Antonín Říha - Svatava Machová

COMPUTER TESTING OF A GENERATIVE GRAMMAR

1.1. The emergence of materially elaborated variants of generative description of languages has brought about a pragmatical question as to whether a particular generative grammar is working in such a way as it is assumed to be. This problem proved to be unsolvable by a linguist not aided by computer. Therefore, programmes started to be written, so as to enable the computers either *a*) to make a grammar work or *b*) to allow to find out – through a recognition procedure, e.g. changing the directions of arrows in all the rewriting rules – whether an initial *S* can really be reached when starting from assumed output strings.

1.2. In the Computing Centre of Charles University, Prague, the former of the above methods of computer testing of generative grammars was selected – similarly as at the University of Michigan (J. Friedman, 1971) and at other research centres. Tests are being worked out for a certain variant of functional generative description of the Czech language (the author of its frame is P. Sgall). For the time being, the object of the testing is the generative component (GC) of this description enumerating semantic representations (SR's) of sentences.

1.3. Before demonstrating the way in which the testing is carried out we shall give a brief outline of the basic properties of the GC of the functional generative description. For a more detailed characterization of this type of description see P. Sgall, *et al.* (1969).

The GC is a context-free phrase structure grammar, i.e. a grammar of the type 2 in Chomsky's classification of grammars. Together with the theory of immediate constituents the dependency syntax finds its application in it. In the GC there are several types of context-free rules: modifying, substitutional and selectional ones. They are shown illustratively in Fig. 1, where *U*, *V*, *W* are auxiliary non-terminal symbols, *u* is a terminal symbol, *r* is some functor – terminal symbol indicating which of the two non-terminal symbols on the right-hand side of the rule is the dependent one and which is the governing one as well as the type of dependency. For the sake of simplicity each of the terminal and non-terminal symbols is represented by a single letter.

$$\begin{tikzcd}[arrows=-]
 & \textit{modifying rules} & U \text{ --- } (V\ r\ W) \\
\textit{rewriting rules} \arrow[ur] \arrow[r] \arrow[dr] & \textit{substitutional rules} & U \text{ --- } V \\
 & \textit{selectional rules} & U \text{ --- } u
\end{tikzcd}$$

Fig. 1.

In the written form of the grammar, non-terminal symbols of the grammar are ordered $(n + 2)$-tuples $X, X_0, X_1, \ldots, X_n$ where X is the so-called name-symbol, i.e. a name shared by a certain class of non-terminal symbols, and $X_0, \ldots, X_n$ are indices specifying individual non-terminal symbols of that class. Terminal symbols are characterized by a similar structure. From a linguistic point of view the name-symbols in terminal symbols correspond to the so-called lexemes, grammatemes and functors. For the time being the lexemes represent semoglyphs, their total number in the variant tested being 275. Thus this experiment ranks among those experiments operating with a "lexicon" of a small extent which is typical so far of most computer experiments with generative grammars. One name-symbol corresponds to the left bracket and one to the right one.

A substitution of some units for others is often possible in Czech in certain contexts only. The GC meets this fact generally by introducing new, more refined categories specified by indices. The number of indices actually used differs with the individual name-symbols, the brackets and some other name-symbols having no indices at all. The maximum number of indices attached to one non-terminal name-symbol is 15; the maximum number of indices attached to one terminal name-symbol is 30; the maximum number of values of each individual index is 94 and the average number of values per one index is 8.

The right-hand side of the rules in the GC is subject to certain restrictions, namely the right-hand side of the rules does not contain more than two non-terminal symbols. Hence we work with a binary context-free grammar. The GC contains recursive rules. The order of the rules applied is determined by the form of the rule itself and by a selection of non-terminal symbols. The rules are not distinguished as obligatory and optional ones. However, the answer to the question whether it is necessary, or only possible, to use some rule in generating some SR of a sentence is given by the form of the rules itself. To make the representation brief and to increase the legibility, rule-schemes are largely made use of.

2.1. The form of grammar was maintained in a shape close to the original one, with which the linguists are used to work; it makes a good orientation in the grammar possible. Some modifications, however, were introduced.

The values of the indices were coded with natural numbers and some designations were employed, such as so-called references, which make it possible to register only once the lists of index-values and even whole non-terminal symbols that occur more than once, and to refer to these values in other cases; they also make it possible to describe the fact that the value of some index is determined by a value of some other index, etc. These lists, symbols and values are usually referred to by means of references in the frame of a single rule-scheme so that these references do not cause any serious slowing down of the work of the programme and at the same time they save the storage space.

The leftmost-derivation method is used, as well as a random choice of alternatives. It was, however, necessary to avoid, e.g., a repeated choice of some recursive rules, which would lead either to an excessive prolongation of the string generated, possibly without any transition to terminal symbols, or to a situation where the number of some types of SR's in the generated sample would be far removed from their actual frequency in the language. Therefore, the alternatives on the right-hand sides are not picked out simply at random but with a certain prescribed probability. As a consequence of the use of rule-schemes it is not possible to prescribe probabilities for the schemes of the right-hand sides as wholes only, and it is necessary to prescribe also probabilities for various values of indices used in these schemes. The prescribed probabilities make it possible to control the derivation of the SR's so as to make the generated sample contain, first of all, some strings of a particular type chosen a priori, which we intend to examine more closely. Thus it will be possible to change the set of the generated strings by means of a change of the probabilities prescribed. In other words: by a modification of probabilities of some subset of rules it will always be possible to meet the demand: " Let the given phenomenon occur in SR's with much higher (or lower) frequency! ". However, an answer to the question: "What will be the effect of a modification of the probability in the rule number n_i upon the generation of SR's?" can only be given after evaluating the further experiment of random generation in which the probability modification intended will be materialized.

The programme makes use of a special subroutine to obtain pseudorandom numbers.

An example of a modifying rule:

VERBUM 0 = 18 7 = 1 8 = 0,1 $9 = 13 →
→ 40 (*NP* 0 = 9(20),10 2 = 1 3 = *L*8 *RD* *LS*)

VERBUM, *NP*	are non-terminal name-symbols
RD	is a functor – terminal name-symbol
8 = 0,1	means that this rule can be used when the value of index 8 with given non-terminal name-symbol is 0 or 1
$9 = 13	means that the rule can be used for non-terminal name-symbol *VERBUM* when index 9 either has the value 13 or is not used
40	a prescribed probability for choice of this alternative (other alternatives are not quoted here for this example)
0 = 9(20),10	means that in 20 % of cases the value 9 is to be chosen, in the rest of the cases the value 10
3 = *L*8	the value of index 3 with a non-terminal name-symbol *NP* will equal the value of index 8 of the rewritten non-terminal symbol
LS	reference; it means that at this place the whole left-hand side should be repeated (i.e. *VERBUM* with all its indices).

Some quantitative characteristics of the tested variant:

(The given average values are mere estimates based on partial calculations, exact values will be reached on the computer in the course of transducing the grammar into a form suitable for the work of the programme).

The number of non-terminal name-symbols	62
the number of indices with one non-terminal name-symbol	15 max, 5 average

the number of possible values of one index with one non-terminal name-symbol	94 max (with the so-called determinations) 34 max (in other cases) 8 average
the number of terminal name-symbols	304
the number of « proper » terminal name-symbols (corresponding to lexemes)	275
the number of indices with one terminal name-symbol	30 max (with verbal lexemes) 7 max (with noun lexemes) 3 average
the number of possible values of one index with a terminal name-symbol	94 max (with the so-called determinations) 18 max (in other cases) 7 average
the number of schemes of left-hand sides for one name-symbol	79 max 6 average
the total number of schemes of left-hand sides	460
the number of schemes of right-hand sides for one left-hand side	20 max 2 average
the total number of schemes of right-hand sides	1010
the number of all name-symbols of one right-hand side	6 max 3 average

3. In giving information on research work, including linguistics, a great deal depends on the way chosen for the presentation of the results (A. MARTINET, 1970). In the field of computer treatment of

*

grammar we have not found it very easy to choose the most adequate way for transferring the acquired knowledge and we assume to have reached this goal only partially.

3.1. The most essential work involved in the transduction of the grammar into a shape which is convenient for the work of the computer was left to the GRAMMAR TRANSDUCER programme which also performs the input check of the representation.

The programmes are being prepared for a computer of the type IBM 360 or 370 and will be written in PL/1 programming language.

3.2. The whole grammar will be written on a magnetic disk as one file by means of the GRAMMAR TRANSDUCER programme. This file will contain records of variable length. One record will always contain information about all left-hand sides with the same name-symbol and about the corresponding right-hand sides.

The structure of a record on a magnetic disk:

LS_1	RS_{11}	RS_{12}	...	RS_{1n_1}	LS_2	...	LS_k	RS_{k1}	...	RS_{kn_k}

LS_i ... information about a left-hand side (about indices, their values and their probabilities)

RS_{ij} ... information about right-hand sides

having the form

P	W_1	W_2	...	W_m

,

where W_i describes the word of the right-hand side, i.e. the code of its name-symbol, indices and their values, and the probabilities

P stands for the probability of the selection of the given scheme of the right-hand side

Fig. 2.

The main programme DERIVATION OF SR's and the generated string will be stored in the internal storage of the computer. The programme reads a corresponding record of the file on the disk storage, using the code of the non-terminal name-symbol as a key. The read record is processed directly in the buffer with the use of based variables. First, an appropriate left-hand side of the rule is found, then a corresponding right-hand side of the rule, and the substitution is carried out.

It follows from the shape of the rules that approximately $\frac{1}{4}$ out of the total number of symbols in the generated string are the so-called semoglyphs. An analysis of the process of generation has shown that, if the generated string contains *N* semoglyphs, it was necessary to use, in the course of its generation, about 5 *N* to 8 *N*-times a substitution rule, *N*-times a selectional rule and (*N*-1)-times a modifying rule. E.g. when simulating a generation of a string of 13 symbols – containing 3 semoglyphs – 23 substitutions were carried out, which corresponds well to the estimation according to the method described above. We can thus estimate the number of readings from the external storage of the computer in the course of the generation of one string.

The maximum length of a string will be determined by the dimension of the storage space declared for this string. The probabilities prescribed for the choice of the right-hand sides of the rules ensure that a premature finishing of the process of generation caused by exceeding a given length may occur only in exceptional cases.

On the output of the programme there will be two output files containing generated SR's. One will be on the magnetic tape and will serve as an input for the programme for the next (transductive) component of the generative description, the other will be a print file serving for checking up the results.

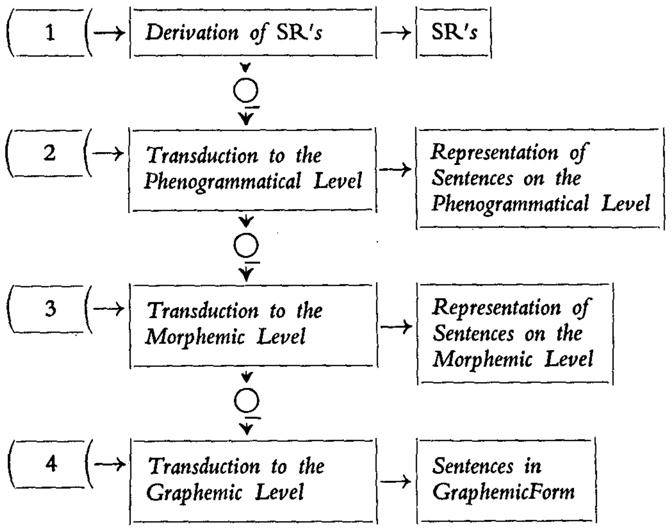
Possible changes and corrections of the rules of the grammar will be carried out by means of a programme called GRAMMAR MODIFIER, which will carry out the changes in the file stored on the magnetic disk according to the data on punched cards.

In the present moment, we have finished the rewriting of the grammar into the shape which will be punched on the cards. Some basic problems have already been solved, e.g. the shape of the data has been decided upon and the algorithm of the generation of SR's has been formulated. The programming itself will be done during the autumn of this year.

A system of programmes for testing the whole generative grammar is illustrated by means of a flow-chart diagram in Fig. 3.

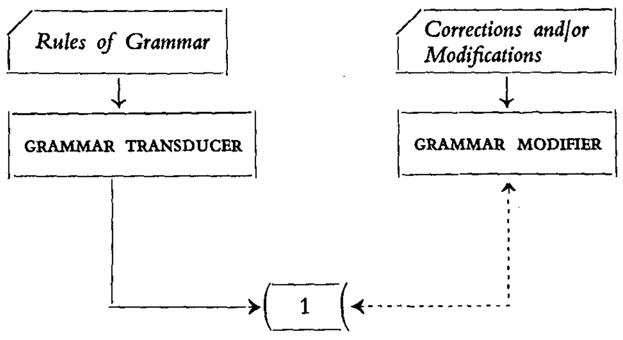
The diagram of the data preparation for a programme corresponding to the GC is shown in Fig. 4. The other data files will be prepared in an analogous way.

4. The main contribution of the computer testing of a generative grammar is usually seen in the fact that the linguist acquires knowledge on the interaction of the rules of grammar (J. FRIEDMAN, 1971). Moreover, the preparation of the data for the computer calls for more accu-


1
Derivation of SR's
SR's
2
Transduction to the Phenogrammatical Level
Representation of Sentences on the Phenogrammatical Level
3
Transduction to the Morphemic Level
Representation of Sentences on the Morphemic Level
4
Transduction to the Graphemic Level
Sentences in GraphemicForm

1 ... *Rules of grammar*
2,3,4 ... *Rules of transduction*

Fig. 3



Rules of Grammar
Adapted for Computer

Fig. 4

rate formulations and solutions of some questions, which would otherwise be neglected as less important.

We also find some value in the fact that the results of computer testing of any particular generative grammar can help even a linguist who is not precisely familiar with the theory in question: he can promptly verify his own views, or those of somebody else, on linguistic properties of the sequences generated, and gain useful information for his own further work.

REFERENCES

J. Friedman *with* T. H. Bredt, R. W. Doran, B. W. Pollack, T. S. Martner, *A Computer Model of Transformational Grammar*, New York, 1971.

A. Martinet, *Analyse et Présentation: Deux Temps du Travail du Linguiste*, in *Linguistique Contemporaine. Hommage à E. Buyssens*, Bruxelles, 1970, pp. 133-139.

P. Sgall, L. Nebeský, A. Goralčíková, E. Hajičová, *A Functional Approach to Syntax in Generative Description of Language*, New York, 1969.