István Bátori

WORKING WITH THE INTERACTIVE VERSION OF THE T.G.T.-SYSTEM OF JOYCE FRIEDMAN

The present paper does not claim to be a description of the TGT-System, since it was already presented by Professor Friedman herself at the International Conference on Computational Linguistics in Stockholm in 1969. In addition the system has been described also in the book JOYCE FRIEDMAN, *A Computational Model of Transformational Grammar*, Elsevier, 1971. Our intention is to present the new interactive version of the TGT-System, which has been developed at the Basic Research of IBM Germany, and to show how it can be used in linguistic research.

In order to appreciate the present interactive version, it will be, however, necessary to recall some essential aspects of the TGT-System, yet we do not want to discuss the Friedman System as such in a systematic fashion.

Accordingly, in the first part of the paper I shall talk about the batch version, and about our experiences with the system and then I procede to the interactive version.

1. THE TGT-SYSTEM IN GENERAL

The TGT-System of Friedman grew out of the necessity to verify or control a formal grammar. It becomes increasingly difficult to control any formal system beyond a certain size: if one wishes to follow the interaction of two or three abstract rules with all their implications,

The new interactive version of the System has actually been installed at the C.N.U.C.E. in order to enable the participants of the Conference to see the systems as it works. I take this opportunity to thank the organizers of the conference, the C.N.U.C.E., and particularly Professor Faedo, Torrigiani, and Zampolli, once again, for their generous support of the demonstration. I also thank my collegues Mrs. Schirmer, Miss Zoeppritz and Mr. Henning, who assisted me to prepare the demonstration. I am especially indebted Dr. Picchi, who adopted the interactive version to the local CMS-System.

he may still use his head; for a dozen rules, he will need paper and pencil; and for hundreds of rules, he must have a computer.

As primary objective Friedman wanted to give a computational aid to the transformationally oriented linguists. Her system as it stands now can, however, be considered also as an attempt to formalize the transformational grammar in the strict mathematical sense as well. The basic intention of Friedman was not to argue for a specific type of generative grammar but rather to offer a framework as general as possible and let the linguist impose restrictions on his particular grammar. However, it cannot be overlooked that the starting point of Friedman is clearly CHOMSKY's *Aspects*-model.

Accordingly, it is easy to learn how to work with the TGT-System if you are familiar with transformational theory. On the other hand, you can use to system " to learn " transformational grammar, as a tutorial aid. Since we do not want to discuss either the transformational grammar directly, nor the purely technical details of Friedman's System, please, let me presume familiarity with the basic notions of generative grammar and refer for the purely notational conventions once again to Friedman's book.

2. THE FORM OF THE GRAMMAR

The form of the Grammar is strictly prescribed, but as already mentioned, it is very close to current transformationalist notation.

For the TGT-System a grammar consists of a *phrase structure, a lexicon*, and a *transformational part*. In the first phase of the processing the grammar is built up according to the users specifications and in the second, subsequent phase one sentence (or more) are constructed according to the grammar. Each of these major components is subdivided further into smaller units. The structuring of the Grammar is indicated by keywords, which must be used in certain positions and are anticipated by the System.

Let me shortly comment on some points of this scheme of grammar.

3. THE TREATMENT OF THE CONTEXTUAL FEATURES

Friedman introduced a new type of feature, called contextual, which comprises Chomsky's strict subcategorization and selectional restrictions; i.e., it is all the same for further processing, whether a contextual rule involves features, like (a) or just category symbols, like (b) in (Fig. 1.).

Contextual Rule

(a) *HUMSBJ* = ⟨ *S*/ ⟨ # (*ART*) *N* | + *HUMAN* | %___%⟩⟩ ·
(b) *DIROBJ* = ⟨ *VP* ⟨ % *NP*___⟩⟩.

Lexical Entries

⋮

Key | + *N* .... − *HUMAN* |,
Student | + *N* .... + *HUMAN* |,

⋮

Imagine | + *V* .... + *HUMSBJ* ...|,

Lexical Insertion

1. Category *N V* ... 2. Category *V N* ....

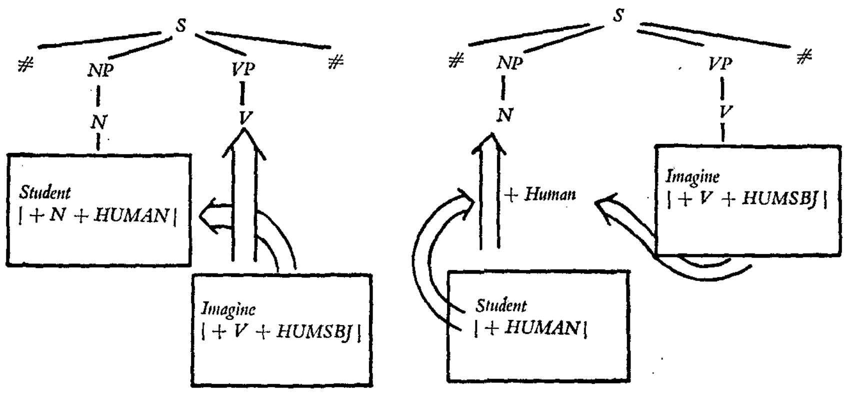

Fig. 1. *Side Effects*

But apart from this simplification, the treatment of these contextual features is significantly different from that of CHOMSKY's in the *Aspects*-model. The main innovation is the concept of the " side effects ", which makes the selectional rules independent of the order of inserting lexical items into the derivational tree.

If the contextual feature refers to a node (or nodes) to which a lexical entry has already been attached, (as in (1) on Fig. 1) the program checks the compatibility of the item with its environment, just as in the *Aspects*-model. If on the other hand (as in (2), Fig. 1) the node referred to in the contextual rule is still empty, the new item is introduced and the consequences of the contextual features, i.e. the feature on which the insertion depends, are projected into the invironment.

4. THE REPRESENTATION OF TREES

Note that the output trees are leaned on the side to simplify printing. In addition the nodes are numbered for ease of reference. These numbers can be used, among others to localize the feature, which belong to a specific node, also with higher, non-terminal nodes. Note also that features coming from the lexicon are associated originally with the lexical entries. After lexical insertion they are adjoined to the immediately dominating category node and not to the actual word any more.

5. THE FORM OF TRANSFORMATIONS

In comparison with the Phrase Structure Rules the notational conventions for transformations are less uniform. The notational unsteadiness is largely due to the lack of a strict, mathematically founded and universally accepted transformational theory.

There are two notational styles in use; the more popular of them is the MIT-Style. (Fig. 2).

Verbal Description of Passive:

1. EXCHANGE SUBJECT AND OBJECT
2. INSERT THE WORD *BY* AS LEFT SISTER OF THE AGENT
3. MARK THE MAIN VERB AS PAST PARTICIPLE

M. I. T. – Notation:

$$\left.\begin{array}{llllllll} SD & \# & NP & X & PASS & V & NP & X \\ & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}\right\} ===>$$

$$\begin{array}{llllllll} SC & 1 & 6 & 3 & 4 & \begin{bmatrix} 5 \\ +PART \end{bmatrix} & BY+2 & 7 \end{array}$$

MITRE – Notation:

```
SD % 1 # 2NP % PASS 3V 4NP %,
SC (PREP < BY > ) ACHLE 2,
   | + PART | MERGEF 3,

          4  ALESE 2,
          2  ARISE 3.
```

Abbreviations

ALESE	...add left sister
ARISE	...add right sister
MERGEF	...merge feature
ACHLE	...add by Chomsky adjunction to the left

Fig. 2. *Writing Transformations*

This convention for transformation is generally advocated in standard introductory works. Accordingly, transformations are written in the form of pseudo-rewriting rules, where apparently, the structural description (SD) part should be replaced by the structural change (SC) part. With other words: you define the input and you define the output. The convention is self explanatory, but perhaps somewhat vague. The MIT notation is regarded even by its own adherents rather as a convenient short-hand for indicating structural changes and not as a proper, full scale formalism.

The other style is the MITRE-notation, which is less known and resembles computer commands. This convention defines the input into a transformation and lists the elementary operations, to be carried out on the input tree. The elementary operations should be defined in advance. On the whole this way of representing transformations is more abstract but it can be formalized more readily. Friedman uses this style of notation: there is no problem to reformulate a transformation from the pseudo-rewriting style into the operational representation.

6. " THE TRAFFIC RULES "

The purpose of the *control program* (CP) is to determine in which order, and at which point in a derivational tree, a transformation should be applied.

By means of a FORTRAN-like control language (by the so called " traffic rules "), the linguist can execute the transformations cyclically, i.e. applying the same set of transformations to every clause, he can determine in which order the clauses of a sentence should be processed, he may change the order of execution depending on certain condition, e.g. on the success of preceding transformations etc. This control part of Friedman's System provides an enormous generative power, the possibilities of which have hardly been discussed in the linguistics. You can easily define several successive transformational cycles by the CP of Friedman, you can solve the ordering problem of transformations by defining unique jumps in order to leave out the execution of a transformation, which in a " simple ", cyclically ordered grammar would be impossible.

7. USING THE SENTENCE GENERATOR

The actual testing of the Grammar is done by the Sentence Generator. As already said the Grammar is laid on in the first phase of the processing and subsequently the system should be instructed to generate sentences according to the given grammar. Trivially in as much as the system generates correct sentences, the grammar is verified to the extent the generated sentences are false, the grammar is wrong and has to be corrected.

The *sentence generator* as such can operate in one of three nodes (Fig. 3):

1. *Random Sentences:*

```
S
```

A random sentence will be generated

2. *Predefined Structures:*

```
S  #
   NP  ART  THE
       N    BOY
   VP  AUX  MOD   Q
            TNS   PRS
            V     PASS
       V    READ
       NP   PRON  WHAT
   #
```

The structure will be operated on according to your grammar

3. *Directed Random Generation:*

```
S  RES
   NDOM
   Q
```

A random sentence will be generated with the restriction that it will not dominate a Q-node

Fig. 3. *Types of Input into the Sentence Generator*

1. It can generate sentences completely at random, where a random number generator mechanism controls the selection of grammatical rules and lexical insertion. All you have to do is to enter the sentence symbols *S*.

2. You can predefine a sentence entirely at the level of deep structure and let the system check the tree and carry out the transformations leading up to the surface structure.

3. You can use partially defined input, e.g. defining just the structure, but leaving open the lexical insertions, or just specifying a particular structural configuration you are interested in, while letting the system fill up the rest at random.

For practical testing the second and the third way of using the sentence generator is clearly preferable. The random generator may produce spectacular sentences, but practically never the ones which have bearing on the problem you are interested in. The sentences delivered by the random generator may be and are revealing, and nobody experimenting with the system would withstand the temptation to see what his grammar would produce " left entirely alone ", but it is not suitable for systematic work. You may correct a mistake detected by the random generator, but you better test the correction by a predetermined skeleton, otherwise you may get a totally different sentence, from which you cannot see whether the error has really been corrected or not.

According to our experience, entirely predetermined structures including lexical entries are the best to test a grammar. In this case you can anticipate a normal sentence as the final output of the generator, and can immediately decide whether the generation is correct or not.

There are two input formats: a free, bracketed (FTRIN) format, and a fixed tree format (TRIN). It is perhaps a matter of personal taste, yet for us the FTRIN, that is the bracketed input, seemed to be more convenient. (Fig. 4)

FTRIN Format:

```
S ⟨ # NP ⟨ PRON ⟨JEMAND⟩⟩ VP ⟨ NP⟨ART⟨D⟩ N⟨BUCH⟩⟩
      V ⟨ LES⟩⟩ MOD ⟨ V ⟨HAB⟩ TNS⟨ PRS⟩⟩ # ⟩.
```

TRIN Format:

```
S   #
    NP     PRON   JEMAND
    VP     NP     ART  D
                  N     BUCH
           V      LES
    MOD    V      HAB
           TNS    PRS
    #
```

Fig. 4.

Usually, the interaction of the phrase structure rules is fairly straightforward, while that of the transformational rules is much more intricate. Therefore you can easily predefine a skeleton by using your own phrase structure rules " manually " and then let the system apply the transformations to the prefabricated input. If you use partially predetermined trees, you may be distracted by mistakes, which occur at places which are of no interest to you. Note that you cannot correct all errors, at least not at once, and therefore you had better concentrate on a few points, otherwise you loose sight of you own grammar.

8. THE OUTPUT OF THE BATCH VERSION

The original batch output of the TGT-System has been designed to provide all possible information about the processing, which the linguist may possibly need. First the input grammar is listed, followed by the content of the major internal tables, according to which the subsequent generation procedes. Then, the process of sentence generation is reported in such a manner, that the linguist can follow the significant steps of the processing (Fig. 5 (1)).

9. THE INTERACTIVE VERSION OF THE TGT-SYSTEM

The present interactive version has been developed according to the experiences gained by working with the original batch version. We have noticed in general that we are interested in the linguistic aspects of the derivation, such as changes in the tree, or in the final output, but not the actual computation.

The demand for a more condensed output will be even more imperative in a terminal environment where the time and the output should be restricted to a minimum. Therefore we defined a new additional output file, containing just the essential information in which a linguist is interested (Fig. 5).

The original batch protocol enables you to follow the actual flow of computation, e.g. in the case of a transformation you get the modules called to perform the successive steps of the processing. The interplay of the different subroutines is, however, always the same: *ANTEST* calls *PASSIV*, *PASSIV* calls *ELEMOP* etc. Since Friedman's System works practically free of error, there is no need to check the subroutine

The Transformations as formulated in the Grammar:

```
TRANS 1 PASSIV " PASSIVBILDUNG " | OB.
SD ≠ 1NP 2NP 3V 4V | + PASS | %.
SC (PREP | + DAT | ⟨VON⟩ ) AFIDE 1,
      | + PART | MERGEF 3,
            2 ALESE 1.
```

The Derivational Tree to be Manipulated on:

```
1 S    2 ≠
       3 NP      4 ART     5 D
                 6 N       7 STUDENT3
       8 VP      9 NP     10 PRON        11 WAS
                12 V      13 LES
      14 MOD    15 V      16 WERD
                17 TNS    18 PRS
                19 Q
      20 ≠
```

The Report on the Successful Completion:

1. *The original batch protocol*

```
ANTEST CALLED FOR   2" PASSIV "(AC  ) ,SD = 7. RESTRICTION = O.
TOP = 1 : S
ANTEST RETURNS ** 1 **
CHANGE. CALL ELEMOP FOR AFIDE    21  3
CHANGE. CALL ELEMOP FOR ALESE     9  3
```

2. *The new interactive protocol*

```
PASSIV  AFIDE     21     3      ADDED SUBTREE: 21 IS HEADED BY PREP
PASSIV  MERGEF  12   + PART
PASSIV  ALESE      9     3
```

Fig. 5. *The Protocol of the Transformations*

calls every time. This information, therefore, can be dispensed with for the most purposes.

We have designed a slightly different, more comprehensive format, which contains only the linguistically relevant information. The new output format of the interactive version makes a clear reference to the input grammar, such as the name of the transformation, the name of the elementary operations, the nodes affected by them. In one point the interactive version provides information, which has not been explicitly reported in the original batch version. You can follow now also the feature operations in the same form as you follow the tree operations: the interactive protocol delivers the features names and the actual feature value. For a linguist testing feature operations this is an innovation over the original batch version, which suffices to give a hint at this point, that the feature operation has been successfully completed without further details.

It should be noted, that batch-output and terminal output are not mutually exclusive, the terminal output is a summary extracted from the original and placed on a separate file output. The original output is, however, still available. The file on which is written is normally set dummy, but it can be reactivated and listed, in the very same form as in the original version.[1]

10. THE COMMANDS OF THE INTERACTIVE VERSION

The interactive version on the whole uses a fairly straightforward language. The answer to most of the questions is either *yes* or *no* (or just the first letter of these words). Every answer is prompted; and should be answered by saying yes or no. In such cases where an other answer is expected the book of Friedman should be consulted. Note that in case you want to enter the input skeleton not from the terminal you must have the file allocated prior to calling the TGT-System.

11. THE CONTROL OF THE INPUT

Summarizing: if you want to run the TGT-System you have to define and enter a grammar, give a command for the sentence generator, and you have to deliver a skeleton to be expanded (Fig. 6). Originally all these three kinds of input were entered in sequence into the system on the same file as data.

It should be noted that the grammar is a part of the input data, which is entered and processed in each run. This homogenous input is then interpreted by the system as grammar or as input into the sentence generator according to the internal logics of the program. In order to achieve greater flexibility while testing a grammar, we separated the three logically different input into three logically different files. The input grammar, usually a text of several hundreds of lines,

[1] The files 8 and 9, containing system messages have been, however, dropped; they were of no interest to ordinary users.

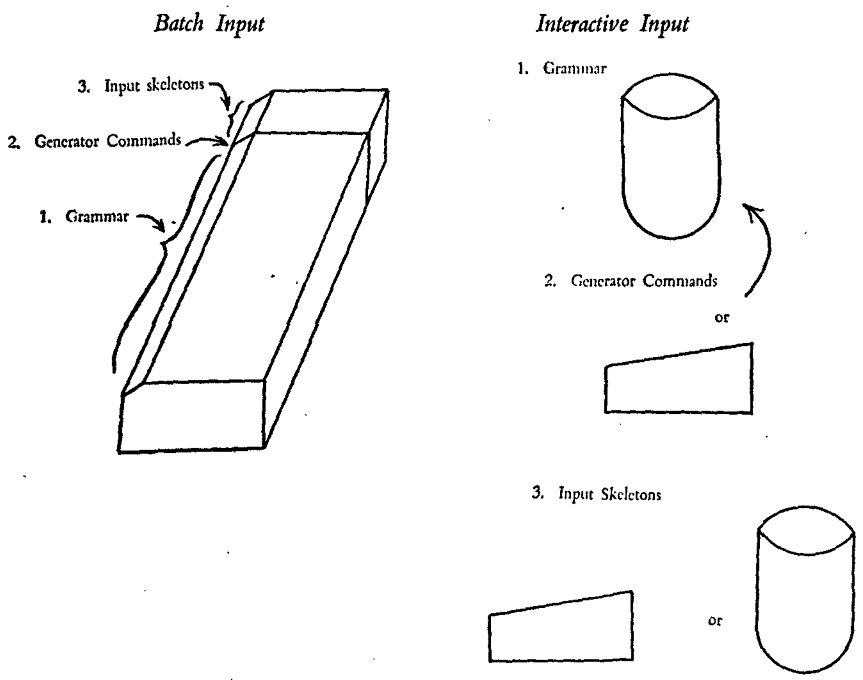

Fig. 6. *The Reorganization of Input*

is normally already stored on an external device and entered accordingly. The generator command (the $MAIN-card) may be attached to the grammar, if not, it is prompted and you may enter it from the terminal.

Similarly, you may predefine input skeletons to be tested and enter them just as you enter the grammar as a separate file. You have, however, the choice to enter skeletons directly from the terminal. In case of interest you may enter as many skeleton as you like. The random generator then provides for variation.

Technically, the separation of the three logically different kinds of input has been accomplished by introducing a file variable, which is set first to accept the grammar from a permanent data set and then changed over to the terminal or an other permanent input data set according to user specifications at session time.

12. THE TREATMENT OF THE ERROR MESSAGES

The same file variable technique is used to control the error messages. The error file is set either to the terminal or to the batch file alternatively. There would be no problem to assign the error messages permanently, yet an eventual change of the file requirements in terminal environment would mean a revision of several hundreds of error messages, while a file variable can be controlled by a single instruction.

There is a further problem to be faced and that is the reference point of the error message. In the original batch version the error message precedes the actual erroneous line in the grammar or inserted in the protocol at the appropriate point.

In the first case the interactive version does not display the original input grammar, and therefore a message that e.g. brackets are opened, but not closed or "special character expected", but not found, and the like are not very informative, since the user would be left alone to find the critical place in the grammar. Therefore the error messages during the processing of the input grammar are preceded by the actual line in which the error has occurred. The line numbering will help the linguist to localize the erroneous section in the input grammar.

If on the other hand the error occurs during sentence generation, the message will be inserted in the terminal protocol at the appropriate place.

13. THE CONTROL OF OUTPUT

Another crucial point is the control of the terminal output. You can have the following choices as regards extent of output:

1) You are not interested in any further details, you do not want to see the full input tree. In this case you still get: 1., the linear representation of the input, 2., the list of transformations which have been applied and 3., the output of the transformations, also in the linear form. This is the minimal amount of output (Fig. 7):

2) You wish to see the input tree into the transformational component, you answer to the question *PRINTOUT INPUT TREE?* by saying "yes". In this case you get also the full output tree of the

```
 ERROR MESSAGES WANTED?
y
 INPUT TREE FROM TERMINAL?
n
 PRINTOUT INPUT TREE?
n
 INPUT TO TRANSFORMATIONS:
      # Q NEG WER EIN PREIS ERHOEH WERD HAB KOENN WERD PRS #
 *
 TRANSFORMATIONS WHICH APPLY:
 *
 AUXELIM     ALESE            20       19
 AUXELIM     ALESE            22       19
 AUXELIM     ALESE            24       19
 AUXELIM     ALESE            26       19
 AUXELIM     ALESE            28       19
 KEIN        AFIDE     K               13
 KEIN        ERASE             0        6
 PASSIV      ALESE            12        8
 PASSIV      AFIDE            32        8      ADDED SUBTREE: 32
logoff in 15 min
 PASSIV      ALESE             8       17
 KNGRUM      ERASE             0       28
 VRISE       ARISE            26       11
 VRISE       ARISE            24       11
 VRISE       ARISE            22       11
 VRISE       ARISE            20       11
 VRISE       ARISE            17       11
 TOPIC       ERASE             0        3
 TOPIC       ALESE             8       12
 VERBUM3     ARISE            26        8
 GRMARK      ERASE             0        2
 GRMARK      ERASE             0       30
 INFPAR      ALADE     T               17
 INFPAR      SUBSE     WORDEN          21
 INFPAR      SUBSE     SEIN            23
 INFPAR      ALADE     EN              24
 MORPHY      SUBSE     WIRD            27
 INDART      SUBSE     EIN             14
 PROMOR      SUBSE     WEM             10
 PUNKT       ALADE            41        1      ADDED SUBTREE: 41
 PUNKT       ALADE            43        1      ADDED SUBTREE: 43
 OUTPUT GENERATED BY TRANSFORMATIONS:
 *
 *
      VON WEM WIRD KEIN PREIS ERHOEHT WORDEN SEIN KOENNEN
      QMARK PUNKT
1
```

Fig. 7.

```
 *
```

whole generation automatically. You get also the list of the transformations which apply displayed; to be more precise you get the list of tree operations (Fig. 8):

```
 ENTER RANDOM GENERATOR COMMANDS!
$main trin gen tran.
 INPUT TREE FROM TERMINAL?
yes
 ENTER RANDOM GENERATOR INPUT IN THE FORM OF TRIN!
       +      +         +      +      +
s

 PRINTOUT INPUT TREE?
yes
 PRINTOUT FEATURES?
no
 INPUT TO TRANSFORMATIONS:
   1 S         2 #
               3 NP       9 PRON     15 JEMAND
               4 VP       7 NP       10 N              14 HANS
                          8 V        16 SEH
               5 MOD     11 TNS      17 PRS
                         12 Q
                         13 NEG
     # JEMAND HANS SEH PRS Q NEG #
 *
 TRANSFORMATIONS WHICH APPLY:
 *
 NICHT         ALESE              18        9        ADDED SUBTREE: 18
 NICHT         ERASE               0       13
 KONGRUE5      ERASE               0       11
 VERBUM        ALESE               8        3
 VERBUM        ERASE               0       12
 GRLOE         ERASE               0        2
 GRLOE         SUBSE      FRGZ              6
 OUTPUT GENERATED BY TRANSFORMATIONS:
 *
 *
        1 S    8 V       16 SEH
               3 NP       9 PRON     15 JEMAND
               4 VP       7 NP       10 N              14 HANS
                         18 NEG      19 NICHT
              20 FRGZ
     SEH JEMAND HANS NICHT FRGZ
1
 ENTER GENERATOR INPUT AS TRIN    OR QUIT BY >/*>!
     +          +     +     +     +
/*
 TRIN .NO MORE INPUTS.
READY
```

Fig. 8.

3) You may want to see also the features associated with the nodes in the tree – then you respond to the next question of the system *PRINTOUT FEATURES* correspondingly – and you get the features displayed both of the input and the output tree. In addition you get also the list of transformations applying, now including also the feature operations (Fig. 9):

```
call new(tgt 250)
 *
 WELCOME TO THE INTERACTIVE VERSION OF
           FRIEDMAN|S TGT-SYSTEM!
 ERROR MESSAGES WANTED?
yes
 INPUT TREE FROM TERMINAL?
no
 PRINTOUT INPUT TREE?
yes
 PRINTOUT FEATURES?
yes
 INPUT TO TRANSFORMATIONS:
        1 S      2 #
                 3 NP      4 ART      5 D
                           6 N        7 STUDENT3
                 8 VP      9 NP      10 PRON        11 WAS
                          12 V       13 LES
                14 MOD    15 V       16 WERD
                          17 TNS     18 PRS
                          19 Q
                20 #
 NODE      4 ART
       | + ART + DEF |
 NODE      6 N
       | + N + MASC -SG + HUMAN + ANIM -PRPNM |
 NODE     10 PRON
       | + PRON + SG -HUMAN + ANIM + <S/<%_%Q%>> |
 NODE     12 V
       | + V + EN + STRK + <S/>#NP<(ART)* | + HUMAN |>%_%>> |
 NODE     15 V
       | + V + PASS |
 NODE     17 TNS
       | + TNS + PRS |
       #D STUDENTEN WAS LES WERD PRS Q#
 *
 TRANSFORMATIONS WHICH APPLY:
 *
 PASSIV      AFIDE       21       3      ADDED SUBTREE: 21
 PASSIV      MERGEF      12    + PART
 PASSIV      ALESE        9       3
 NOMIN       MERGEF       9    + NOM
 NOMIN       MERGEF      10    + NOM
 DATIV       MERGEF       3    + DAT
```

```
KONGRUE5    MOVEF          10       15       + SG
KONGRUE5    MOVEF          17       15       + PRS
KONGRUE5    ERASE           0       17
VERBUM      ALESE          15        9
VERBUM      ERASE           0       19
ERGFR       ARISE           9        2
INFPAR      ERASEF         12       SG
                                    PRS
INFPAR      AFIDE    GE             12
INFPAR      ALADE    N              12
MORPHY      SUBSE    WIRD           16
DEFART      SUBSE    DEN             5
DEFART      SUBSE    DEN             5
GRLOE       ERASE           0        2
GRLOE       SUBSE    FRGZ           20
OUTPUT GENERATED BY TRANSFORMATIONS:
*
*
       1 S     9 NP      10 PRON     11 WAS
              15 V       25 WIRD
               3 NP      21 PREP     22 VON
                          4 ART      26 DEN
                          6 N         7 STUDENT3
               8 VP      12 V        23 GE
                                     13 LES
                                     24 EN
              28 FRGZ
NODE      9 NP
       | + NOM |
NODE     10 PRON
       | + PRON + SG -HUMAN + ANIM + NOM + <S/<%-%Q%>> |
NODE     15 V
       | + V + SG + PRS + PASS |
NODE      3 NP
       | + DAT |
NODE     21 PREP
       | + DAT |
NODE      4 ART
       | + ART + DEF |
NODE      6 N
       | + N + MASC -SG + HUMAN + ANIM -PRPNM |
NODE     12 V
       | + V + EN + STRK + PART + <S/<≠NP<(ART)* | + HUMAN |>
       %-%>> |
       WAS WIRD VON DEN STUDENTEN GELESEN FRGZ
1
READY
```

Fig. 9.

4) You may be interested in even more details, for instance in some intermediate trees and you have inserted TRACE-cards in the control program of the grammar just as they are inserted in the original batch version. Now if you answer to the question *PRINTOUT INPUT TREE* by saying *ALL*, you will receive every intermediate tree as well, in addition to the input and output tree with features and feature operations. Otherwise the TRACE function returns just the terminal string of the derivation. Fig. 10 shows the general logics of the output control:

Printout input tree? Printout features?	No —	Yes No	Yes Yes	All —
Output produced:				
Terminal strings	+	+	+	+
Trees	—	+	+	+
Features	—	—	+	+
Tree operations	+	+	+	+
Feature operations	—	—	+	+
Intermediate Trees by TRACE	—	—	—	+

Fig. 10. *The Control of Interactive Output*

14. CLOSING REMARKS

A grammar developed directly with the aid of the TGT-System is practically never complete, it generates only a subset of the language in question. You may add, change, remove parts of the grammar and thus you can easily produce minor variants of the same grammar one of which may be preferable over the other. In fact this is the normal way to work with the system.

At the C.N.U.C.E.-installation there was a number of test-grammars (German, Italian, English and Spanish), offered to the participants to try how such testing looks like. The participants of the Conference were invited to look at the Grammar Tester as it works. In the Centro Nazionale Universitario di Calcolo Elettronico the Transformational Grammar Tester was running on a IBM System/360 Model 67 under CP-CMS-67.

REFERENCES

N. Chomsky, *Aspects of the Theory of Syntax*, Cambridge (Mass.), 1965.

J. Friedman, T. H. Bredt *Lexical Insertion in Transformational Grammar*, Palo Alto (Calif.), 1968.

J. Friedman, R. W. Doran, *A Formal Syntax for Transformational Grammar*, Palo Alto (Calif.), 1968.

J. Friedman, P. Myslensky, *Computer Experiments in Transformational Grammar: The UCLA English Grammar*, Ann Arbor (Mich.), 1970.

J. Friedman, *Application of a Computer System for Transformational Grammar*, Preprint No. 14, in *International Conference on Computational Linguistics*, Stockholm, 1969.

J. Friedman, *Directed Random Generations of Sentences*, in « Communications of the ACM », XII (1969) 1.

J. Friedman, *A Computer System for Transformational Grammar*, in « Communications of the ACM », XII (1969) 1.

J. Friedman, *Distribution and Use of the Computer System for Transformational Grammar*, Working paper in Computational Linguistics, M-27, The University of Michigan, 1973.

J. Friedman, *et al.*, *A Computer Model of Transformational Grammar* (*Mathematical Linguistics and Automatic Language Processing* Nr. 9), New York, London, Amsterdam, 1971.

Y. Ch. Morin, *Computer Experiments in Transformational Grammar: French I*, Ann Arbor (Mich.), (*Natural Language Studies* Nr. 3), 1969.

P. Rosenbaum, D. Lochak, *The IBM Core Grammar of English*, Yorktown Heights (New York), 1966.