INTRODUCTION

Chomsky defines a generative grammar as one that "attempts to characterize in the most neutral possible terms the knowledge of the language that provides the basis for actual use of language by a speaker-hearer." It is "a system of rules that in some explicit and well-defined way assigns structural descriptions to sentences." The syntactic component of such a grammar specifies the well-formed strings of formative elements in the language and assigns structures to them.

Transformational grammars are built on the concept of the logical separation of two types of structure, deep and the surface structure. Accordingly, there are two systems of rules in the syntactic component of a transformational grammar: phrase structure rules generate deep structure, taking the form of a labeled bracketing or tree; transformational rules map trees to other trees and determine the ultimate surface structure of a sentence. The deep structure is operated on by the semantic component of the grammar to determine meaning; the surface structure is interpreted by the phonological component.

The emphasis on explicitness is a distinct advantage of generative grammars. However, it imposes a great burden on the linguist. Given a deep structure, the determination of the applicable rules in the derivation of the surface structure of a particular sentence is an extremely tedious and time-consuming task, difficult to perform with accuracy. For example, verifying by hand the correctness of the derivation of typical sentences in the IBM Core Grammar took us on the average two hours per sentence. And, as the grammar becomes large (as the linguist attempts to account for more phenomena in the language he is describing) it becomes more difficult to provide for all of the possible interrelations of the rules.

TGT: Transformational grammar tester*

by DAVE L. LONDE and WILLIAM J. SCHOENE
System Development Corporation
Santa Monica, California

TGT is a system of computer programs that can provide assistance to linguists in building and validating transformational grammars. The name TGT, "Transformational Grammar Tester," connotes a more ambitious project than was intended or undertaken. Not enough is known about some of the components (e.g., the semantic component) to warrant testing. And although there has been a considerable amount of work done on phonological rules, TGT has been addressed almost exclusively to the debugging and maintenance of the syntactic component.

For more than a decade, computers have been used experimentally to process segments of natural language text according to syntactic rules. The experiments have included both synthetic syntactic processing (generating sentences, together with their structural descriptions according to some previously specified grammar) and analytic syntactic processing (recognizing for a given sequence of formatives considered to be a sentence, each structural description, if any, that the grammar can assign to the sequence).

Unlike TGT, projects engaged in syntactic processing of natural language have seldom had as their sole objective the refining and extending of the grammar initially specified; often this has not been the primary objective.

Nevertheless, important contributions to grammar validation have been made by Petrick, although the technique used, that of recognition, is appropriate only for grammars that are relatively complete.

The system for syntactic analysis at The MITRE Corporation, described in Zwicky, et al. [1965], and modified as described in Gross [1967], is currently being used to perform some of the same functions included in TGT.

An off-line system for compiling, updating and testing the IBM Core Grammar was written in LISP by Blair [1966] and is currently in use at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

Further work on an off-line computer system for grammar testing by synthesis and by constrained...
"random" generation is being currently performed by Dr. Joyce Friedman at the Department of Computer Sciences, Stanford University, with whom we have.probably exchanged information.

TGT was designed with the goal of being as general as possible while still accommodating its immediate and primary user, the Air Force UCLA English Syntax Project (AFESP)* which is currently reviewing and attempting to integrate the work that has been done on transformational analyses of English. While it was impossible for AFESP at the outset to make definite decisions regarding its needs, since principal decisions on matters such as rule conventions could only be made after the data were gathered and analyzed, it seemed apparent to us that its needs could better be served by a new system, that (among other things) would be user-oriented and interactive, would handle subcategorization and selectional features, and would facilitate extension and modification. (However, for comments on factors limiting these goals, see the Discussion section at the end of this paper.)

System overview

In constructing a grammar with TGT, the linguist will be asking the same questions and employing many of the same procedures that he would have used were the tester not available. With the tester, many of those amenable to programming can now be done by the computer. The current version of TGT is programmed in the JTS version of the JOVIAL language and occupies 38,800 words of 48 bit memory, operating under the time-sharing system for the AN/FS Q-32 computer. An ITT model 33 teletype is used as the standard input/output device. A CRT with a capability of displaying 680 characters on a 1024 × 1024 matrix is an optional device for output. CRT output is faster and more readable; but its use, because of hardware requirements, dictates proximity to the Q-32. Teletype output is not subject to such distance limitations and is frequently used, even by those who have a CRT, as a means of obtaining hard copy output.

The most important tasks to be performed by TGT center around its ability to execute transformations. Using TGT the linguist can determine the applicability of transformations, execute them and display their results. Many ancillary functions are provided for specification and manipulation of rules and test structures. Combinations of these functions are employed to aid the user in determining the implications of changes in the rule set. For example, the linguist can save entire derivations of sentences that he considers correct. He may then insert a new rule, or change or delete an existing one, and then have the computer apply the rule set to the base phrase markers of the saved sentences. Changes in the derivations can then be immediately determined.

Capabilities*

A. Displaying and saving trees

Most TGT operations apply to the most recent or "current" tree in memory. Trees may be created by the primitive DISPLAY (abbreviated as D). The tree in Figure 1 could have been initially input by typing:

\[
D \, \text{(Si(# PRE(Q) NP(APR(ART(WH DEF)))))}
\]

\[
D \, \text{N(T(eN(APR(THING)))))}
\]

\[
D \, \text{AUX(T(PAST)))}
\]

\[
D \, \text{VP(V)
}\]

\[
D \, \text{DISAPPEAR(<+V>)
}\]

The components of the tree (categories, features, and complex symbols) are automatically numbered.

The "current" tree may be named and saved if desired by typing SAVE or S followed by an alphanumeric string. If there is already a tree in the system with that name, TGT informs the user, who is then given the option of changing the name of the new tree or replacing the old tree. The most recently used trees are saved in core memory. The others are stored on disc. Saved trees may be restored at any time by using the DISPLAY primitive with the name of the saved tree. This then qualifies as the most recently displayed or "current" tree.

Trees are initially left-justified on the scope. If the entire tree cannot be displayed, a message is output indicating the numbers of the top-most nodes of the major subtrees that are not displayed. Normally a right-justify command (RJ) is sufficient to display the missing subtree(s). However, an additional command, CENTER, is also useful in these cases. Accompanied by a number, it causes the subtree with that number to be displayed exclusively. CENTER also enables the user to position the current tree anywhere on the scope by specifying the direction in which the tree is to be moved (left, right, up, or down) and the number of units it is to be moved.

The tree in Figure 1 may be listed on the teletype by typing the command LIST, resulting in the TTY printout shown in Figure 2.

It is seldom necessary to input an entire tree parenthetically as in (1) above. Most desired trees can be produced by altering a few basic trees with the following primitives:

\[
D \, \text{Si(# PRE(Q) NP(APR(ART(WH DEF)))))}
\]

\[
D \, \text{N(T(eN(APR(THING)))))}
\]

\[
D \, \text{AUX(T(PAST)))}
\]

\[
D \, \text{VP(V)
}\]

\[
D \, \text{DISAPPEAR(<+V>)
}\]

* The official title of the project is, Integration of Transformational Studies on English Syntax, Principal Investigator - Robert P. Stockwell, Co-Principal Investigators - Paul M. Schachter and Barbara Hall Partee.
**Input**

<table>
<thead>
<tr>
<th>No.</th>
<th>Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ERASE N1</td>
<td>Erases subtree whose number is N1</td>
</tr>
<tr>
<td></td>
<td>E N1</td>
<td>(that is, N1 and all that it dominates).</td>
</tr>
<tr>
<td>3</td>
<td>AD A N1</td>
<td>Add A as a daughter to N1</td>
</tr>
<tr>
<td>4</td>
<td>ALS A N1</td>
<td>Add A as a left sister to N1</td>
</tr>
<tr>
<td>5</td>
<td>ARS A N1</td>
<td>Add A as a right sister to N1</td>
</tr>
<tr>
<td>6</td>
<td>SUB A N1</td>
<td>Substitute A for N1</td>
</tr>
<tr>
<td>7</td>
<td>EAD N1 N2</td>
<td>Erase N1 from its original position and add it as a daughter to N2.</td>
</tr>
<tr>
<td>8</td>
<td>EALS N1 N2</td>
<td>Erase N1 and add it as a left sister to N2.</td>
</tr>
<tr>
<td>9</td>
<td>EARS N1 N2</td>
<td>Erase N1 and add it as a right sister to N2.</td>
</tr>
<tr>
<td>10</td>
<td>ESUB N1 N2</td>
<td>Erase N1 and substitute it for N2</td>
</tr>
</tbody>
</table>

*Where N1 and N2 represent the numbers of nodes of the current tree and A may be one of the following: the number of a node in the current tree, the name of a saved tree, or a structure (subtree, feature or complex symbol) input in the parenthetic notation of (1) above.*

**B. Phrase structure rules**

Phrase structure rules are not essential to the operation of TGT, but their presence permits a legality check on the trees. The command VERIFY, input from the teletype, will return the message YES or NO indicating that the current tree could or could not have been produced by the phrase structure rules. In the present system, the complex symbols (e.g., those symbols enclosed in straight line brackets in Figure 1 and numbered 11 and 22), are ignored by the VERIFY subroutine.

As in Chomsky [1965] phrase structure rules are expected to be context free. They may be entered into the system by typing the identifier PS followed by the symbol to be expanded, an arrow (minus sign and greater-than sign) and the string indicating the legal expansion.

\[
PS \text{ VP} \rightarrow AUX (MV (NP, NIL), COP (NP, ADJ, NIL))
\]  

Example (11) above indicated that VP may be expanded as any one of the following strings:

- AUX MV
- AUX MV NP
- AUX COP
- AUX COP NP
- AUX COP ADJ

Parentheses indicate that of the strings within that are separated by commas, just one is to be chosen. A NIL within the parentheses indicates that none need be chosen.

\[
PS \text{ S} \rightarrow (S \text{ CONJ S} (\text{CONJ S, NIL}), \text{NP VP})
\]  

---

*From the collection of the Computer History Museum (www.computerhistory.org)*
Example (12) indicates that $S$ may be rewritten as

\[
NP \quad VP \\
S \quad CONJ \ S \\
S \quad CONJ \ S \quad CONJ \ S \\
S \quad CONJ \ S \quad CONJ \ S \quad CONJ \ S \\
\]

etc.

Thus, an asterisk following parentheses indicates reapplicability of the set within.

The expansion of a symbol can be changed merely by reinputting the symbol and its new expansion. TGT saves only the most recent.

C. Transformations*

Transformations are input via the teletype using the operator RULE. The canonical form is:

RULE rule-name structural-description $=>$ structure-change \ (conditions

RULE HYPOTHETI $X \ A \ *1B \ *2(C, D, E, *3F)$

$X \ G \ *4B \ X =>$

$1 = 2 + 1; 2 = \emptyset; 3 = \emptyset. 4 \ EQ \ 1.$ \hspace{1cm} (13)

In our format, only structures to which a structure change or a condition applies need be numbered. Any structure, including features and dominating and dominated symbols, may be numbered. (However, the variable $X$ is subject to certain limitations.) A number immediately preceding a choice set applies to each of the left-most members of the set that are not already numbered. Thus, if a certain proper analysis of the "current" tree is found, the change $1 = 2 + 1$ will add the structure headed by $C$ or $D$ as a left sister to the structure headed by $B$ and then erase the original $C$- or $D$-dominated structure. If, however, the proper analysis was found with $*3 \ F$ following $*1 \ B$, the structure headed by $F$ is erased. This rule applied to the tree in Figure 3 yields the tree in Figure 4.

A special condition on this transformation, which must be satisfied before any structural changes can be performed, is that the structure headed by $*4 \ B$ be the same as the structure of $*1 \ B$. TGT is designed to facilitate the addition of conditions on transformations. There is no limit to the number of conditions that may be imposed on a structure.

Rule H2 in (14) below introduces a notation for expressing proper analyses of subtrees within the current tree, and for specifying fixed length variables and features.

RULE H2 $X [B *1C \ D] A / < + F >> + G > / E *2N$

$*3P [X / <+H - I >] / [Z X] ] X => 1 = 1$

$+ *4M; 4 < L 2; 4 < L 3 \hspace{1cm} (14)$

\*A detailed description of the algorithm used to determine proper analyses and to perform structure changes can be found in "An Algorithm for Pattern Matching and Manipulation with Strings and Trees" by D. Londe and W. Schoene, System Development Corporation document (in press).11

\**A formal description of proper analysis can be found in Petrick [1965].6
Square brackets indicate that the first symbol following the right bracket (]) must dominate the sequence of symbols inside the brackets and, furthermore, that this sequence must constitute a proper analysis of the dominating symbol.

TGT permits bracket nesting to any level. If the linguist is concerned only that a symbol A dominate a symbol B regardless of whatever else it may dominate, he can surround the B with general variables, [X B X]A, thus preserving the convention for interpretation requiring a proper analysis. *Immediate* dominance can be expressed as a special condition on the dominated symbol, i.e.,

\[ X[X \times 1B X] A X \Rightarrow \text{structure changes.} \text{I COND5.} \]

Where condition 5 requires that the node to which it applies (in this case \( \times 1B \)) be immediately dominated by the symbol immediately following the right bracket (A).

Because of the comparatively recent acceptance of features in transformational grammars and the variety of their function in different writings, TGT handles them internally as if they were special conditions on the symbols immediately dominating them, thus facilitating modification. Rule H2 specifies that the E following \([B C D] A\) must *immediately* dominate the features \(< + F >\) and \(< + G >\). Where the linguist is concerned that a symbol dominate a particular feature at some unspecified distance, he may make use of the notation indicated by \([X/ \times 1H-1] /Z X]J\) in rule H2, where Z is a variable of fixed length, one node. We thus preserve our convention of specifying the immediate dominator of features.

A very convenient innovation is illustrated in the structure change portion of this rule. Generally structure changes manipulate structures that are present in the tree before execution of the rule. Thus, when we specify that some structure numbered 2 is to be added as a daughter to some structure 1, and that 1 is to be adjoined as a right sister to 3, we intend to adjoin 1 in its original state, i.e., before 2 was added.

In TGT we have provided a notation that allows numbers to be assigned to structures that are created by the structural change portion of the transformation, whether these be newly introduced constants or structures that were already in the tree and moved to new positions. These new structures may subsequently be moved or have structures adjoined to them. In H2 we are adding a new symbol M as a right sister to \(*1C\), and to this new symbol we are adding \(*3P\) and \(*2N\) as left daughters. Rule H2 applied to the tree in Figure 5 yields the tree in Figure 6.

RULE is used only to define transformations and assign them names. (As with trees, TGT informs the user when the name is not unique.) Transformations are normally assigned a position, an order of operation, * in a cycle or post-cycle. TGT readily allows the user to define and redefine the order of operation (see discussion of INSERT, Section D).  

Figure 5 — Sample tree to which transformational rule H2 is applied

Figure 6 — Results of applying rule H2 to the tree in Figure 5

*Chomsky [1965]* mentions the possibility of intrinsically ordered rules. Although we could have accommodated this concept in TGT by arbitrarily assigning numbers to rules and using a random number generator, we deferred this until someone attempts to write a grammar where the rules are not extrinsically ordered.
TEST, EXECUTE and DERIVE actually apply the transformations to the current tree. In general, TEST is used to determine whether the structural description of a transformation successfully applies to the current tree.

The TEST command is accompanied by the name of a rule or the names of the first and last rule of a rule sequence, or the word ALL, in which last case every transformation in the cycle and post-cycle is matched against the current tree. The name of the rule and the message YES or NIL are output for each rule as it is tested. For each transformation that successfully applies, the user is given the option to execute the structural changes of the transformation and continue testing, to execute and wait for another command, to continue testing without changing the current tree, or to wait for a new command.

EXECUTE may specify one rule or the first and last of a sequence of rules. No message, other than the rule names and YES or NIL is output. This command is used in preference to TEST when the user knows that he wants the structural changes of each successful transformation to be performed.

DERIVE applies each transformation in the cycle to the “lowest” S in the tree. It recomputes “lowest” S after each cycle until all subtrees have been processed. It then applies all of the postcyclic rules. DERIVE records the S node number associated with each cycle and the transformations that were executed during the cycle.

It is not meaningful to talk of the execution times of transformations because they will vary with the tree, the rule, the conventions of interpretation, and, of course, the computer. However, to give some rough idea of the speed of TGT on the Q-32, we input rules 1-13 of IBM Core Grammar I (except rule 4, which has complex special conditions on its application) and applied them to test trees 28 and 29 of that grammar. The average execution time per rule was .04 seconds.

D. Other capacities

The command MATCH enables the linguist to determine easily how a successful transformation applies to a tree, by printing on the TTY those symbols of the structural description that were successfully matched by a node in the tree and by printing that node number. After HYPOTHES1 has applied to the tree in Figure 3, the command MATCH would output:

\[
\begin{align*}
5 & \quad X \\
7 & \quad A \\
8 & \quad B \\
18 & \quad C \\
21 & \quad G \\
24 & \quad B
\end{align*}
\]

Any successful transformation may be reapplied to determine whether it could have applied to the current tree in more than one way. The command AGAIN causes the search for a proper analysis to continue by successively backing up from the last node matched as if there had been no match. Thus, in (15) above, the rule interpreter would look below node 24 for another B; finding none it retreats to G 21, ignores the original match and eventually finds the analysis below:

\[
\begin{align*}
5 & \quad X \\
7 & \quad A \\
8 & \quad B \\
18 & \quad C \\
23 & \quad G \\
24 & \quad B
\end{align*}
\]

Applied once more, AGAIN would eventually look below 18 for a C, a D followed by an E, or an F, and it would find the analysis indicated by (17) below:

\[
\begin{align*}
5 & \quad X \\
7 & \quad A \\
8 & \quad B \\
33 & \quad F \\
20 & \quad X \\
21 & \quad G \\
24 & \quad B
\end{align*}
\]

The analysis that includes 33F and 23G can be attained by invoking a special condition, condition 6. This suspends the A over A convention* allowing the interpreter to go below a symbol, A, to find another A. This condition is symbol specific in TGT and thus may be selectively applied among the symbols of a structure description.

Where a successful match is made on an optional symbol immediately preceded or immediately followed by an X, there is always at least one alternate analysis, that where the NIL is chosen.

*Variables are not printed when they encompass no nodes.
**The reference to the A over A convention under (17) is pertinent to this analysis.
* A discussion of this convention and exceptions can be found in Chomsky [1962] and Chomsky [1964].

*Suggestions have been made recently, however (e.g., UCLA Conference on English Syntax, summer 1967) that each recursive symbol in the phrase structure rules be so treated.
For each transformation, the command INSERT allows the user to specify information concerning its operation in the rule set: its order of operation; whether it is cyclic or postcyclic; whether it is obligatory or optional; whether (if successful) it may be reapplied to the tree before the next sequential transformation; and whether the interpreter may look below an S in trying to find a proper analysis.

The command EDIT facilitates the correction of faulty transformation input by removing the necessity to retype the entire rule. This command is accompanied by two strings of symbols, X and Y. EDIT searches the erroneous rule for string X and replaces it with string Y.

A printout of the names of all the trees in the system at any time can be obtained with the command TREES. Similarly, all rule names are printed in response to the command RULES. The primitive DISPLAY and LIST also accept rule names as input and will either print on the TTY or display on the CRT scope the appropriate transformation.

Trees and rules are erased from the system's memory by means of the DELETE command accompanied by the rule or tree name.

Upon termination of a run, TGT presents the user with the opportunity of saving on magnetic tape the trees and rules he may have created during the run. Each initiation of TGT allows the user to input a file of trees and rules from tape, or to start up with neither rules nor trees in the system. Thus, many versions of a grammar may be extant and may be undergoing modification and testing simultaneously. Files of trees, rules, and derivations may thus be accumulated from run to run and may be subsequently used by others and merged with the files created by other linguists.

The program

The organization of the TGT program is straightforward, consisting of an executive routine and an indefinitely extensible set of service routines, which perform the basic system tasks such as tree creation and manipulation, rule testing, and generation of displays and teletype output. In TGT's basic interactive cycle, the executive routine interprets each teletype request to determine which service routine is needed, reads and legality checks the parameters associated with the request, and transfers control to the appropriate service routine. At the completion of its task, the service routine returns control to the executive routine and the cycle is ready to begin again.

Of more particular interest are the functions associated with the interpretation of transformations. When the transformation is input each symbol name is looked up in the dictionary and is replaced by its dictionary number. If the symbol is not present it is added to the dictionary.

The rule is then converted to a form convenient for interpretation and resides in a table whose capacity is, at present, 1100 entries. Each symbol in the transformation occupies an entry.

Each entry contains the following information: dictionary entry number of this symbol; location in this table of the next sequential symbol; location of previous sequential symbol; location of next optional symbol (in the case of a symbol within choice parentheses); location of the first special condition on this symbol.

\[ X * 1A ((C D) * 2Z, F G) T \Rightarrow 1 = 0. Z N Q 1. (18) \]

The transformation (18) above would be represented in the rule table as indicated below.

<table>
<thead>
<tr>
<th>Number</th>
<th>Dictionary Entry of Symbol*</th>
<th>Next Sequential</th>
<th>Previous Sequential</th>
<th>Optional Next</th>
<th>Special Condition Pointer/ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Z</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>G</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>T</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Entry 6 above represents the special condition imposed on the symbol Z. The number 4 indicates which condition is imposed (nonequality), and the number 2 under column "Optional Next" of entry 6 is a parameter pointing to the structure headed by the symbol A in entry 2.

We handle recursion as a special condition. Thus the symbol Z has two conditions 4 and 1. Condition 1 is always the last condition to operate since it essentially effects a re-entry of the procedure, at which point a proper analysis consisting of the symbols C and D is attempted for the substructure headed by this Z.

The numbers of the tree nodes corresponding to rule symbols are saved in this table so that the tree can be changed in case the pattern match is successful, and so that the pattern matching algorithm can try all possibilities.

*For convenience the actual symbol is used here.
If, for example, the node to the right of A in the tree does not dominate a C and D and is not the symbol F, the pattern matching algorithm will back up to the tree node recorded in entry 2 as matching rule symbol A. The variable X is extended to encompass this A and the algorithm attempts to find another A.

A flow diagram and more details of the program can be found elsewhere.\textsuperscript{10,11}

DISCUSSION

We feel that TGT is a tool that is flexible in its capacity to perform operations and is convenient with regard to the manner in which commands may be expressed. We found that within the present framework we could often handle conditions attached to the structure change portion of a transformation, although we had not intentionally designed the system to do this. Other difficulties have been resolved by breaking complex rules into several simpler rules. We anticipate that many future problems will be solved by expanding the existing set of conditions, which can be expressed directly in the rule. Some other linguistic capabilities that the user may need are less tractable.

The advisability of using an entirely different rule schema for handling conjunction has been demonstrated in Schane [1966]\textsuperscript{14} and Schachter [1967].\textsuperscript{15} Plans to program the Schachter schema and integrate it into the current model of TGT are under way.

Several different conventions regarding the movement of tree branches by structure changes have been provided internally, but have not yet been made easily accessible to the user. More programming would be necessary, however, were a set of tree-pruning conventions, such as suggested in Ross [1965],\textsuperscript{16} to be adopted.

A computer system such as TGT can make significant contributions in testing a lexicon and integrating it into a transformational grammar. We are at present familiarizing ourselves with various proposals regarding the form of the lexicon and conventions for lexical insertion.

The authors have no illusions regarding the comprehensiveness or generality of TGT. Transformational grammar is in a dynamic state of development. By the time a system of programs is written and checked out, it is in danger of being obsolete. However, a linguist who sets out to write a grammar faces the same problems. Once he has written a number of rules, he may not readily change rule conventions unless he is willing to play the Red Queen, who must run as fast as she can just to keep from falling behind.

ACKNOWLEDGMENTS

We are indebted to Dr. Barbara Hall Partee of AFESP for her criticisms and suggestions regarding the design and use of TGT, to John Olney of System Development Corporation for his contributions to the initial specifications, and to Dr. Joyce Friedman of Stanford University for suggesting the utility of a single node variable.

REFERENCES

\begin{enumerate}
\item N CHOMSKY
  Aspects of the theory of syntax
  THE MIT Press Cambridge Massachusetts 1965
\item P S ROSENBAUM D LOCHAK
  The IBM core grammar of English
  In Specification and Utilization of Transformational Grammar
  Scientific Report no 1 IBM Corporation Yorktown Heights
  New York 1966
\item N CHOMSKY
  The formal nature of language
  Appendix A
  In E H Lenneberg Biological Foundations of Language John
  Wiley & Sons New York 1967
\item D BOBROW
  Syntactic theories in computer implementations
  In H Borko Automated Language Processing: The State of the
  Art John Wiley & Sons New York
\item BUNKER RAMO CORPORATION
  A syntactic analyzer study
  Final Report under Contract AF 30(602)-3506 Canoga Park
  California 1965
\item S R PETRICK
  A recognition procedure for transformational grammars
  Unpublished Doctoral Dissertation MIT Cambridge
  Massachusetts 1965
\item A M ZWICKY J FRIEDMAN B C HALL
  The MITRE syntactic analysis procedure for transformational
  grammars
  MTP-9 The MITRE Corporation Bedford Massachusetts
  August 1965
\item L N GROSS
  On-line programming system user's manual
  MTP-59 The MITRE Corporation Bedford Massachusetts
  March 1967
\item F BLAIR
  Programming of the grammar tester in specification and
  utilization of a transformational grammar
  Scientific Report no 1 IBM Corporation Yorktown Heights
  New York 1966
\item W J SCHOWEN
  TGT user's manual
  System Development Corporation document in press
\item D L LONDE W J SCHOWEN
  An algorithm for pattern matching and manipulation with
  strings and trees
  System Development Corporation document in press
\item N CHOMSKY
  The logical basis of linguistic theory
  Ninth International Congress of Linguistics
  Cambridge Massachusetts 1962
\end{enumerate}
13 N CHOMSKY
Current issues in linguistic theory
In Fodor and Katz The Structure of Language Prentice-Hall
Englewood Cliffs New York 1964

14 S A SCHANE
A schema for sentence coordination
MTP-10 The MITRE Corporation Bedford Massachusetts
April 1966

15 P SCHACHTER
A schema for derived constituent conjunction
Unpublished Working Paper UCLA Air Force
English Syntax Conference September 1967

16 J R ROSS
A proposed rule of tree pruning
Unpublished paper presented to the Linguistic Society of
American December 1965