CS-80

DIRECTED RANDOM GENERATION OF SENTENCES

by

JOYCE FRIEDMAN

This research was supported in part by the United States Air Force Electronic Systems Division, under Contract F19628-C-0035.

STANFORD UNIVERSITY COMPUTER SCIENCE-DEPARTMENT

COMPUTATIONAL LINGUISTICS PROJECT

OCTOBER 1967

DIRECTED RANDOM GENERATION OF SENTENCES

by

Joyce Friedman

Abstract

The problem of producing sentences of a transformational grammar by using a random generator to create phrase structure trees for input to the lexical insertion and transformational phases is discussed. A purely random generator will produce base trees which will be blocked by the transformations, and which are frequently too long to be of practical interest. A--solution is offered in the form of a computer program which allows the user to constrain and direct the generation by the simple but powerful device of restricted subtrees. The program is a directed random generator which accepts as input a sub-tree with restrictions and produces around it a tree which satisfies the restrictions and is ready for the next phase of the grammar. The underlying linguistic model is that of Noam Chomsky, as presented in Aspects of the Theory of Syntax. The program is written in Fortran IV for the IBM 360/67 and is part of the Stanford Transformational Grammar Testing System, It is currently being used with several partial grammars of English,

I. INTRODUCTION

1. Motivation

In Aspects of the Theory of Syntax [1], Noam Chomsky presents a model of transformational grammar in which linguistic insights about a natural language are expressed in a precise formalism. Writing such a grammar for even a small part of a natural language is intrinsically difficult because of the complex interrelationships of the phrase structure rules, the lexicon, and the transformations. In addition, global decisions, such as ordering, or cycling conventions, or conventions on the meaning of the notation, may likewise have unexpected effects. If, in the course of writing a grammar, a global decision is changed, this may have repercussions in the already completed parts of the grammar.

These formal problems of grammar writing are likely to be regarded as secondary by the linguist, who is first concerned with what is in the language and how it is derived, and would prefer to pay less attention to formal detail. Yet, if the grammar does not produce the derivations intended, the linguist cannot be said to have succeeded,

This is a situation in which it seems natural to use a computer: the model is formal; the problem lies in the mass of detail, Most of the errors can be fixed if only they are brought to the attention of the linguist.

2. Directed random generation

One valuable way to provide feedback to the linguist is by exercising a grammar as a generator. If interesting base trees can be produced, from the phrase structure rules, they can be used to test the grammar as a whole. However, the use of base trees generated purely at random has certain major difficulties. If the phrase structure rules are recursive, the derivation may fail to terminate. An even more serious problem is that for a transformational grammar the relation between embedding and embedded subtrees is a special one, and trees generated at random will block in the transformation phase.

Even if these difficulties could be somehow bypassed, it would be desirable to have some control over the generator. At any given time some types of trees are of more interest than others; we may wish to test some particular set of transformations, or to study trees with a particular subtree, and so on. We would like both to constrain the generation away from the pitfalls of infinite length and of blocking and to direct it toward areas of interest.

The solution offered here is a directed random generator, which is as random as you like, but not more so. The user gives the program a rough description of the trees desired, and the program then fills in the rest of the tree using phrase structure rules selected at random. If the input is the sentence symbol only, then the output is a random tree (without embedding). However, if more detailed directions are given, they will, if consistent with the grammar, be followed,

The description given to the generator is in the form of a "skeleton" which is to be a subtree of the result and which may contain directions

governing the generation. These include restrictions of dominance, nondominance, and equality, and some special variables, Tree size is controlled by allowing recursion on. the sentence symbol only if specified in the restricted skeleton.

This use of a restricted skeleton to direct the generation is the novel and distinguishing feature of the program. The purpose of the generation routine is to provide tests for the grammar as a whole, including the lexical and transformational parts. The use of restricted skeletons makes it possible to generate trees which will undergo a specific transformation. In testing the program it was found that several tries are occasionally necessary to find the right skeleton for a particular transformation. Once found, however, the skeleton can continue to be used even though other transformations of the grammar are modified.

3. Historical remarks

Yngve's random generation program. The first program to generate sentences at random from a grammar was the well-known COMIT program of Yngve [7], who used random generation to test a small grammar for its adequacy to natural language. Since the grammar was a (discontinuous) phrase structure grammar the problems with respect to blocking did not arise. There was no need or desire to direct the sentences in any way; it was precisely their randomness which made them useful as a test of adequacy. Although the grammar did contain several types of recursion, the rules were such that excessively long sentences were highly

improbable,[1] and apparently Yngve was lucky.

The semantic generator of Sakai and Nagao. Sakai and Nagao [6] describe a program which uses a special form of controlled generation to produce "semantically correct? sentences. The generation is controlled by allowing it to start with an arbitrary grammatical category and a word to be dominated by it. The generation then works in both directions, up to the sentence symbol, and down from the category to the word, Sakai and Nagao use a transformational model, without complex symbols, but with a formalism which allows a lexical item to be associated with a higher non-lexical category. Their type of specification would be handled by the dominance restriction in our program.

Meyers and Yang, A brief report by Meyers and Yang [3] from Ohio State University indicates that an attempt was made to use a random generation program to test a transformational grammar,, They report that "it is seldom possible to generate two sentences at random such that one can be embedded into the other".

MITRE generation program. A random generation program written by the present author was used in testing the grammars whichwere part of the MITRE syntactic analysis procedure [8]. This program was a

[1] For example, the node adjectives has only probability $1/2^n$ of expanding into n+1 adjectives. This is in sharp contrast with, say, the adjective phrase rule AP → {AP AND AP (AND AP)*, (DEG) ADJ (S) (ADV)} [5], which, if all choices are taken with equal probability, will almost never terminate,

first attempt at solving the problems which are solved by the present program. A device was included which made it possible to generate trees which underwent the transformation that embedded relative clauses. However, the device was ad hoc and did not extend to other embedding transformations. There were other minor disadvantages which have also been eliminated in the new program,

Conaale-controlled grammar testers. The programs mentioned so far, and the program to be discussed in this paper, are all non-interactive programs, An alternative approach is to allow the grammar tester to be controlled by the user from a console. Such on-line grammar testers are being written by Louis Gross at the MITRE Corporation and David Londe at Systems Development Corporation.

4. Stanford Transformational Grammar Testing System

The Stanford Transformational Grammar Testing System, of which the generation program is a part, includes facilities for dealing with all the components of a transformational grammar. The System includes programs for phrase structure, for transformations, and for complex symbols and lexicons. The phrase structure programs include input programs which accept the usual compact linguistic form, and also a parsing program, Feature-handling programs accept features and complex symbols, and compare, expand, and modify complex symbols. A lexical insertion program is now being written, The transformational programs will include an analysis program (also used in lexical insertion) and programs to accept transformations and cycling rules, and to transform a base tree into a surface tree.

The generation program uses other programs of the system, in particular the input and output programs for phrase structure grammars and for trees. The trees produced-by the generation program can be fed directly into the lexical insertion program and thence into the programs which apply transformations to obtain sentences, or they can be punched on cards for later input.

II. PROGRAM DESCRIPTION

1. Generation algorithm

The generation routine GEN requires as basic input an ordered context-free grammar. For each set of trees to be generated, a "restricted skeleton" is also input, These inputs will be described in detail in a later section,

A basic skeleton is a tree and has the sentence symbol as root, It may contain any of the symbols of the grammar, and also special variable node symbols,. A restricted skeleton is a basic skeleton which may also contain restrictions in terms of dominance, nondominance and equality,

The skeleton is expanded by a process which begins with a current string consisting of only a sentence symbol. (The generation process is illustrated in Figure 1 using the grammar given in Figure 2.) The program cycles through the rules of the grammar, and for each rule searches from left to right through the current string for the next symbol to be expanded by the rule. If none is found it proceeds to the next rule.

When all rules are done, they are recycled if' and only if there are unexpanded sentence symbols in the current string. If there are, the current string will be revised to contain only the sentence symbols and the program will begin again with the first rule.

In expanding a symbol the set of possible expansions given by the grammar is first reduced by eliminating expansions incompatible with the basic skeleton and with the special symbols ^{X}, ^{Y}, and NL. X and Y are variables over 0 or more and 1 or more nodes.

NL is a null node. No new sentence symbol will be introduced unless explicitly specified either as part of the skeleton or in a dominance restriction,

From the expansions which are consistent with the basic skeleton, the program eliminates those which cannot meet some restriction on that node in the skeleton. Then an expansion is chosen at random from those which are left, the tree is expanded, and the current string is updated,

The dominance and nondominance restrictions are now reconsidered, If a dominance restriction has now been satisfied, or will now inevitably be satisfied,. it is dropped; otherwise it is moved to a node selected at random from those which can later satisfy it. A nondominance restriction NDOM M is dropped when no further occurrence of M is possible ; otherwise , copies of the restriction are attached to all new nodes which might dominate M . An important exception is that the dominance and nondominance restrictions are never moved down over a sentence symbol,

Treatment of the dominance andnondominance restrictions is facilitated by the use of a special subroutine which determines whether a symbol M must, may, or cannot dominate a symbol N .

The equality restrictions are handled by expanding only the first node encountered with a particular equality restriction EQ i . Expansion of other nodes with EQ i is not carried out, but a marker is attached, Then, as the final stage in generation, copies of the first expansion are filled in for the other equal nodes,

An example of a skeleton with an equality restriction and a possible output tree for that skeleton are given in Figure 3.

2. Restricted skeletons

We now describe restricted skeletons more completely, beginning with some underlying notions

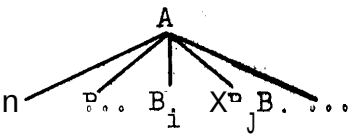
Basic skeleton

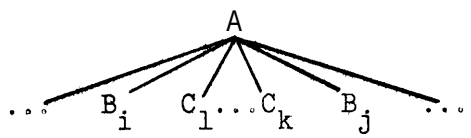
Definition, A basic skeleton (for a grammar G) is a subtree of some tree of G . It further satisfies the condition that the daughters of any node in the basic skeleton are adjacent in some tree of G . That is, if the daughters of a node A of the skeleton axe (from left to right) $B_1, B_2, \ldots, B_n$ $(n \geq 1)$, then G must contain a rule $A \rightarrow C_1 \ldots C_k B_1 \ldots B_n D_1 \ldots D_r$ $(k,r \geq 0)$.

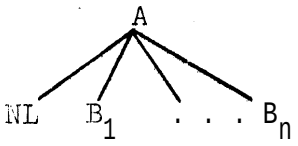
The basic skeleton will be a subtree of the generated tree. Although the basic skeleton is not the most general subtree one might consider, the cases excluded can be included by the use of the special symbols

Special symbols

Definition X , Y and NL are special symbols. X and Y are variables over 0 or more and 1 or more nodes; NL is the null symbol.

The skeleton  will be expanded as

 where $k \geq 0$ and $A \rightarrow \ldots B_i C_1 \ldots C_k B_j \ldots$ is a phrase structure rule. If occurs in place of X , then $k > 0$. NL is meaningful only when used as the left-most or right-most daughter

of a node. 

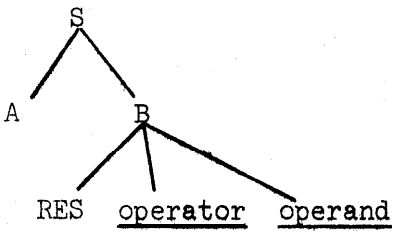
will result in node A having B_1 as its left-most daughter, whereas without NL additional daughters might be introduced to the left of B_1 .

The special symbols are used as terminal nodes of the skeleton and increase the expressive power of the basic skeleton. X and Y can be used to make the basic skeleton more general; NL is used to prevent expansions to the left or right of the explicitly-given expansion. Figure 4 illustrates the use of the special symbols.

Restriction

Definition. A restriction consists of RES , followed by an operator DOM , NDOM or EQ and its operand, The operators and their operands are described in the next section. A restriction is attached as daughters of the node to which it applies.

Example.



Restricted skeleton

Definition. A restricted skeleton is a basic skeleton which may have as node names not only symbol:; of the grammar, but also restrictions, and ape cial symbols.

Restrictions on a skeleton

Three types of restrictions may be used in specifying a skeleton: dominance, nondominance, and equality. Dominance is useful in trying to get a tree which will undergo a specific transformation. Nondominance is also used in that way, but is particularly useful in avoiding the lengthy trees which would otherwise be generated by such rules as NP → NP AND NP . Equality is an essential requirement for many transformations, most particularly for embedding transformations.

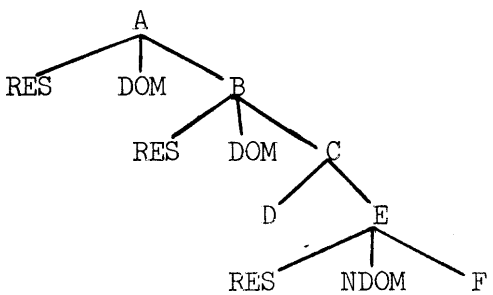
Form of restrictions

A restriction is input as three consecutive daughters of the restricted node. The restrictions are RES DOM A , RES NDOM A , and RES EQ i , where A is a symbol of the grammar and i is an integer ($1 \leq i \leq 20$) . A node may have more than one restriction.

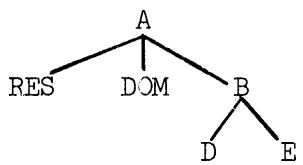
Dominance restriction, DOM

If a node A bears the restriction RES DOM B , then in the resulting tree, node A will dominate at least one node B . The condition will be satisfied without introducing any intervening sentence symbol.

The restriction RES DOM B may be extended by allowing B to be the root of a restricted skeleton. Thus, among the possibilities are



and



Nondominance restriction, NDOM

If A is restricted by RES NDOM B , no node B will be dominated by A (unless a sentence symbol comes between them).

Equality restriction, EQ

The equality restriction is used to cause two or more nodes to dominate identical subtrees. For any i , all nodes with the restriction RES EQ i will have identical subtrees. The program actually expands only the first such node encountered and copies the result for other similarly restricted nodes. As a consequence of this treatment, only this node can have a partial expansion or have additional restrictions

3. Phrase Structure Grammar

The program requires that the underlying phrase structure grammar be ordered and context-free, In deciding to use ordered rather than unordered grammars, we were following our interpretation of the model in Aspects, Implementation of an alternative model with unordered rules would of course be possible, but would result in a slower and less elegant program, particularly in the treatment of the dominance and nondominance restrictions,

The ordering constraint is:

The rules are ordered linearly. Each nonterminal symbol is expanded by precisely one rule (which may give more than one alternative expansion), Except for the sentence symbol, no nonterminal is reintroduced after the rule which expands it,

In Aspects (p. 137) Chomsky expresses the belief that recursiveness in the base component is limited to rules which reintroduce the sentence symbol. The ordering restriction above is in fact less restrictive, since it allows other recursion in the form A → ... A ... . It does, however, exclude loops of the form A $\overset{*}{\rightarrow}$ . . . B . , , , B $\overset{*}{\rightarrow}$ ... A ... .

The decision to use context-free rather than context-sensitive grammars also follows Aspects. In unrestricted generation the use of a context-sensitive grammar introduces no serious problems. However, it would vastly complicate the handling of our restrictions, particularly the equality restriction.

4. Formats

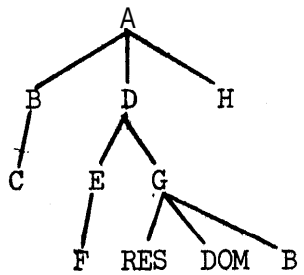
The generation program uses other programs of the System for all its input and output requirements, These programs will not be described here, but we will give the formats.

Formats for trees

The fixed-field input for trees (and for restricted skeletons) is used in the example of Figure 3. It is essentially a mirror reflection of the tree, followed by a deformation which puts the first daughter of a node in the same row (card) as the node, and puts all daughters of a node in the same column (field). The daughters of a node in field L appear in field L+1 . The first (left-most) daughter is in the same card as its parent, Daughters to the right appear on later cards. Thus

```
A     B     C
      D     E     F
            G . . RES
                  DOM
                  B
      H
```

represents the tree



A tree begins with a title card (which may be blank) and is ended by a blank card.

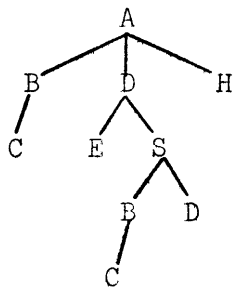
Substitution feature for trees

A potential difficulty in the basic format is that the depth of a tree may exceed the maximum number of fields allowed. A substitution feature avoids this by replacement of a dummy node by a subtree. This is indicated by the use of a substitution card with XXX in the first field and the dummy node in the second, Thus, the input

EXAMPLE

```
A     B     C
      D     E..   F
            G
      H
XXX   G
S     B     C
      D
   (blank)
```

represents the tree



The output format for trees is essentially the same as the input format, An option allows the output to be punched on cards which are then acceptable as input. Another output option displays the internal node numbers for the trees.

The System also has an alternative of free-field tree input [2].

Input format for phrase structure grammar

The phrase structure grammar is input in a compact parenthesized form. It is described syntactically by the following B.N.F. grammar:

<phrase structure> ::= <rule> . | <phrase structure> <rule> .

<rule> ::= <word> = <RHS>

<RHS> ::= <node list> | <RHS> , <node list>

<node list> : := <node> | <node> <node list>

<node> : := <word> | (<RHS>)

This corresponds to common linguistic use except that curly brackets have been replaced by parentheses, and items which would be displayed on different lines within brackets are now separated by commas.

Example:

$$VP \rightarrow \left(\begin{array}{l} AUX \left\{ \begin{array}{l} MV\ (NP) \\ COP\ (\left\{ \begin{array}{l} NP \\ AP \end{array} \right\}) \end{array} \right\} \\ S \end{array} \right) (ADV)$$

VP = (AUX(MV(NP),COP((NP,AP))),S)(ADV).

Figure 2 shows a phrase structure grammar as input, followed by the expanded form produced by the input routine.

5. Computer considerations

GEN is written in FORTRAN IV H and currently runs under OS on Stanford's IBM 360/67. Table I shows the subroutine structure in a run of GEN.

Table I

GEN and its subroutines

Generation routine

Test for dominance

System programs used by GEN

Phrase Structure grammar input

Phrase structure grammar output

Free-field output

Free-field read

Tree input (used for skeletons)

Tree output

Elementary operations on trees

Running time

The running time for the program is approximately .16 seconds per generated tree.

6. Final remarks

The current version of the generation program has been used both with our own phrase structure grammar, OLAG [4], and with drafts of UCLAG, a grammar at U.C.L.A. [5]. We believe that it could be useful to other transformational linguists, and welcome inquiries.

The Grammar Testing System is being extended by a lexical insertion model which will be an implementation of one of the models suggested in

Aspects. The generation program will then accept complex symbols and restrictions containing complex symbols.

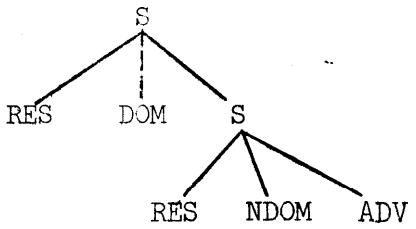
ACKNOWLEDGEMENT

John H. Gilman programmed the addition of restricted skeletons to a purely random generation routine, Thomas H. Bredt improved the code, and extended it to allow the dominance restriction to govern a restricted skeleton. The program used for input of the phrase structure rules was designed and written by Robert W. Doran.

FIGURE 1

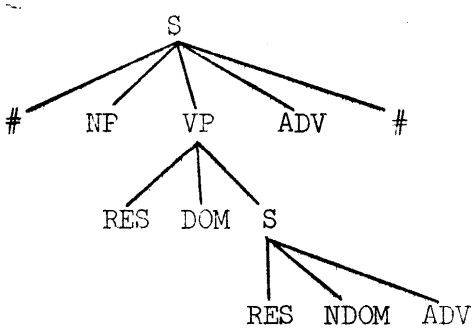
Steps in the Generation of a Tree

Skeleton



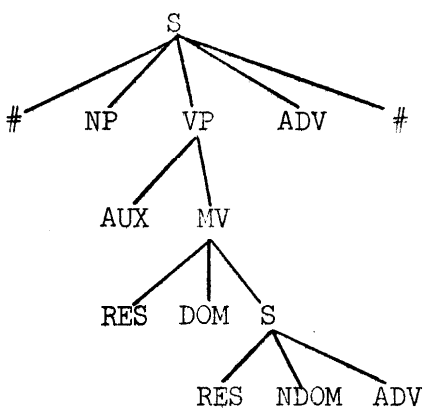
Current string: S

Expansion of top S. The restriction is moved to a random choice among NP, VP and ADV.



Current string: # NP VP ADV #

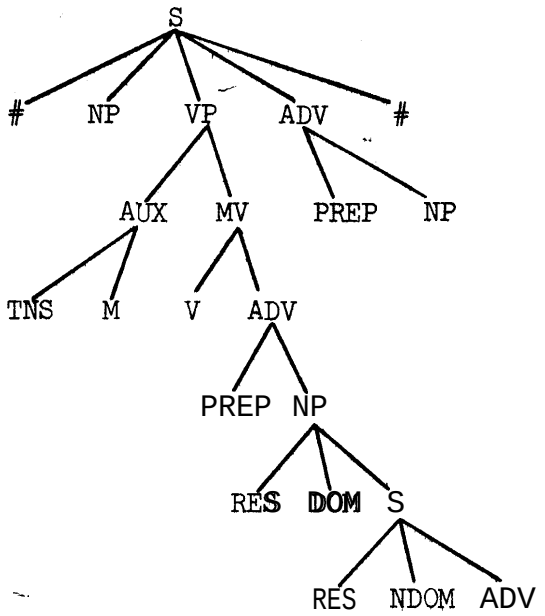
Expansion of VP. The restriction is moved to MV.



Current string: # NP AUX MV ADV #

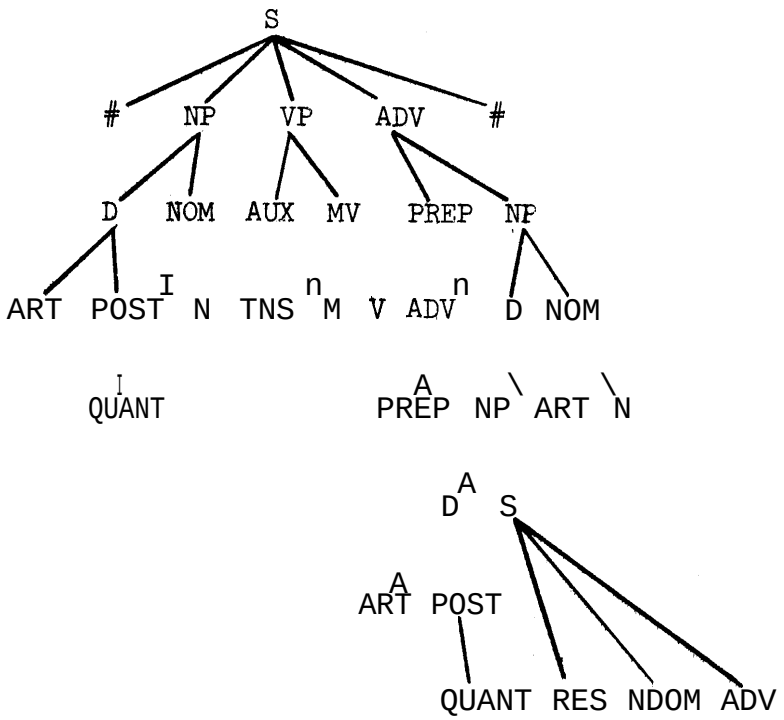
(Figure 1 continued)

Expansion of MV, AUX, and ADV's. Choices are random, but do not introduce S. The restriction continues to move down.



Current string: # NP TNS M V PREP NP PREP NP #

Expansion of rest of main S. The restriction is dropped when satisfied,



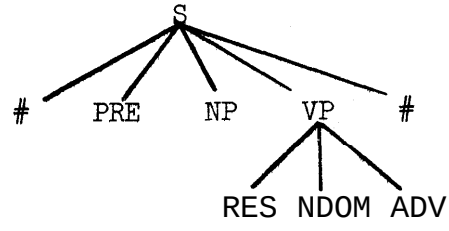
Current string: # ART QUANT N TNS M V PREP ART QUANT S PREP ART N #

(Figure 1 continued)

After main S is completed.

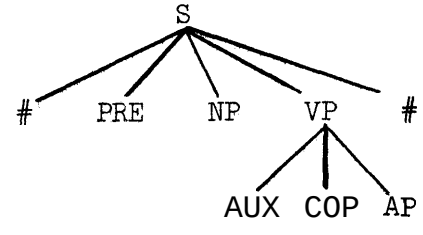
Current string: S

Expansion of embedded S. Restriction moved to the VP.



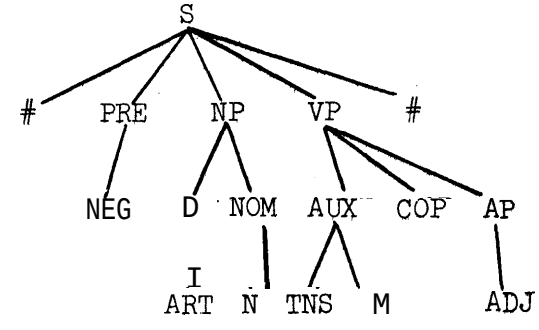
Current string: # PRE NP VP #

Expansion of VP. Restriction is no longer needed, since AUX, COP, AP cannot dominate ADV.



Current string: # PRE NP AUX COP AP #

Embedded subtree completed.



Current string: # NEG ART N TNS M COP ADJ #

FIGURE 2

Phrase-Structure Input

```
     "REDUCED VERSION OF UCLAG 15 SEPT 67"
S = # (S CONJ S(CONJ S),(PRE) N-P VP (ADV)) # .
VP = (AUX(MV(NP),COP((NP,AP))),S)(ADV) .
MV = V(ADV) .
AP = ADJ.
AUX = TNS (M(IMP)) (ASP) .
ADV = (PREP NP, ADVB).
NP = D (NOM,S).
NOM = ((NOM) S, N).
D = ART (POST).
PRE = NEG.
CGNJ = ((WH) OR, BUT, AND),
ASP = (PERF)(PROG).
POST = QUANT.
IMP = (NOT)(PLEASE) IMPER.
$END
```

as input

```
 1    S =
 2    #S CONJ S COW S#,
 3    #S CONJ S#,
 4    #PRE NP VP ADV#,
 5    #PRE NP VP#,
 6    #NP VP ADV#,
 7    #NP VP#.
 8    VP =
 9    S ADV,
10    S,
11    AUX MV NP ADV,
12    AUX MV NP,
13    AUX MV ADV
14    AUX MV,
15    AUX COP N-E' ADV,
16    AUX COP NP,
17    AUX COP AP ADV,
18    AUX COP AP,
19    AUX COP ADV,
20    AUX COP,
21    MV =
22    V ADV,
23    V.
24    AP =
25    ADJ.
26    AUX=
27    TNS M IMP ASP,
28    TNS M IMP,
29    TNS M ASP,
30    TNS M,
31    TNS ASP,
32    TNS.
```

expanded form produced by input routine

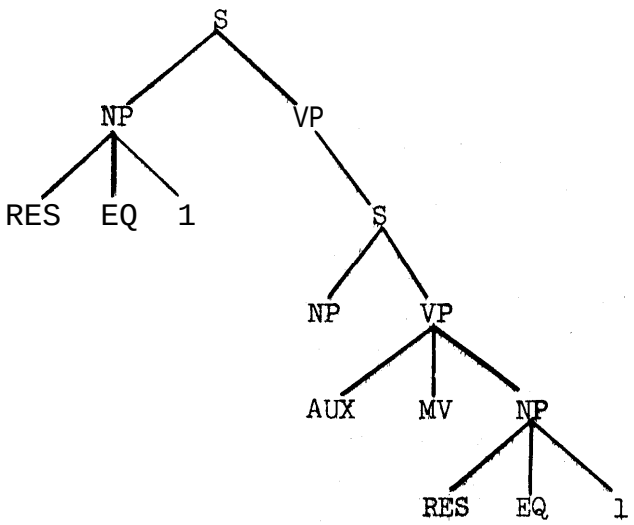
(Figure 2 continued)

```
33    ADV =
34    P-REP NP,
35    ADVB.
36    NP =
37    D S,
38    D NOM.
39    NOM =
40    S,
41    NOM S,
42    N.
43    D =
44    ART POST,
45    ART.
46    PRE =
47    NEG.
48    CONJ =
49    WH OR,
50    OR,
51    BUT,
52    AND,
53    ASP =
54    PROG ,
55    PERF PROG,
56    PERF.
57    POST =
58    QUANT.
59    IMP =
60    PLEASE IMPER,
61    NCT PLEASE IMPER,
62    NOT IMPER,
63    IMPER .
```

FIGURE 3

A Skeleton and An Output Tree

Skeleton

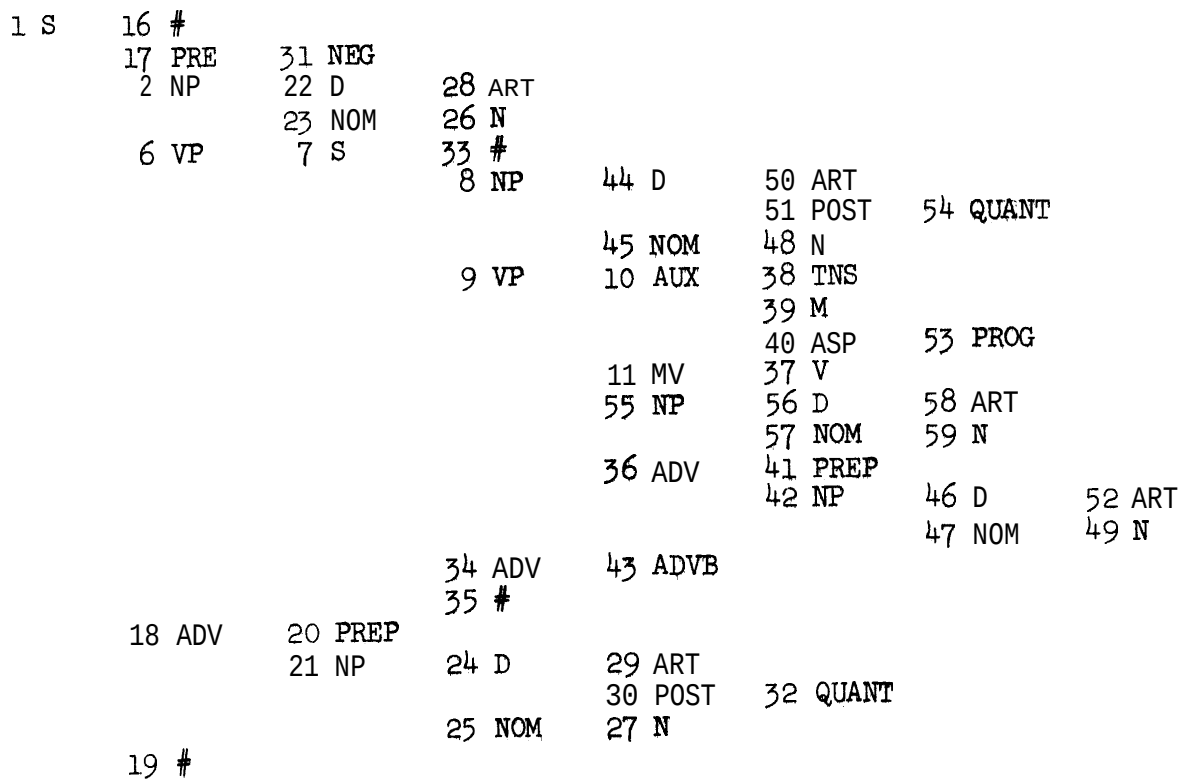


Skeleton as Input

```
S     NP   RES
           EQ
           1
      VP   S    NP
                VP   AUX
                     Mv
                     NP   RES
                          EQ
                          1
```

(Figure 3 continued)

Generated Tree (Node numbers reflect the order of generation)

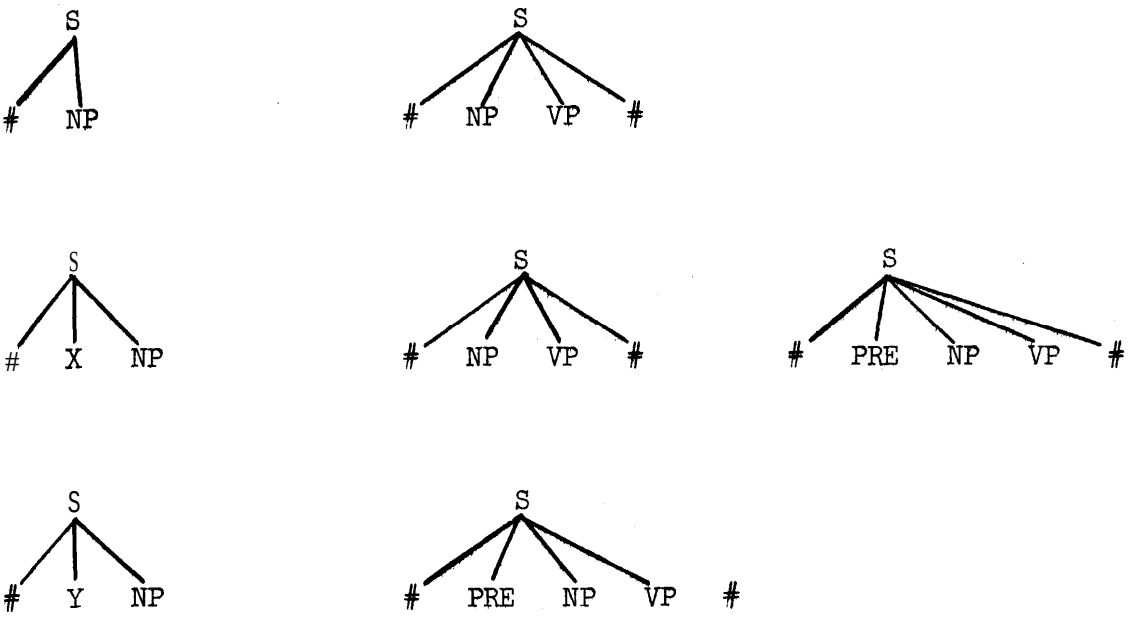


#	NEG	ART	N	#	ART	QUANT	N	TNS	M	PROG	V	
ART	N	PREP	ART	N	ADVB	#	PREP	ART	QUANT	N	#	

FIGURE 4

Use of Special Node Symbols

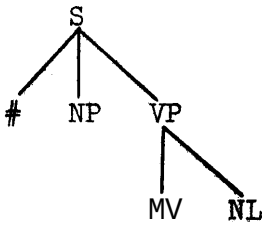
Rule: S = # (S CONJ S(CONJ S),(PRE) NP VP) # .



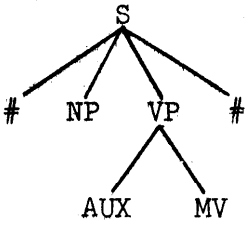
(Figure 4 continued)

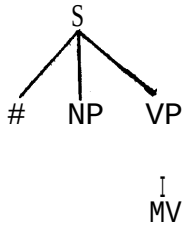
Rule: VP = (AUX(MV(NP), COP((NP,AP))), S)(ADV).

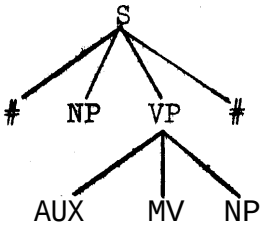
Skeleton

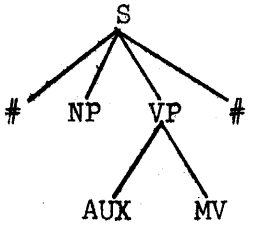


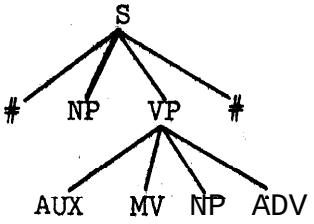
Possible Expansions

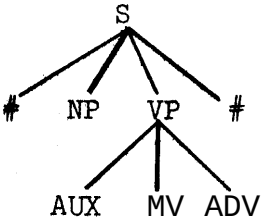












REFERENCES

[1] Noam Chomsky, Aspects of the Theory of Syntax, M.I.T. Press, Cambridge, 1965.

[2] R. W. Doran, 360 0. S. Fortran IV free field input/output subroutine package, Report AF-14 of the Computational Linguistics Project, Stanford University Computer Science Department, Oatober 1967.

[3] L. F. Meyers and J. Yang, Chinese grammars and the computer at the Ohio State University, Project on Linguistic Analysis, RF 1685-3, Report No. 10, May 1965, pp. 28-37. Research Foundation, Ohio State University, Columbus, Ohio.

[4] Olasope 0. Oyelaran, AF test grammar, Report AF-13 of Computational Linguistics Project, Stanford University Computer Science Department, September 1967.

[5] B. H. Partee, P. Schachter, R. Stockwell, et. al., Working Papers, U.C.L.A.-Air Force English Syntax Conference, September 4-15, 1967 (multilithed).

[6] Toshiyuki Sakai and Makoto Nagao, Sentence generation by semantic concordance, International Conference on Computational Linguistics, 1965, (multilithed) 22 pp.

[7] Victor H. Yngve, Random generation of English sentences, Proc. 1961 Int'l Conf. on Machine Translation of Languages and Applied Language Analysis, Teddington, H.M.S.O., London, 1962, pp. 66-80.

[8] A. M. Zwicky, J. Friedman, B. C. Hall and D. E. Walker, The MITRE syntactic analysis procedure for transformational grammars, Proc. Fall Joint Computer Conference, 1965, pp. 317-326.