# THE MITRE SYNTACTIC ANALYSIS PROCEDURE FOR TRANSFORMATIONAL GRAMMARS*

Arnold M. Zwicky,† Joyce Friedman,†
Barbara C. Hall,† and Donald E. Walker
*The MITRE Corporation*
*Bedford, Massachusetts*

## INTRODUCTION

A solution to the analysis problem for a class of grammars appropriate to the description of natural languages is essential to any system which involves the automatic processing of natural language inputs for purposes of man-machine communication, translation, information retrieval, or data processing. The analysis procedure for transformational grammars described in this paper was developed to explore the feasibility of using ordinary English as a computer control language.

*The research reported in this paper was sponsored by the Electronic Systems Division, Air Force Systems Command, under Contract AF19 (628) 2390. The work was begun in the summer of 1964 by a group consisting of J. Bruce Fraser, Michael L. Geis, Hall, Stephen Isard, Jacqueline W. Mintz, P. Stanley Peters, Jr., and Zwicky. The work has been continued by Friedman, Hall, and Zwicky, with computer implementations by Friedman and Edward C. Haines. Walker has directed the project throughout. The grammar and procedure are described in full detail in reference 1. This paper is also available as ESD-TR-65-127.

†Present addresses of Zwicky, Friedman and Hall are, respectively: Department of Linguistics, University of Illinois, Urbana; Computer Science Department, Stanford University, Stanford, Calif.; Department of Linguistics, University of California, Los Angeles.

### *The Problem of Syntactic Analysis*

Given a grammar‡ G which generates a language L(G), we can define the *recognition problem for G* as the problem of determining for an arbitrary string x of symbols, whether or not $x \in L(G)$. The more difficult problem of *syntactic analysis* is to find, given any string x, all the structures of x with respect to G.

The syntactic analysis problem varies with the class of grammars considered, since both the formal properties of the languages generated and the definition of *structure* depend on the form of the grammar.

A context-free (CF) phrase-structure grammar is a rewriting system in which all rules are of the form $A \rightarrow \varphi$, where $\varphi$ is a non-null string and A is a single symbol; in context-sensitive (CS) phrase-structure grammars all rules are of the form $\psi_1 A \psi_2 \rightarrow \psi_1 \varphi \psi_2$, where A and $\varphi$ are as before and $\psi_1$ and $\psi_2$ are strings (possibly null) of terminal and/or nonterminal symbols. A derivation in a phrase-structure grammar is represented by a tree in which the terminal elements constitute the derived string. In a transformational grammar there is, in addition to a phrase-structure

‡The linguistic concepts on which this work is based are due to Noam Chomsky; see, for example, references 2-6.

component, a set of transformational rules which operates upon trees from the phrase-structure component to produce trees representing sentences in their final form.

For both types of phrase-structure grammars the syntactic analysis problem is known to be solvable. In the case of CF grammars, a number of general recognition and syntactic analysis procedures have been developed and programmed. Several syntactic analysis algorithms for CS grammars given by Griffiths (1964).[8]

The case of transformational grammars is complicated by the fact that they do not correspond exactly to any well-studied class of automata. In fact, a number of decisions crucial to their formalization have yet to be made. This situation makes it impossible to describe a general recognition procedure for transformational grammars without explicit conventions about the form and operation of transformational rules. Since there is no widespread agreement as to the most desirable conventions, it is likely that different people working on the analysis problem for transformational grammars are actually working on quite different problems. A solution to the problem with one set of conventions will not necessarily be a solution to the problem with a different set of conventions. Furthermore, the solution in one case would not necessarily imply the existence of a solution in another.

The area of formal properties of transformational grammars needs more study; the results of this attempt to solve the syntactic analysis problem for a particular one may help in determining the further restrictions needed on the form of transformational rules.

## THE MITRE GRAMMAR

In order to develop an analysis procedure it was necessary to fix on a particular set of conventions for transformational grammar. Many of these conventions agree essentially with the more or less standard conventions in the literature; points on which general agreement has not been reached will be noted.

The grammar contains two principal components: a CS phrase-structure component and a transformational component.* The rules in the phrase-structure component serve to generate a set of *basic trees* that are then operated upon by the rules of the transformational component to produce a set of *surface trees*.

### *Phrase-structure Component*

A CS phrase-structure rule $\psi_1 A \psi_2 \rightarrow \psi_1 \varphi \psi_2$ is written in the form $A \rightarrow \varphi / \psi_1 - \psi_2$. $\psi_1$ or $\psi_2$ or both may be null. The rules are ordered; consecutive rules expanding the same symbol in the same context are considered to be subrules of a single rule. A rule in this sense is thus an instruction to choose any *one* of the specified expansions.

The initial symbol of the grammar is SS, and the first phrase-structure rule is SS → # S #.† Further instances of SS and S are introduced by later rules. These instances are expanded during a succeeding pass through the phrase-structure rules during which new instances may be introduced, etc. The result is a tree that may contain sentence-within-sentence structures of arbitrary depth. This version of the phrase-structure component differs somewhat from the more usual versions, but is similar to the version presented in Chomsky.[3]

We shall use the following tree terminology: x is a *daughter* of y, x (not necessarily immediately) *dominates* y, x is the (immediate) *right* (*left*) *sister* of y, x is *terminal*, and the sequence $x_1, x_2, \ldots, x_n$ is a (proper) *analysis* of x. We shall also refer to the (*sub*) *tree headed* by x. These terms are all either standard of self-explanatory.

### *Transformational Component*

*Form of the Rules.* A transformational rule specifies a modification of a tree headed by the node SS or S. Every such rule has two main parts, a *description statement* and an *operation statement*.

The description statement sets forth general conditions that must be satisfied by a given tree. If these conditions are not met, then the rule cannot be applied to the tree. The conditions embodied in a description statement are conditions on analyses of

*There is a third component, the lexicon, which will not be discussed in detail here.

†The first phrase-structure rule in the MITRE grammar differs from this rule by allowing for the conjunction of any number of sentences. SS may then dominate a sequence of conjoined sentences. S, on the other hand, never immediately dominates such a sequence.

sentences (trees headed by SS, # S #, or S‡); a description statement is to be interpreted as requiring that the given tree have at least one analysis out of a set of analyses specified by the description statement.

If a tree satisfies the conditions embodied in a description statement, then the operations apply to the subtrees headed by the nodes in the analysis. The operation statement lists the changes to be made — the deletions, substitutions, and movements (adjunctions) of subtrees.

In addition to a description statement and an operation statement, a transformational rule may involve a number of *restrictions*. A restriction is an extra condition on the subtrees. The extra condition is either one of equality (one subtree must be identical to another) or of dominance (the subtree must contain a certain node, or must have a certain analysis, or must be a terminal node). Boolean combinations of restrictions are permitted.

The form of a transformational rule can be illustrated by the following example:

```
TWH2
(#) (Q) ($NIL NG) (AUXA) ($SKIP NP AP $RES 19)
 1   2      3       4            5
(5) ADLES 4                  $RES 19:    dom WH
ERASE 5
```

The description statement of this rule (TWH2) consists of five numbered and parenthesized *description segments*. Each segment specifies one part of an analysis. When several grammatical symbols (symbols not beginning with $) are mentioned in a segment, the interpretation of the segment is that the corresponding part of the analysis must be a subtree headed by *one* of these symbols. When $NIL is mentioned in a segment, the interpretation is that the corresponding part of the analysis is optional—that is, the corresponding part may be a null subtree; if, however, some analysis can be found in which the corresponding part is *not* null, that analysis must be chosen. The occurrence of $SKIP in a segment is equivalent to a variable between that segment and the preceding one.* $RES must be followed by the number of the restriction to which it refers. There is an implicit variable at the end (but not at the beginning) of every description statement.

In a more informal and traditional notation, the description statement of TWH2 would be written as

$$\underbrace{\#\ +\ Q\ +\ (NG)}_{1}\ -\ AUXA\ -\ \underset{2}{X}\ -\ \underset{3}{\begin{Bmatrix} NP \\ AP \end{Bmatrix}}\ -\ \underbrace{Y\ +\ \#}_{5}$$

In our system there is no way of referring to a sequence of subtrees as a single part of an analysis, althiugh there is in the more informal notation.

In outline, the routine that searches through a tree for an analysis that conforms to a given description statement searches from left to right through the tree, attempting (in the case of a segment containing $NIL) to find a real node before assuming that a segment is null, attempting always (in the case of a segment containing $SKIP) to "skip" the smallest possible number of nodes, and checking (in the case of a segment containing $RES n) to see if a restriction is satisfied as soon as a node to which the restriction applies is found. In case one part of the search fails, either because the required nodes cannot be found or because a restriction is not satisfied, the routine backs up to the most recent point at which there remains an alternative (e.g., the alternative of searching for NP or for AP in the fifth segment of TWH2). As each part of the analysis is found, the appropriate subtrees are marked with numbers corresponding to the numbers on the description segments. The tree then undergoes the modifications specified in the operation statement.

The operation statement of TWH2 consists of an (ordered) list of two *instructions*. There are three types of instructions: the *adjunction* instructions, the *substitution* instruction, and the *erasure* instruction. The adjunction instructions are of the form ($\varphi$) AD n, where $\varphi$ is a sequence containing numerals (referring to the marked subtrees) or particular grammatical symbols or both, where AD is one of the four adjunction operations — ADLES (add as left sister), ADRIS (add as right sister), ADFID (add as first daughter), or ADLAD (add as last daughter) — and where n is a numeral referring to a marked subtree. The instruction (5) ADLES 4 specifies the adjunction of a copy of the subtree marked 5 as the left sister of the node heading the subtree marked 4. Substitution instructions are of the form ($\varphi$) SUB n, where $\varphi$ and n are as before. When such an instruction is applied, copies of the elements of $\varphi$ replace the

‡This distinction is not important for our discussion here. See the discussion in reference 1.

*$SKIP and $NIL may not both be used in a single segment.

subtree marked n, and this subtree is automatically erased.*

Erasure instructions are of the form ERASE n (erase the subtree marked n and any chain of nonbranching nodes immediately above this subtree) or ERASE Ø (erase the entire tree). The ERASE Ø instruction permits us to use the transformational component as a "filter" that rejects structures from which no acceptable sentence can be derived.

*Derivations.* The transformational rules are distinguished as being *obligatory* or *optional, cyclical* or *noncyclical,* and *singulary* or *embedding.* The obligatory/optional distinction requires no special comment here.

A rule is cyclical if it can be applied more than once before the next rule is applied. A rule may be marked as cyclical either (a) because it can be applicable in more than one position in a given sentence (say, in both the subject and object noun phrases), or (b) because it can apply once to yield an output structure and then apply again to this output. Otherwise, the rule is marked as noncyclical. In the present grammar case (b) does not occur.

Singulary rules are distinguished from embedding rules on the basis of the conditions placed upon the tree search. In the case of a singulary rule the search cannot continue "into a nested sentence"—that is, beneath an instance of SS or S within the sentence being examined; the search may, of course, pass over a nested sentence. In the case of an embedding rule the search can continue into a nested sentence, but not into a sentence nested in a nested sentence. Singulary rules operate "on one level," embedding rules "between one level and the next level below."

The transformational rules of our grammar are grouped into three sets—a set of initial singularies, a set of embeddings with related singularies, and a set of final singularies.† The rules are linearly ordered within each set.

The initial singularies operate on the output of the phrase structure component; they can be considered as applying, in order, to all subtrees simultaneously, since these rules do nothing to disturb the sentence-within-sentence nesting in a tree. There are numerous ways to order the application of these rules with respect to the nesting structure of a tree, and they are all equivalent in output.

The embeddings and related singularies operate on the output of the initial singularies. These rules require a rather elaborate ordering. Let us define a *lowest sentence* as an instance of # S # in which S does not dominate # and a *next-to-lowest-sentence* as an instance of # S # in which S dominates at least one lowest sentence and no instance of # S # that are not lowest sentences. At the beginning of the first pass through the embeddings and related singularies, all lowest sentences are marked. The rules will be applied, in order, to the marked subtrees. At the beginning of each subsequent pass, all next-to-lowest sentences will be marked, and the rules will again be applied, in order, to all marked subtrees. Characteristically, the embedding rules, when applied during these later passes, erase boundary symbols and thus create new next-to-lowest sentences for the following pass. However, only those subtrees marked at the beginning of a pass can be operated upon during the pass. The process continues until some pass (after the first) in which no embedding rules have been applied.

The final singularies operate on the output of the embeddings and related singularies. They can be considered as applying, in order, to all subtrees simultaneously.

A tree that results from the application of all applicable transformational rules is a *surface tree.* Each surface tree is associated with one of the sentences generated by the grammar.

*Dimensions*

The MITRE Grammar generates sentences with a wide variety of constructions—among them, passives, negatives, comparatives, *there*-sentences, relative clauses, yes-no question, and WH-questions. The dimensions of the grammar (excluding all rules concerned with conjunction) are as follows:

Phrase Structure Component:

Transformational Component:

75 rules
approximately 275 subrules

13 initial singularies

*If $NIL is chosen in the nth description segment, then ($\varphi$) AD n or ($\varphi$) SUB n is vacuous. Null terms in $\varphi$ are ignored; if all of $\varphi$ is null the instruction is vacuous.

†There is also a fourth set, conjunction rules. Because of the treatment of conjunction in the "English Preprocessor Manual"[1] is currently being revised, conjunction has been omitted from this presentation.

26 embeddings and related singularies, including 9 embeddings

15 final singularies

54 rules

## THE MITRE ANALYSIS PROCEDURE

The MITRE analysis procedure takes as input an English sentence and yields as output the set of all basic trees underlying that sentence in the MITRE grammar. If the procedure yields no basic tree, the input sentence is not one generated by the grammar. If the procedure yields more than one basic tree, the input sentence is structurally ambiguous with respect to the grammar.

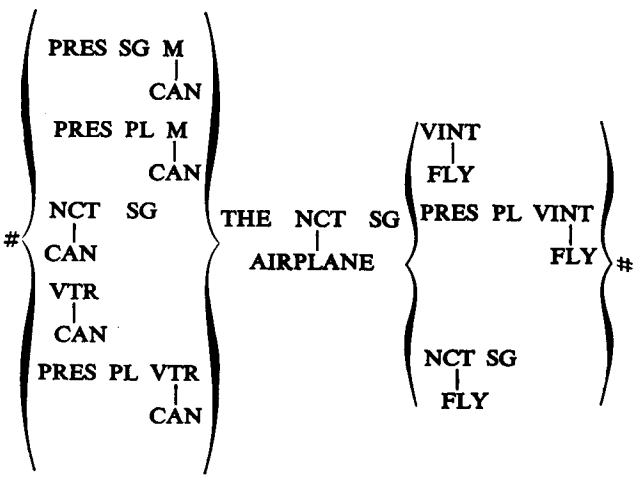
There are five parts to the procedure: lexical look-up, recognition by the surface grammar, reversal of transformational rules, checking of presumable basic trees, and checking by synthesis. These parts are described in detail in the following sections.

### *Lexical Look-up*

The first step of the process is the mapping of the input string into a set of *pre-trees,* which are strings of subtrees containing both lexical and grammatical items. The pre-trees are obtained from the input string by the substitution of lexical entries for each word.

A lexical entry for a word may be identical to the word (in the case of grammatical items like A and THE). More often, a lexical entry for a word indicates a representation of the word in terms of more abstract elements (NEG ANY for NONE), a category assignment for the word ( $\frac{\text{ADJ}}{\text{GREEN}}$ for GREEN), or a combination of abstract representation and category assignment ( $\frac{\text{PRES SG VTR}}{\text{OPEN}}$ for OPENS). A word may have several lexical entries.

The number of pre-trees associated with an input string is then the product of the numbers of lexical entries for the words in the string. Thus, the string # CAN THE AIRPLANE FLY # has 15 associated pre-trees, which can be schematically represented as:




Of these 15 pre-trees, only


is a correct assignment of lexical entries to the words in the input string.*

### *Recognition by the Surface Grammar*

The surface grammar is an ordered CF phrase-structure grammar containing every expansion which can occur in a surface tree. Unavoidably, the surface grammar generates some trees which are not correct surface trees, even though the corresponding terminal string may be a sentence obtainable by the grammar with some other structure.

In the second step of the analysis procedure the surface grammar is used to construct from each pre-tree a set of *presumable surface trees* associated with the input string. Since the surface grammar is context-free, and context-free parsing algorithms are known to exist, no details will be given here for this step of the analysis.

In the course of recognition by the surface grammar, some pre-trees may be rejected. For example, 9 of the 15 pre-trees in the previous section are rejected in this way. From other pre-trees one or more presumable surface trees will be constructed.

The remaining steps of the analysis procedure are designed to determine, for each presumable surface tree, whether or not the tree is in fact a surface tree for the input sentence.

*Since the MITRE grammar generates neither imperatives nor noun-noun compounds, the interpretation of CAN THE AIRPLANE FLY as analogous to CORRAL THE SADDLE HORSE is excluded.

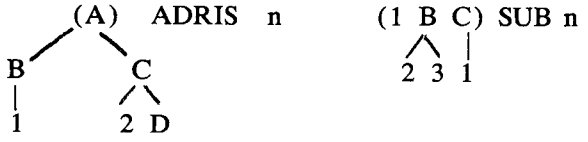
*Reversal of Transformational Rules*

The next step in the analysis procedure reverses the effect of all transformational rules that might have been applied in the generation of the given presumable surface tree.

The "undoing" of the forward rules is achieved by rules that are very much like the forward rules in their form and interpretation. The discussion under Form of the Rules, above, applies to reversal rules as well as to forward rules, with the following additions:

(a) There is a new adjunction instruction, ADRIA (add as right aunt — that is, add as right sister of the parent).

(b) Adjunction and substitution instructions have been generalized to permit instructions like:




Such instructions are used to restore entire subtrees deleted by forward rules.

(c) In the reversal of optional forward rules, a marker OPTN is added as a daughter of a specified node, which in every case is either terminal or else has only a lexical expansion. Some such device is required if the result of the final synthesis step is to correspond to the original input string. The constraint on the placement of OPTN insures that the marker will not interfere with the operation of other reversal rules.

As with forward rules, reversal rules are either cyclical or noncyclical, and either singulary or embedding. All reversal rules are obligatory.*

The reversal rules are grouped together in the same way as the forward rules, and the order of their application within each group is essentially the opposite of the order of the corresponding forward rules. In many cases, one reversal rule undoes one forward rule. There are three types of exceptions, however: (a) several reversal rules may be required to attain the effect of undoing a single forward rule; (b) for some rules, notably the rules with ERASE Ø instructions, no reversal is needed; (c) in some cases the reversing of several forward rules can be combined in whole or in part into a single reversal rule.

Reversed final singularies are first applied to all subtrees. Then reversed embeddings and related singularies are applied in several passes. The first pass deals with the highest sentences in the tree. Later passes move downward through the tree, one level at a time. New lower sentences are created when boundary symbols are inserted during the reversal of embedding transformations; in general, a sentence created on one pass is dealt with on the next. Finally, reversed initial singularies are applied everywhere.

The effect of transformational reversal is to map each presumable *surface* tree into a presumable *basic* tree.†

*Checking of Presumable Basic Trees*

In the next step of the analysis procedure, each presumable basic tree is checked against the phrase-structure component of the (forward) grammar. The check determines whether or not the presumable basic tree can in fact be generated by the phrase-structure component; if it cannot, it is discarded.

Checking by Synthesis) It is possible that transformational reversal and phrase-structure checking could map a presumable surface tree $T_1$ into a basic tree $T_2$ that is not the basic tree underlying $T_1$. For example, the reversal rules map at least one presumable surface tree associated with THOSE PIG IS HUGE into a basic tree underlying THAT PIG IS HUGE. Even under the assumption that input sentences are grammatical, the possibility remains. For example, the reversal rules map at least one presumable surface tree associated with THE TRUCK HAS A SURFACE THAT WATER RUSTS into a basic tree underlying THE TRUCK HAS A SURFACE THAT RUSTS. Similarly, they map at least one presumable surface tree associated with THEY CAN FISH into a basic tree underlying THEY CAN A FISH.

Revision of the present reversal rules and the introduction of rejection rules into the transformational reversal step might make a synthesis step

*Optional reversal rules are required when two distinct basic trees are mapped into identical surface trees by the application of forward rules. No such example occurs in the present MITRE grammar.

†Distinct presumable surface trees may be mapped into identical presumable basic trees; the resultants of distinct presumable surface trees will continue to be processed separately, however.

unnecessary. However, the above examples demonstrate that with the present rules this step is essential.

In the synthesis step, the full set of forward transformational rules is applied to each basic tree that survives the previous checking step. Each optional rule becomes obligatory, with the presence of the marker OPTN (in the appropriate position) as an added condition on its applicability.

The synthesis step maps a basic tree $T_2$, derived from a presumable surface tree $T_1$, into a surface tree $T_3$. If $T_1$ and $T_3$ are not identical, then $T_2$ is discarded as a possible source for the input string. If $T_1$ and $T_3$ are identical, then $T_2$ is a basic tree underlying $T_1$ (and hence, underlying the input string).

*Dimensions*

The dimensions of the additional components of the analysis procedure are as follows:

| | |
|---|---|
| Surface Grammar: | 49 rules |
| | approximately 550 subrules |
| Reversal Rules: | 30 final singularies |
| | 92 embeddings and related singularies |
| | 12 initial singularies |
| | 134 rules |

## AREAS FOR FURTHER INVESTIGATION

We are investigating a number of problems both in the grammar and in the analysis procedure, with the objectives of making the grammar more adequate and the procedure more efficient.

Among the grammatical problems are the use of syntactic features (see Chomsky[3]) and the addition of further rejection rules in the transformational component. The treatment of conjunction is being revised. Other topics requiring investigation include adverbial clauses, superlatives, verbal complements, imperatives, and nominalizations.

We are examining a number of ways to improve the efficiency of the analysis procedure. If the input vocabulary is to be of an appreciable size, an efficient and sophisticated lexical look-up routine will be required. We are using computer experiments to determine the extent to which the use of a CS surface grammar, either as the basis of a CS parsing routine or as a check on the results of CF parsing would improve the procedure by eliminating some incorrect surface trees at an early stage.

Some increase in the efficiency of the reversal step might be achieved by making use of a preprogrammed path through the reversal rules, or by using information that certain surface grammar rules signal the applicability or inapplicability of certain reversal rules. Similarly, the efficiency of the final synthesis step might be improved by making use of a preprogrammed path through the forward transformational rules, or by using information that certain reversal rules have been applied.

*Analysis by Synthesis*

The first analysis procedure proposed for transformational grammars was the "analysis by synthesis" model of Matthews.[9] Basically this procedure involves generating sentences until one is found which matches the input sentence; the steps used in the generation provide the structural description. No attempt to program the analysis-by-synthesis procedure for transformational grammars has been reported in the literature. In its raw form this procedure would take an astronomically long time. One way to refine the procedure would be to use a "preliminary analysis" of some sort, which would have to be extensive to make any appreciable change in efficiency. As a result, there may be no sharp boundary between refined analysis-by-synthesis and direct analysis with a final checking-by-synthesis step. In the case of the MITRE procedure the final synthesis step plays a relatively minor role in the total procedure.

*Petrick's Procedure*

S. R. Petrick[10] has proposed and programmed a general solution to the analysis problem which is similar in many respects to the MITRE procedure. One of the main differences between his approach and ours is that he alternates the use of reversal rules and phrase-structure rules, while we use first the phrase-structure rules of the surface grammar and then the reversal rules. Furthermore, while Petrick's reversal rules are all optional, ours are all obligatory. It follows that although we may have a larger number of structures to consider at the beginning of reversal step, this number does not increase as it does at every step in Petrick's procedure.

At the present time the procedures differ in gen-

erality, for Petrick has shown that there are algorithms for the construction of his surface grammar and reversal rules. In the case of the MITRE procedure, the question of the existence of comparable algorithms has not yet been resolved.

*Kuno's Procedure*

Another approach to the analysis problem has been proposed in Kuno.[11] Kuno attempts to find basic trees, without using reversal rules, by constructing a context-free surface grammar and associating with each of its rules information about the form of the basic tree.

Kuno reported that an experimental program for this system had been written and was being tested on a small grammar. At that time it was not known whether an algorithm for constructing the required phrase-structure grammar existed.

## COMPUTER TESTS

To test the grammar and the procedure a set of FORTRAN subroutines (called SYNN), designed to be combined in several different programs, has been written. In one order, the subroutines carry out the procedure from the stage at which presumable surface trees have been obtained, through the base tree, to the final step of comparison of the derived surface tree with the given presumable surface tree. In other orders they can, for example, convert base trees to surface trees and back, or check surface trees against context-sensitive grammars.

We describe first the subroutines, in groups corresponding to the major components of the MITRE procedures, then some of the programs and the results of running the programs on a subset of the grammar.

*Subroutines*

Because the primary operations are operations on trees, the main subroutines of the SYNN package analyze and manipulate trees. Three of the subroutines treat trees without reference to the grammar: CONTRE reads in a tree and converts it to the internal format, TRCPY stores a copy of a tree for later comparison, and TREQ compares two trees to see if they are identical.

In the SYNN package there are four subroutines that deal with phrase-structure grammars. CONCSG and CONCFG read in context-sensitive and context-free grammars, respectively, and convert them to internal format. CHQCS and CHQCF check the current tree against the indicated grammar by a regeneration procedure.

Most of the subroutines of SYNN are concerned with the transformational components. Separate subroutines read in the transformational rules, control the application cycle, mark levels of embedded subtrees, search for an analysis, check restrictions, and perform the operations.

The application of the forward rules is controlled by the subroutine APPFX, and the application of the reversal rules by APPBX. The application cycles are as described in the section Reversal of Transformational Rules, above, except that each transformational rule has a keyword which is used to bypass the search if the transformational keyword does not occur in the tree.

There is also a generation subroutine GENSR which is best described as a "constrained-random" generator. Within constraints specified by the user the subroutine generates a pseudo-random base tree to which other subroutines of SYNN can be applied.

*Programs*

In initial tests of the grammar and procedure the most useful combination of subroutines was in the program SYN1, which goes from basic tree to surface tree and back to basic tree, checking at every step. This first program is an iteration of the subroutines CONTRE, TRCPY, CHQCS, APPFX, CHQCF, APPBX, CHQCS, TREQ. When all parts are correct, the final result is the same as the input, and this is indicated by the final comment of the TREQ subroutine.

The program SYN2 carries out the steps of the MITRE procedure without the first two steps, lexical look-up and context-free parsing. Its basic cycle is CONTRE, TRCPY, APPBX, CHQCS, APPFX, CHQCF, TREQ. After each of the subroutines an indicator is checked to see if the tree should be rejected.

The program SYN3, which uses the generation subroutine, is like SYN2 except that GENSR replaces CONTRE in the basic cycle. Inputs for GENSR are easier to prepare than those of

CONTRE, so that SYN3 is being used extensively in debugging the grammar.

The lexical lock-up and context-free parsing steps of the procedure have not been programmed. Because algorithms for these steps are known to exist, it was decided that their programming could be postponed and an existing program used.

*Test Grammar*

A subset of the grammar, familiarly known as the JUNIOR grammar, was selected for initial tests of the procedure. Its dimensions are:

| | | |
|---|---|---|
| *(Forward) Grammar* | | |
| Phrase-Structure Component: | 61 | rules |
| | 105 | subrules |
| Transformational Component: | 11 | initial singularies |
| | 6 | embeddings and related singularies, including two embeddings |
| | 3 | final singularies |
| | 20 | rules |
| *Surface Grammar* | 32 | rules |
| | 306 | subrules |
| *Reversal Rules* | 6 | final singularies |
| | 15 | embeddings and related singularies |
| | 11 | initial singularies |
| | 32 | rules |

Twenty-six sentences (plus some variants) constitute a basic test sample for the JUNIOR grammar. This sample, which includes at least one test for each transformational rule, contains (among others) the sentences:

1. The airplane has landed.
2. Amphibious airplanes can land in water.
3. Did the truck deliver fifty doughnuts at nine hundred hours?
4. Were seven linguists trained by a young programmer for three months?
5. The general that Johnson met in Washington had traveled eight thousand miles.
6. Are there unicorns?
7. John met the man that married Susan in Vienna.
8. There were seven young linguists at MITRE for three months.
9. Can all of the ambiguous sentences be analyzed by the program?
10. The linguist the ambiguous grammar was written by is young.

SYN1 has been run on the full set of sample sentences. The total time for a run with 28 sentences was 5.11 minutes on the 7030 computer.

SYN3 has likewise been run with the JUNIOR grammar. As an example of running time, a typical run generating 20 trees carried all of them through the transformations and reversal rules in a total of 5 minutes. All but one of these trees contained embedded sentences; half of them contained two embeddings.

In another experiment, a CF parser was used with SYN2 to simulate the full procedure. The results for sentences (1), (2), and (6) are:

| | Sentence | | |
|---|---|---|---|
| | (1) | (2) | (6) |
| Pre-trees | 12 | 90 | 1 |
| Presumable surface trees | 8 | 15 | 1 |
| Presumable base trees | 3 | 4 | 1 |
| Correct base trees | 1 | 2 | 1 |

In the worst case encountered, sentence (5), there are 48 presumable surface trees.

It is clear from even these few numbers that if the procedure is to be practical, it will be necessary to incorporate a highly efficient routine for obtaining surface trees and to work on the rapid elimination of spurious ones.

## REFERENCES

1. "English Preprocessor Manual," SR-132, MITRE Corp. 1964, rev. 1965.

2. N. Chomsky, *Syntactic Structures,* Mouton, The Hague, 1957.

3. ———, *Aspects of the Theory of Syntax,* M.I.T. Press, Cambridge, Mass., 1965.

4. ———, "Formal Properties of Grammars," *Handbook of Mathematical Psychology,* R. D. Luce, R. R. Bush and E. Galanter, eds., Wiley, New York, 1963, vol. 2, pp. 323-418.

5. ——— and G. A. Miller, "Introduction to the Formal Analysis of Natural Languages," ibid., pp. 269-321.

6. ——— and ———, "Finitary Models of Language Users," ibid., pp. 419-491.

7. T. V. Griffiths and S. R. Petrick, "On the Relative Efficiencies of Context-Free Grammar Recognizers," *Comm. ACM,* vol. 8, pp. 289-300 (1965).

8. T. V. Griffiths, "Turing machine recognizers for general rewriting systems. *IEEE Symp. Switching Circuit Theory and Logical Design.* Princeton, N. J., November, 1965, pp. 47-56.

9. G. H. Matthews, "Analysis by Synthesis of Sentences of Natural Languages," *1961 International Conference on Machine Translation and Applied Language Analysis,* HM Stationery Office, London, 1962, vol. 2, pp. 531-540.

10. S. R. Petrick, "A Recognition Procedure for Transformational Grammars," Ph.D. thesis, M.I.T., 1965.

11. S. Kuno, "A System for Transformational Analysis," paper presented at the 1965 International Conference on Computational Linguistics, New York City, May 20, 1965.

# ERRATA

Page 318
Column 1, line 10

Insert are

analysis algorithms for CS grammars are given by Griffiths (1964). [8]

Column 2, line 30

Read or

standard or self-explanatory.

Page 319
Column 2, line 4

Read although

although there is in the more informal notation.

Page 320
Column 1, Footnote 2, line 1

Delete of

There is also a fourth set, conjunction rules. Because __ the treatment, etc.

Column 2, line 12

Read instances

one lowest sentence and no instances of #S# that are

Column 2, line 40

Read questions

clauses, yes-no questions, and WH-questions. The

Column 2 (bottom) and
Page 321, Column 1 (top)

For correct format of listing, read:

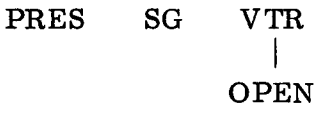
| | |
|---|---|
| Phase Structure Component: | 75 rules<br>approximately 275 subrules |
| Transformational Component: | 13 initial singularies<br>26 embeddings and related singularies, including 9 embeddings<br>15 final singularies<br>54 rules |

ERRATA (cont.)

Page 321

Column 1, Para. 4, line 8

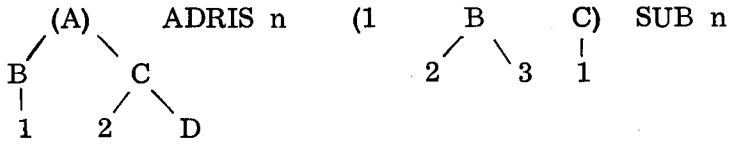
For correct representation of OPEN read:




Page 322

Column 1, item (b)

For proper representation of displayed items read:




Column 2, line 26

| | |
|---|---|
| Correct heading format for Checking by Synthesis | Identical in type face and placement to heading shown in line 18: Checking of Presumable Basic Trees |

Page 323

Column 2, after line 12

| | |
|---|---|
| For major heading above the secondary heading, Analysis by Synthesis, insert: | OTHER APPROACHES TO THE ANALYSIS PROBLEM (identical in type and placement to the major heading, AREAS FOR FURTHER INVESTIGATION, Page 323, Column 1) |

Page 325

Column 1, line 3

| | |
|---|---|
| Read look-up | The lexical look-up and context-free parsing |

ERRATA (cont.)

Page 325 (cont.)
Column 1, displayed listing

For correct format of listing, read:

(Forward) Grammar

| | |
|---|---|
| Phase Structure Component: | 61 rules |
| | 105 subrules |
| Transformational Component: | 11 initial singularies |
| | 6 embeddings and related singularies, including 2 embeddings |
| | 3 final singularies |
| | 20 rules |
| Surface Grammar | 32 rules |
| | 306 subrules |
| Reversal Rules | 6 final singularies |
| | 15 embeddings and related singularies |
| | 11 initial singularies |
| | 32 rules |

Page 326
Column 1, Ref. 8, line 2

| | |
|---|---|
| Replace period with comma; insert closing quotation marks | for general rewriting systems," IEEE Symp. Switching |