrase
Structure Grammars, Comm. ACM, Vol. 11, No 4, pp.
240-247.
- 10 Griffiths, T.V., Petrick, S.R. On the Relative
Efficiencies of Context-Free Grammar Recognizers,
Comm. ACM, Vol. 8. No 5, pp. 289-300.
- 11 Cf. Vakulovskaja, G.V., Kulagina, O.S. Ob odnom al-
goritme sintaksicheskogo analiza ruskix tekstov,
Problemy kibernetiki 18, p. 218.

- 12 Bastian, L. A Phrase-Structure Language Translator, AFCL Rep. 62-549, AF Cambridge Research Labs., Bedford, Aug. 1962.
- 13 Varga, D. Yngve's Hypothesis and Some Problems of the Mechanical Analysis, Computational Linguistics 3, pp. 47-72.
14. Sakai, I. Syntax in Universal Translation, Proc. 1961 Internat. Conf. on MT of Languages and Applied Language Analysis, London, 1962, pp. 593-608.
- 15 Nagao, M. Studies on Language Analysis Procedure and Character Recognition, Kyoto University, 1965.
- 16 Cf. Hays, D.G. Automatic Language-Data Processing, Computer Applications in the Behavioral Sciences, Prentice-Hall, Englewood Cliffs, N.J., 1962, pp. 394-421.
- 17 Kuno, S. A Context-Sensitive Recognition Procedure, NSF-18, Aug. 1967. VII-1-28.
- 18 Woods, W.A. Context-Sensitive Recognition, NSF-18. Aug. 1967. VIII-1-23.
- 19 Borscsev, V.B., Efimova, E.N., O sokrashchenii perebora pri sintaksicheskom analize, Nauchno-Tekhnicheskaja Informacija, 1967, No 10, pp. 27-33.