Applications of a Computer System for Transformational Grammar*

by

Joyce Friedman
The University of Michigan
Ann Arbor, Michigan, U.S.A.

Writing a transformational grammar for even a fragment of a natural language is a task of a high order of complexity. Not only must the individual rules of the grammar perform as intended in isolation, but the rules must work correctly together in order to produce the desired results. The details of grammar-writing are likely to be regarded as secondary by the linguist, who is most concerned with what is in the language and how it is generated, and would generally prefer to pay less attention to formal and notational detail. It is thus natural to ask if a computer can be used to assist the linguist in developing a grammar. The model is formal; there are a large number of details to be worked out; the rules interact with one another in ways which may not be foreseen. Most of the errors which occur in writing grammars can be corrected if only they are brought to the attention of the linguist. Those which cannot be so corrected should be of even greater interest to the writer of the grammar.

---

*This research was supported in part by the United States Air Force Electronic Systems Division under Contract F19628-C-0035, and by the National Science Foundation under Grant GS-2271.

A computer system which attempts to provide this assistance to the writer of a grammar is now available at both the University of Michigan and Stanford University.[1] The system is written in Fortran IV and runs on the IBM 360/67 computer.

The linguistic model embodied by the system is the theory of transformational grammar, roughly as described by Chomsky in Aspects of the Theory of Syntax.[2] The programs represent the linguistic metatheory. Grammars are accepted by the programs as data. The program handles all three components of a transformational grammar: phrase structure, lexicon, and transformations. It carries out the full process of sentence generation, including phrase structure generation, lexical insertion, and transformation.

The technical details of the particular model of transformational grammar have been described elsewhere [3]. This presentation will emphasize the ways in which the programs can be used, and will describe experiences in using them both in grammar writing and in teaching.

An example of a grammar

The notation for grammars and the use of the programs will not be described formally here, but will be illustrated by an extended example. The example consists of a small grammar and a sample derivation. Each part will be presented twice, first as initially

---

[1] This system was designed and programmed by the author, with T. H. Bredt, R. W. Doran, T. S. Martner, and B.W. Pollack.

prepared by linguists at the University of Montreal [1], and then as redone in the computer system. The grammar has been greatly reduced by selecting only those transformations which were used in the derivation of the sample sentence selected.

In Figures 1 and 2 the phrase-structure rules are given in parallel, first as written by the linguists, secondly as prepared for input to the computer system. The computer form can be seen to be a linearization of the usual form, with both parentheses and curly brackets represented by parentheses. No ambiguity arises from this since the presence of a comma distinguishes the choices from the options. The only other differences are minor: the symbol "Δ" has been replaced by "DELTA", the sentence symbol "P" has been translated into English "S", and accents have been omitted. None of these changes is of any consequence.

Figure 3 is a listing of a partial lexicon. This component is present only implicitly in the original French grammar, where the complex symbols are indicated in the base trees. The lexicon specifies first the features which are used in the grammar. The list of category features also determines the order in which lexical insertion takes place. The inherent features include some which are included in complex symbols in the lexicon, and others which are added only by transformations. Contextual features are defined in the lexicon by giving a structural analysis of the context in which the item can appear. The location in which the word may be inserted is indicated by an underline symbol. Thus, common nouns are marked +NCOM, which means that they can occur only following a determiner (DET) in a

Figure 1

Phrase Structure Rules, from [1]

P → # (PRE) SN PRED #

PRE → (INT) (NEG)

NEG → ne pas

PRED → { SV (ADV / INST) / SA }

ADV / INST → par { SN / Δ }

SV → V (COMPL)

SA → COP ADJ (COMPL)

COP → est

SN → { (SN) P / (DET) N }

COMPL → { SN / P } (SN)

DET → ({ DEF / quel }) (CARD)

DEF → { ANAPH / DEM }

ANAPH → { ce ({ ci / là }) / le }

DEM → ce ({ ci / là })

CARD → { SING / PLUR }

```
     "MONTREAL FRENCH"
PHRASESTRUCTURE
S = # (PRE) SN PRED # .
PRE = (INT) (NEG).
NEG = NE PAS.
PRED = (SV (ADVINS),SA).
ADVINS = PAR (SN,DELTA).
SV = V (COMPL).
SA = COP ADJ (COMPL).
COP = EST.
SN = ((SN) S, (DET) N).
COMPL = (SN,S) (SN).
DET = ((DEF,QUEL))(CARD).
DEF = (ANAPH,DEM).
ANAPH = (CE ((CI,LA)),LE).
DEM = CE ((CI,LA)).
CARD = (SING,PLUR).
SING = UN.
PLUR = (PROCARD,QUELQUES,DEUX,TROIS).
PROCARD = NOMBRE DE.
    $ENDPSG
```

Figure 2

Phrase Structure Rules

Figure 3

Lexicon

```
LEXICON
  CATEGORY V COP N ADJ .
  INHERENT PERS TWOPERS FEM PLUR
           HUM MOI NOMIN TOI ASN
               INF DASHQUE DEINF A SUBJ PRENOM
               PRON FUTUR PROG PRET PASSIF E .
  CONTEXTUAL
    NCOM = <SN<DET _>>,
    IDSUJ = <S/<# (PRE) ISN PRED<SV<_ S/<% 2SN V % >>>#>,
            WHERE 1 EQ 2>,
    IDOBJ = <SV/<_ COMPL<ISN S/<% 2SN V %>>>, WHERE 1 EQ 2> .
ENTRIES
  TRUDEAU JEAN ALCESTE
  DEGAULLE \+N -PERS -TWOPERS -FEM -PLUR -NCOM\,
  CELIMENE \+N -PERS -TWOPERS +FEM -PLUR -NCOM\,
  LIVRE \+N -PERS -TWOPERS +FEM -PLUR +NCOM\,
  FILLE \+N -PERS -TWOPERS +HUM +FEM -PLUR +NCOM\,
  PROFESSEUR \+N -PERS -TWOPERS -FEM -PLUR +NCOM\,
  MOI\+N +MOI +PERS -TWOPERS -FEM -PLUR -NCOM\,
  TOI\+N +TOI +PERS +TWOPERS -FEM -PLUR -NCOM\,
  LEVE SOURIR CROIT DANSE
  BERNE \ +V \,
  PRIE\+V +IDOBJ +DEINF\,
  OSE\+V +IDSUJ\,
  PARLE
  PLAIT\+V +ASN\,
  ACCEPTE \+V +FUTUR -PROG -PRET\,
  REGARDE RENCONTRE\+V -PRET -FUTUR -PROG\,
  NECESSAIRE\+ADJ +SUBJ\,
  JOLI\+ADJ +PRENOM\,
  EST\+COP -PRET -FUTUR -PROG\
  . $ENDLEX
```

noun phrase (SN).

After the preliminary definitions, the lexicon contains a set of lexical entries. In a computer derivation of a sentence, lexical items will be selected at random from those of the appropriate category which match inherent features already in the tree and have contextual features satisfied by the tree.

Figures 4 and 5 are presentations of the transformational component. In the computer version a transformation consists of three parts, identification, structural description (SD), and structural change (SC). The identification contains the number and name of the transformation, plus information to be used in determining when it is to be invoked. This includes a group number repetition parameters, keywords, etc. The structural description is similar to the linguistic form, but allows also subanalysis to any depth. Representation of the structural change by a sequence of elementary operations removes any possible ambiguity from the statement. In addition to adjunctions and substitutions, there are also elementary operations which alter complex symbols. \+PASSIF\ MERGEF 4 adds a new feature specification to the complex symbol of term 4. *FEM *PERS\ MOVEF 4 7 will change those two features of term 7 so that they are the same as for term 4.

It may be noted that the transformation LOWESTS and the control program of Figure 5 have no correspondents in Figure 4. They are needed because the program requires that the traffic rules be given explicitly as part of a grammar. LOWESTS selects the lowest sentence

[T7] POST -SUJ OBL

#	(PRE)	SN	V	SN	(SN)	par	Δ	#	
1	2	3	4	5	6	7	8	9	=⟩
1	2	8	4	5	6	7	3	9	

[T9] ANTEP-OBL. OBL

#	(PRE)	Δ	V	SN	(SN)	par+SN	#	
1	2	3	4	5	6	7	8	=⟩
1	2	5	4	⟨+passif	∅	6	7	8

[T13] AC-PRED OBL

#	(PRE)	[(DET)	[α fem, α pers, α 2 pers, α plur]N	(P)]SN	(COP)	{V, ADJ}
1	2	3	4	5	6	7
					Z	#
					8	9 =⟩
1	2	3	4	5	6	7⟨ α fem, α pers, α 2 pers, α plur
					8	9

COND: 7 ≠ α fem, α pers, α 2 pers, α plur

[T33] ELLIPSE ## OBL

#	X	#	
1	2	3	⇒
∅	2	∅	

Figure 4

Transformations, from [1]

[T51] M-PASS. OBL

X ⟨+passif⟩ Y
V

1 2 3 ⇒

1+est 2⟨+é ⟨é+3

COND: 2≠+é

[T52] TR-TRAITS-PASS OBL

X est [+passif, ⟨progr, ⟨futur, ⟨preteri, ⟨pers, ⟨2 pers, ⟨inf, ⟨fem, ⟨plur] v
V

1 2 3 4 ⇒

1 2 ⟨progr 3 4
⟨futur
⟨preterit
⟨pers
⟨2 pers
⟨fem
⟨plur
⟨inf

COND: 2⟨progr
futur
preterit

Figure 4 (Continued)

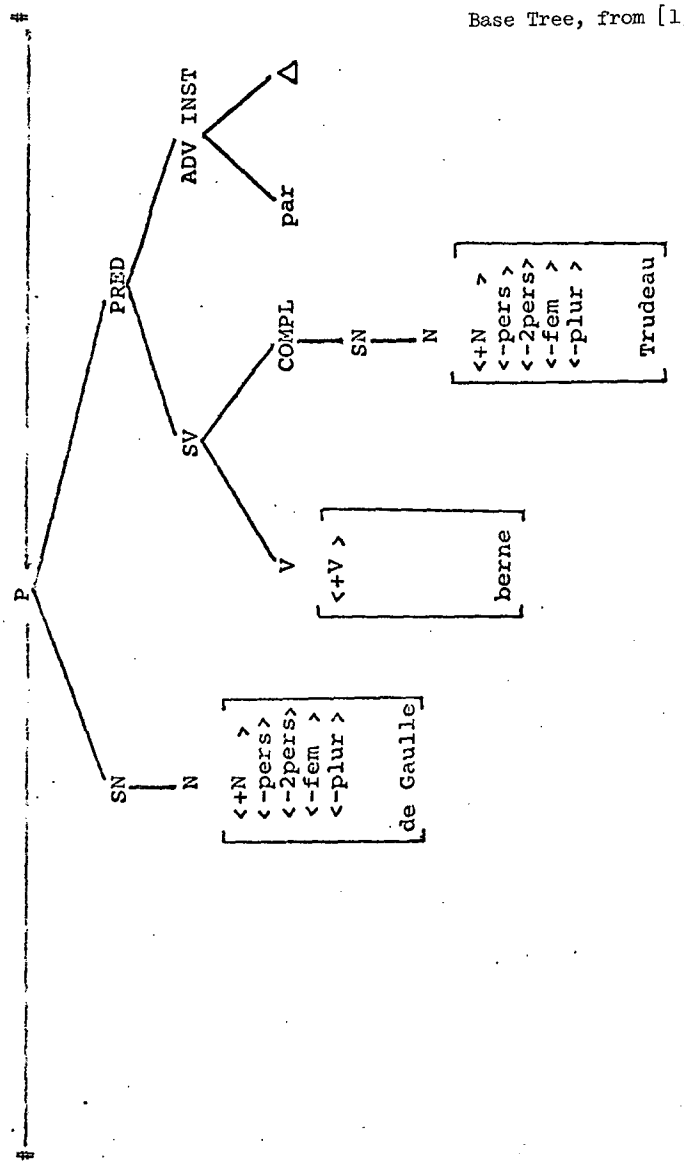
Figure 5

Transformations

```
TRANSFORMATIONS
TRANS 0 LOWESTS III (#).
  SD 1 S "/<Z S<# Z #> Z %>."WHERE 1 DOM #.
    "CYCLIC TRANSFORMATIONS"
TRANS 7 POSTSUJ I (PAR DELTA).
  SD # (PRE) 3 SN V SN (SN) PAR 8 DELTA #.
  SC 8 ADRIS 3, 3 SUBSE 8.
TRANS 9 ANTEPOSL (DELTA PAR).
  SD # (PRE) 3 DELTA 4 V 5 SN (SN) PAR SN #.
  SC 5 SUBSE 3, \+PASSIF\ MERGEF 4.
TRANS 13 ACPRED AACC.
  SD # (PRE) SN/<(DET) 4N
    (S)> (COP) 7 (V,ADJ) Z #.
  SC \*FEM *PERS *TWOPERS *PLUR\ MOVEF 4 7.
TRANS 33 ELLIPSE.
  SD 1 # Z 3 #.
  SC "ERASE 1, ERASE 3.
    "POSTCYCLIC TRANSFORMATIONS"
TRANS 51 MPASS II AACC.
  SD Z 2 V \+PASSIF\ Z.
  SC EST ALESE 2, \+E\ MERGEF 2, E ARISE 2.
TRANS 52 TRTRPA AACC (EST V).
  SD Z 2EST 3V\+PASSIF\ Z.
  SC \*PROG *FUTUR *PRET *PERS *TWOPERS *FEM *PLUR *INF\
    MOVEF 3 2.
    "CONTROL PROGRAM"
CP  IN LOWESTS(1) DO <I>; TREE; II.  $ENDTRA
```

Figure 6

Base Tree, from [1]



Trudeau est berné par de Gaulle

which contains boundaries. The control program specifies that the cyclic transformations are to be carried out for this lowest sentence. After a cycle the boundaries are erased and the next highest sentence becomes lowest. The postcyclic transformations will then be carried out.

A particular tree, created as an example in [1], is presented in Figures 6 and 7. Figure 7 contains two alternative versions, a fixed-field format and a bracketted free-field form. Either of these is acceptable to the programs. The sentence at the top of the figure is merely a title; it will not be processed by the program. The lexical items "Trudeau", "deGaulle", and "berne" have not been included, although they could have been. If these items had been entered in the tree, the lexical insertion process would merely have added the appropriate complex symbols for them.

Figure 8 gives the derivation as presented in [1]. Figure 9 is the final part of the listing of the computer output.

The use of the programs

The system was designed to be used by a linguist who is in the process of writing a transformational grammar. As lexical items or transformations are added they can be tested in the context of all previous rules and their effect can be examined.

The easiest errors to detect and repair in a grammar are syntactic errors. As a grammar is read in by the program a check is made for formal correctness. For each error a comment is produced which attempts to explain what is wrong. The program then continues

Figure 7

Alternative forms of Base Tree

TRUDEAU EST BERNE PAR DEGAULLE.

S
- #
- SN — N
- PRED — SV — V
  - COMPL — SN — N
  - ADVINSPAR
  - DELTA

#

S<# SN <N> PRED <SV <V> COMPL <SN <N>>>ADVINS <PAR DELTA>> #>.

Figure 8

Derivation, from [1]

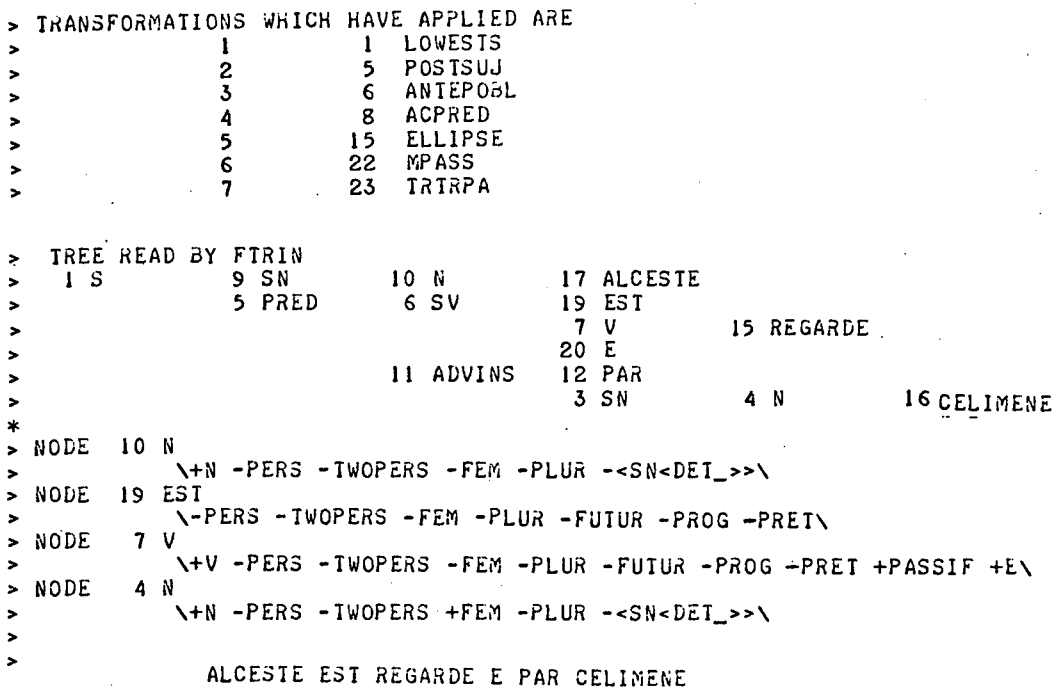


Figure 9
Computer Derivation

to read in the rest of the grammar, recovering as best it can from the error. In most cases a single error will cause a small part of the grammar to be read badly, but the rest of the grammar will be read in and used in whatever tests were requested. An effort was made to make the error comments as clear and explicit as possible, and to make the program continue despite input errors.

Deeper errors arise when a grammar is syntactically correct, but does not correctly describe the language of which it purports to be a grammar. These errors of intent cannot be detected directly by the program, since it has no standard of comparison. The program attempts to provide enough feedback to the linguist so that he will be able to detect and investigate the errors.

The information produced by the program consists of derivations which may be partially controlled by the user. Since random derivations have been found to be of relatively little interest, the system allows the user to control the sentences to be generated so that they are relevant to his current problem. (The device used for this purpose has been described in [4].) It is only in the sense of providing feedback to the user that the system can be called a "grammar tester"; it does not directly seek out errors in a grammar, nor does it evaluate the grammar.

For a standard run of the system the inputs are a grammar, a $MAIN card, and some trees. The grammar consists of one or more of phrase structure, lexicon, and transformations. The $MAIN card is a specification of the type of run to be made. The system must be

told (1) what type of input trees to expect:

TRIN, for fixed-field tree

FTRIN, for free-field bracketted tree

(2) whether to generate a tree around a skeletal input or whether it is only necessary to insert lexical items:

GEN, to generate a tree and insert lexical items

LEX, to insert lexical items

and (3) whether or not transformations are to be applied:

TRAN, if transformations are to be invoked.

The general form of the $MAIN card can be represented as

$$\$MAIN \quad \left\{\begin{matrix}TRIN\\FTRIN\end{matrix}\right\} \quad \left((n) \quad \left\{\begin{matrix}GEN\\LEX\end{matrix}\right\} \right) \quad (TRAN) \; .$$

The integer n specifies the number of time each input tree is to be used.

An an example,

$MAIN TRIN GEN TRAN .

specifies a run in which a skeletal tree is read, a full tree is generated including lexical items, and the transformations are applied.

The specification

$MAIN TRIN 5 LEX TRAN .

might be used in testing a lexicon and transformations against a fixed base tree. The tree will be read and five cases of lexical

insertion plus transformation will be carried out.

```
$MAIN   TRIN   4   LEX .
```

would do four examples of lexical insertion for each input.

After the process is completed for one input, another input is read and the cycle repeats. A run terminates when there are no more inputs.

Computer experiments in transformational grammar

The system has been in use since February 1968, although not fully complete at that time. The first experiments were carried out by the designers of the system, using grammars based on material in the linguistic literature. This was done to provide test material for the programs, but, more importantly, to help ensure that the notational conventions would be adequate. A fragment of grammar from Chomsky's Aspects was used to test ideas and programs for lexical insertion. The IBM Core Grammar of Rosenbaum and Lochak [6] was used in developing and testing the transformational component. Both of these projects led to valuable teaching materials, as we shall discuss later.

Aspects and Core provided us with separate examples of lexicon and transformations. There was at first no single source which contained both. A relatively formal grammar was needed, even though a final translation into the notation of the system would still of course be necessary. Elizabeth Closs Traugott's Deep and surface structure in Alfredian Prose [7] appeared at about that time and was the first grammar which was formalized in the notation after the

fact. Considerable effort had gone into designing the notation; we were anxious to see if it would now seem natural for a grammar which was new to us. Alfred was thus the first real test for the system. As it turned out there were a few difficulties which arose because the notation had not been explained clearly enough, but the results of the run were also revealing about the grammar.

One general effect which was noticed in these first few cases had continued to be striking: the need for complete precision in the statement of a grammar forces the linguist to consider problems which are important, but of which he would otherwise be unaware.

Also during the spring of 1969 Barbara Hall Partee made two sets of runs with preliminary versions of a grammar of English being developed by the U.C.L.A. Air Force English Syntax Project. This grammar presented another kind of challenge to the system, because it was not based directly on the Aspects model, but incorporated some recent ideas of Fillmore. As before, these runs assisted in cleaning up the programs but were also of interest to the linguist. The major advantages from the linguistic point of view seem to have been, first, that the notational system of the computer model provided a framework in which grammars could be stated, and second, that the computer runs made it easier to detect certain errors in the grammars. In the main, these errors were not particularly subtle, and could have been caught by working over the grammar carefully.

The program was also used by L. Klevansky, who wrote a grammar of Swahili for the dual purposes of testing the programs and learning the language.

These early experiments are described in a report [5] which gives the grammars as well as a detailed discussion of the results of the computer runs.

The form of the French grammar used in the extended example above is based on the form of the Core grammar; it was therefore easily translated into the notation of the system. Shortly after the grammar was received, a large part of it was running on the computer. Minor errors in the grammar have been found and corrected; it will now be available to students as another example of a transformational grammar.

The next experiment planned using the system is a project proposed by Susumu Nagara and Donald Smith at the University of Michigan, who plan to use the system to aid in writing a grammar of Japanese.

Modifications to grammars based on computer runs

In almost all cases the grammars used with the system have been sufficiently complete for at least informal distribution. The programs were really designed to make it easier to write grammars, not to test completed grammars. Nonetheless, on the basis of computer runs, certain types of changes have been found to be needed in the grammars. The comments which follow are based on all the grammars; they do not all apply to any one of them.

Trivial corrections

The most common errors are typographical errors in transcription of the grammar. These are not errors in the grammar itself; having

to deal with them is one of the prices of using the computer. In general, these can be caught with relative ease.

More than one grammar has had simple errors with respect to repetition of a transformation. Number agreement transformations are written so that they produce CAT S S S ... where CAT S is wanted. (The grammar as written calls for an infinite sequence of S's to be added. The program, more cautious, adds ten S's, then complains and goes on to the next transformation.)

Transformations are often stated so that they fail to apply in all cases where it is intended they apply. For example, the structural description of PASSIVE as

SD # (PRE) 3NP AUX 5V (PREP) 7NP % PREP 10P % # ,

WHERE 3 EQ 7.

fails to take into account some additional parts of the VP. The correction to

SD # (PRE) 3NP AUX (HAVE EN)(BE ING) 5V (PREP) 7NP % PREP 10P % #, WHERE 3 EQ 7.

will allow PASSIVE to work in the additional cases. Similarly, a NOMINAL-AGREEMENT transformation which marks subjects as +NOMIN must apply not only to pronouns which precede verbs but also to those which precede copulas. Thus the structural description

SD # % 3(PRON,REL) V % # .

must be replaced by

SD # % 3(PRON,REL) (V,COP) % # .

Interrelatedness of transformations

A slightly more interesting set of problems found in the computer runs are those which arise through the interrelatedness of two or more transformations. For example, in one of the grammars there were both WH-questions and TAG-questions. It was found that the TAG transformations was (optionally) applicable to any question, so that for example

TOM HAS PREFER EN WHAT GIRL HAS TOM NOT

was produced. This error was easily repaired once it was detected.

On the other hand, a similar problem which was not easily fixed arose with another transformation which was marked optional. Testing showed that for certain base trees the result was bad if the transformation did not apply; however, when the transformation was temporarily changed to obligatory, the grammar then failed to produce some intended sentences. The proper correction to the grammar would have required specification of the contexts in which the transformation was obligatory.

Incompleteness of grammars

Formal grammars so far have each attempted to describe some subset of a language. In computer testing many problems outside the scope of the grammar are evident. If, for example, a grammar does not treat prepositions seriously, then once this becomes apparent, the computer runs need to be designed to avoid prepositions.

Deep structure problems

Two of the grammars which have been studied suffer problems

with the WH-morpheme when it occurs in non-sentences and not as a relative marker. Thus, for example, sentences such as

WHAT BLAME MAY NT BE BE ING

and

WHICH THING MUST HAVE BE EN APPROVE ING OF
WHAT TABLE

are in fact even worse than they appear, because they are not questions. Although this problem has no simple solution in the current framework, the inputs to the program can be controlled to avoid generating sentences of this form.

Inadequacies in the linguistic model

An interesting change to the system was suggested by the attempt to formalize the Core grammar. In both the WH-attraction and the Question-transformations the structural description contains a two-part choice between a PREP NP pair and simply an NP. This is of the form:

% (PREP NP, NP) %

where % is a variable. Any structure which satisfies the first part of the choice will also satisfy the second, and any analysis algorithm must have some order of search which will either always select PREP NP or always select NP only. But the intent is that there should be a genuine choice, so that the grammar produces both

ABOUT WHAT DID JOHN SPEAK

and

The solution which was found for the problem was to add an additional value (AAC) for the repetition parameter for a transformation.

If a transformation is marked AAC, all possible analyses will be found, but only one of them, selected at random, will be used as the basis for structural change. This seemed the appropriate way to solve the problem for the Core grammar, and it turned out also to solve a slightly different repetition problem in the grammar of Alfredian prose. Notice that this is really an observation about the form of grammars, rather than about a particular grammar. Yet it arose by consideration of particular examples.

Surface structure

The surface structure associated with a sentence derivation is much easier to study if it can be produced automatically. In several cases it has been apparent from the information provided by the computer runs that revisions in the grammar were needed if the surface structure is to be at all reasonable. This is a case where the computer runs are certainly not necessary, but where they reduce the tediousness of studying the problem.

In summary, it seems to me that main value in computer testing of a completed grammar is that the need for a precise statement brings to the consideration of the linguist problems which are otherwise below the surface. These problems may be in the grammar itself or they may be in the linguistic model itself. For a grammar in process of being written the greatest advantage is in allowing rules

to be checked as they are added, and in bringing out the interaction between rules.

Instructional use of the system

The system has now been used by Sziliard Szabo in teaching general linguistics at the University of San Francisco, by Michael O'Malley in a course in natural language structure at the University of Michigan, and by the author in courses in computational linguistics at Stanford and Michigan.

The method of use is to make available to the students a file of one or more grammars to be used as examples and as bases for modifications. The fragments from Aspects and the IBM Core grammar have been most useful, although small grammar written for this purpose have also been used. The students are then asked to make modifications and additions to the grammars.

For graduate students, a reasonable exercise for a term paper is to read a current journal article on transformational grammar, and then show how the results can be incorporated into the basic grammar, or show why they cannot be. The papers chosen by the students have generally been ones in which transformations are actually given. This project has been very successful as an introduction to transformational grammar for computer science students.

Other students have chosen simply to use the computer to obtain fully developed examples of derivations illustrating aspects of grammar in which they are interested.

These experiences have confirmed our belief that specific

examples presented by the computer, and the feedback provided when a student modifies a grammar, are valuable in enabling the student to understand the notion of transformational grammar.

References

[1] Colmerauer, C., M. Courval, M. Poirier, and Antonio A. M. Querido. Grammaire- I, Description syntaxique d'un sous-ensemble du francais, Universite de Montreal, (March, 1969).

[2] Chomsky, N. Aspects of the Theory of Syntax. M.I.T. Press, Cambridge, Massachusetts (1965).

[3] Friedman, Joyce. A computer system for transformational grammar. Comm. ACM (to appear).

[4] Friedman, Joyce. Directed random generation of sentences. Comm. ACM, 12, pp. 40-46.

[5] Friedman, Joyce. (Ed.) Computer Experiments in Transformational Grammar, CS-108, Computer Science Dept., Stanford University, (August, 1968).

[6] Rosenbaum, R., and Lochak, D. The IBM core grammar of English. In Lieberman, D. (Ed.) Specification and utilization of a transformational grammar. AFCRL-66-270 (1966).

[7] Traugott, Elizabeth C. Deep and surface structure in Alfredian prose. Mimeographed. PEGS Paper #14 (August, 1967).