

AD 635520

SPECIFICATION AND UTILIZATION
OF A TRANSFORMATIONAL GRAMMAR

D. Lieberman

INTERNATIONAL BUSINESS MACHINES CORPORATION
Thomas J. Watson Research Center
Post Office Box 218
Yorktown Heights, New York 10598

Contract No. AF19(628)-5127

Project 4641

Task 464102

SCIENTIFIC REPORT NO. 1

Date of Report March 1966

CLEARINGHOUSE FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION			
Hardcopy	Microfiche		
\$ 7.00	\$ 1.50	305	42
ARCHIVE COPY			

Prepared
for

DDC
RECEIVED
JUL 22 1966
C-1150

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS

**SPECIFICATION AND UTILIZATION
OF A TRANSFORMATIONAL GRAMMAR**

D. Lieberman

**INTERNATIONAL BUSINESS MACHINES CORPORATION
Thomas J. Watson Research Center
Post Office Box 218
Yorktown Heights, New York 10598**

Contract No. AF19(628)-5127

Project 4641

Task 464102

SCIENTIFIC REPORT NO. 1

Date of Report March 1966

**Distribution of this document
is unlimited.**

**Prepared
for**

**AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
OFFICE OF AEROSPACE RESEARCH
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS**

Preface

The objective of this project titled "Specification and Utilization of a Transformational Grammar" is the investigation of the applicability of transformational theory as a theoretical support for linguistic aspects of language data processing. As the first step in our research program, we are developing a transformational grammar of English, including a large lexicon, and computer programs to aid in the grammatical and lexicographic work. Our approach and achievements thus far are presented herein in four parts:

Part I, "The IBM Core Grammar of English"

by P. Rosenbaum and D. Lochak

Part II, "Design of a Grammar Tester"

by D. Lieberman

Part III, "Programming of the Grammar Tester"

by F. Blair

Part IV, "Computer Support for Lexicon Development and Use"

by D. Lieberman and D. Lochak

Professors E. Klima and G. H. Matthews served as consultants to the project.

D. Lieberman
Principal Investigator

Abstract

Scientific Report No. 1 contains four parts:

- Part I - The IBM Core Grammar of English. Our current grammar of English is presented in full, and numerous derivations are carried out in detail to illustrate the current generative power of the grammar.
- Part II - Design of a Grammar Tester. The design considerations on which the present version of the tester was based are discussed, and a set of tentative input, output, and control formats are presented.
- Part III - Programming for the Grammar Tester. A LISP implementation of the grammar tester is presented. The overall flow of control and the various special functions are described.
- Part IV - Computer Support for Lexicon Development. A program package (programmed in SNOBOL) to facilitate the compilation, modification, scanning, etc. of the lexicon is described.

PART I

"THE IBM CORE GRAMMAR OF ENGLISH"

P. Rosenbaum

D. Lochak

Table of Contents

	<u>Page</u>
1.0 Introduction	1
2.0 The Phrase Structure Component	1
2.1 Rewriting Rules	1
2.2 The Lexicon	9
3.0 The Transformational Component	17
3.1 Introduction	17
3.2 Transformational Rules	33
4.0 Derivations on the Core Grammar (CG)	52
4.1 Conventions	52
4.2 Sentence Types	53
4.3 Derivations	56
5.0 Notes	

THE IBM CORE GRAMMAR OF ENGLISH

by

Peter S. Rosenbaum
and Dorita Lochak

International Business Machines Corporation
Thomas J. Watson Research Center
Yorktown Heights, New York

Abstract

A transformational grammar is presented which characterizes sentence types exemplifying noun phrase complementation, verb phrase complementation, relative clause formation, two types of question construction, indirect object and prepositional phrase constructions, passive constructions, aspectual constructions, and certain types of negation phenomena.

The presentation consists of 1) a theoretical introduction explaining the version of transformational theory from which the IBM Core Grammar of English is constructed, 2) a presentation of the Core Grammar, and 3) a set of sixty-six derivations constructed in terms of the Core Grammar.

1.0 Introduction

The linguistic description discussed in this section, the IBM Core Grammar of English (henceforth referred to as the CG), conforms, in its essentials, to recent theoretical proposals.¹ The version of the transformation theory of syntactic descriptions underlying the CG provides two basic components, a phrase structure component and a transformational component. The former treats the generation of underlying constituent structures and the introduction of lexical material into these structures. The latter deals with the transformation of these underlying structures into derived constituent structures.

2.0 The Phrase Structure Component

2.1 Rewriting Rules

The phrase structure component of the CG contains an unordered set of rewriting rules of the general form $A \rightarrow \beta$, where A is a single nonnull symbol and β is a nonnull string of symbols which is distinct from and does not contain A . A considerably oversimplified set of phrase structure rewriting rules might be the following:

(1) Sentence $\rightarrow$ Noun Phrase + Verb Phrase

Verb Phrase $\rightarrow$ Verb + Noun Phrase

Noun Phrase $\rightarrow$ Determiner + Noun

Subrules, i. e., alternative expansions of the same symbol, are required to be abbreviated either by braces or by parentheses. For example, the rule given in (2) is an abbreviation for rules (3a) and (3b). If the rule (2) is required to apply, then the symbol tense must be expanded either as Present or as Past.

(2) Tense ----> $\left\{ \begin{array}{l} \text{Present} \\ \text{Past} \end{array} \right\}$

- (3) a. Tense ----> Present
 b. Tense ----> Past

Similarly, rule (4) is an abbreviation for the subrules (5a) and (5b).

- (4) Noun Phrase ----> (Determiner) + Noun
 (5) a. Noun Phrase ----> Determiner + Noun
 b. Noun Phrase ----> Noun

The sequential application of the rewriting rules, starting with an initial symbol, S, and continuing with each successive line formed by the application of one rule to one symbol in a string, results in the generation of a derivation. Symbols appearing on the right side of a rewriting rule are either terminal or non-terminal. Symbols appearing on the left side of a rewriting rule are only non-terminal. Glossaries of non-terminal and terminal symbols are provided in (6) and (7), respectively.

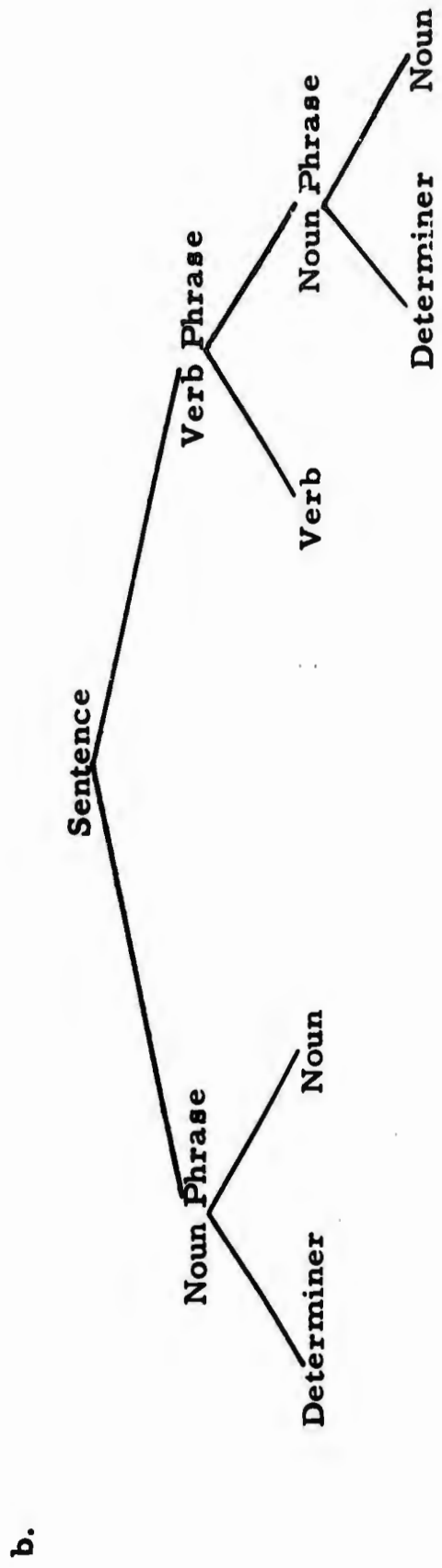
(6) S	Sentence
PRE	Presentence
NP	Noun Phrase
AUX	Auxiliary
VP	Verb Phrase
PP	Prepositional Phrase
MAN	Agentive Phrase
DET	Determiner
ART	Article
T	Tense

(7) NEG	Negative	
Q	Question	
PRES	Present	
PAST	Past	
M	Modal	
have	Have	} Aspectual morphemes
en	En	
be	Be	
ing	Ing	
V	Verb	
PREP	Preposition	
N	Noun	
ADJ	Adjective	
DEF	Definite	
INDEF	Indefinite	
WH	WH	
P	Passive	
#	Sentence Boundary	

The derivation of an underlying constituent structure through the application of the rewriting rules is terminated when the derivation contains no unexpanded non-terminal symbols, i. e., no generated non-terminal symbols remain to be expanded.

Particular derivations are identified by means of a uniquely associated labelled bracketing or tree diagram referred to as a P-marker. These alternative notations are illustrated for the rules in (1) as follows:

(8) a. $[[\text{Determiner Noun}]_{\text{Noun Phrase}} [\text{Verb} [\text{Determiner Noun}]_{\text{Noun Phrase}} \text{Verb Phrase}]_{\text{Sentence}}$



With respect to a P-marker, a substring of symbols which is uniquely traceable back to some node labelled X in a P-marker has the analysis X (is an X). For example, in (8) the substring Verb + Noun Phrase is a Verb Phrase. If A is a symbol in a string β which is itself an X, then A is dominated by X. Finally, X immediately dominates a symbol B if X dominates B and if X dominates no symbol C such that C dominates B.

The rewriting rules of the Core Grammar are as follows:

- (9) S ----> # (PRE) NP AUX VP #
- PRE ----> (NEG) (Q)
- AUX ----> T (M)
- T ----> $\begin{Bmatrix} \text{PRES} \\ \text{PAST} \end{Bmatrix}$
- VP ----> (have en)(be ing) $\left\{ \begin{array}{l} V (\begin{Bmatrix} \text{NP} \\ \text{PP} \end{Bmatrix}) (\begin{Bmatrix} \text{S} \\ \text{PP} \end{Bmatrix}) (\text{MAN}) \\ \text{be (ADJ)} \end{array} \right\}$
- PP ----> PREP NP
- MAN ----> PREP P
- NP ----> (DET) N (S)
- DET ----> ART (S)
- ART ----> (WH) $\begin{Bmatrix} \text{DEF} \\ \text{INDEF} \end{Bmatrix}$

Although the physical dimensions of the CG's rewriting rules are small, the system is extremely powerful. This power stems from the fact that the initial symbol, S, is recursive under the VP (Verb Phrase), under NP (Noun Phrase), and under DET (Determiner). The expansion of VP and NP is of particular interest. Two possible expansions of VP are given in (10) and (11). These expansions characterize intransitive Verb Phrase complementation and transitive Verb

Phrase complementation, respectively. In their fully derived form, these expansions correspond to the sentence complement constructions in (12) and (13).

(10) VP ----> V + S

(11) VP ----> V + NP + S

(12) John condescended to play ball with Sam

(13) John tempted Bill to play ball with Sam

Similarly, two distinct types of sentence complementation are derived through the expansions of NP given in (14) and (15).

(14) NP ----> DET + N + S

(15) NP ----> N + S

The expansion (14) might yield sentences like those in (16), depending upon the distribution of NP in the underlying structure.

(16) a. the fact that John came late bothers me

b. Bill pondered the fact that John came late

c. Harry worried about the fact that John came late

The expansion (15) is more interesting. When NP is dominated immediately by S, in one possible derivation, sentences of the type exemplified by (17) and (18) may result.

(17) it appears that John is honest

(18) John appears to be honest

When NP is dominated immediately by PP, sentences like (19) and (20) can result.

(19) I reminded John of Bill's having left

(20) I reminded John that Bill left

When NP is dominated immediately by VP, sentences like (21) and (22) may be generated.

(21) I believe that John is honest

(22) I believe John to be honest

The system of rewriting rules characterizes yet another instance of S recursion, namely, the expansion of DET into ART + S, which provides for the generation of relative clause structures, some examples of which are given in (23).

(23) a. the boy who saw me smiled

b. the boy I saw smiled

c. whatever you do bothers me

d. whichever article is selected will disintegrate

Beyond these important complex sentence constructions, the system provides for several of the syntactic functions which are usually seen to be fairly central to any adequate linguistic description. These include two types of questions, "yes-no" questions and NP questions as exemplified by the sentences in (24), indirect object and prepositional phrase constructions, passive sentences, sentences with aspectual properties, and negation constructions.

(24) a. did John leave

b. which tall boy did you see

The system of rewriting rules in the CG suffers from three fundamental gaps. First, the rules do not characterize anything approaching a complete Determiner system. Second, no Adverbial constructions of any kind are generated. Third, there is no provision for the description of the processes of conjunction and subordination. At present, we are experimenting

with several schemes for the description of the Determiner system. The results of these studies will most probably be included in an updated version of the CG to appear in approximately eight months. The same is true for the Adverbial system. We have given little attention to the problems of conjunction since theory is not sufficiently advanced to dictate the nature of the device which will offer an explanation of the complex array of facts comprising conjunction phenomena. Hopefully, theoretical insights with regard to conjunction will be forthcoming and research in the description of this process can be more profitably pursued.

The system of rewriting rules contains several questionable formulations. For one, it is becoming increasingly clear that the aspectual and auxiliary systems presented in the CG must be modified. We are, at present, considering the possibility of treating the aspectuals (have en) (be ing) as features of a predicate constituent VB, (Verbal), which will range over both Verbs and Adjectives, thereby eliminating the aspectuals as phrase structure constituents of the underlying structure. We have also been studying questions of greater theoretical import. First, we are interested in determining whether Modals (M) are most properly to be analyzed as Verbals and whether Tense (T) is to be considered a syntactic feature of Verbals. Such an analysis would eliminate AUX from the grammar. Similar considerations concerning the Negative marker (NEG), and the question marker (Q) are being studied. Second, the rewriting rules of the CG treat Present and Past, and DEF and INDEF, as actual constituents of T and ART, respectively. There are many reasons to believe that Present

and Past are not constituents of the non-terminal symbol T, but rather are syntactic features of a terminal symbol T. The same arguments which apply in this case (and which we do not discuss at present) hold for DEF and INDEF. Third, the description makes no clear-cut distinction between terminal symbols which are morphemes and terminal symbols which become the bearer of a complex symbol, i. e., a pair consisting of a stem morpheme given in terms of a distinctive feature matrix on the one hand and a vector of syntactic features determining the idiosyncratic properties of given morphemes. Much current research is devoted to theoretical questions of this sort.

For the practically inclined, it is germane to point out that the sentence types which the system will produce (invariably with less than ideal theoretical motivation) are numerous and of considerable diversity. It is our current research goal to increase the number of sentence types that the grammar can handle and to enlarge its scope in a manner consistent with theoretical developments.

2.2 The Lexicon

The lexicon of the CG consists of a list of lexical entries. Each entry consists of (1) a phonological distinctive feature matrix defining the underlying phonological properties of a given morpheme and (2) a syntactic feature vector (a complex symbol). It is the latter which is of central interest to us at present. (In the CG, phonological distinctive feature matrices are abbreviated by the normal spelling of the morpheme in question.) A syntactic feature specifies that a given morpheme either has or does not have a specific syntactic property. Thus, the features of the CG are marked either "+" (for possession) or

immediately before a Noun Phrase. The verb hit is marked $\langle - \_\_\_\_ \rangle$ by convention. What insures that the NP specified in the feature is in fact the NP dominated by VP in the constituent structure is the principle of strict local subcategorization,³ which asserts that the frames dominated by the constituent which immediately dominates a complex symbol determine the strict subcategorizational properties of the complex symbol. For example, the strict subcategorization of verbs and adjectives is determined by those constituents dominated by VP. The strict subcategorization of nouns is determined by those constituents dominated by NP. The feature $\langle + \_\_\_\_ \text{NP} \rangle$ is interpreted to refer only to the NP which is immediately dominated by VP in the event that the complex symbol containing this feature also contains the feature $\langle +V \rangle$. For verbs, the feature $\langle + \_\_\_\_ \rangle$ means that VP may dominate no constituent other than V.

The third type of syntactic feature renders the notion of inherent feature specification. The characteristics of inherent features are not well understood at present and it is difficult to justify particular classifications. At present, the inherent features of the CG treat several distinct syntactic properties. First, the number of nouns is characterized in terms of the feature $\langle +\text{Sg} \rangle \langle -\text{Sg} \rangle$ where the former is interpreted as singular and the latter as plural. Second, nouns are classified either as human, $\langle +\text{human} \rangle$, or as non-human $\langle -\text{human} \rangle$, and so on for other features such as animateness and concreteness. (Lexical redundancy rules will eliminate lexical redundancy by predicting, for example, that any lexical item marked with the feature $\langle +\text{human} \rangle$ is automatically marked

a rule with transformational power.) For present purposes, the system of selectional subcategorization should be viewed as a classificational scheme, but one without much theoretical support.

At present, the lexicon completely neglects an account of the restrictions which lexical items idiosyncratically impose upon the application of transformational rules. Consider, as an example of this phenomenon, the fact that for is optionally deleted in (25), but obligatorily deleted in (26), at least in the dialect which the CG is intended to characterize.

(25) a. I would prefer for you to say something

b. I would prefer you to say something

(26) a. *I want for you to say something

b. I want you to say something

The fact that an account of these phenomena has not been incorporated into the CG has an important consequence: namely, that it is currently impossible to construct a transformation whose application is determined by an idiosyncratic property of a lexical item. Thus, there is good reason to assign high priority to work in this area and we are, at present, adapting certain aspects of solutions proposed in recent months to the CG.⁴

The version of syntactic theory underlying the CG does not allow the generation of syntactic features in the phrase structure component, although such power is probably required in a fully adequate grammar. Complex symbols are introduced into the phrase structure by means of a lexical embedding rule of the form $A \rightarrow CS$ where A is an N, V, ADJ, or M and CS

This introduction is legal with respect to condition (i), since the complex symbol contains the feature [+N] and has been introduced under the domination of N. But the introduction is illegal with respect to condition (ii). The complex symbol for teapot is specified for the strict subcategorizational feature ⟨+DET_____⟩. A P-marker falling under the domain of this feature is one in which NP dominates only DET, aside from the N itself. The P-marker given in (29) falls under this domain.

(29)



Since NP dominates S in (27), however, this P-marker does not fall under the domain of the one strict subcategorizational feature positively specified for the noun teapot. Thus, this particular application of the lexical introduction rule is illegal and the derivation is consequently blocked. The derivation of an underlying P-marker is terminated when the lexical introduction rule has applied legally to all instances of the terminal symbols N, V, ADJ, M, which have been generated through the application of the rewriting rules. The structure thus produced serves as the input to the transformational component.

The nature of the procedure for introducing prepositions into the underlying structure has not yet been determined. For purposes of mechanical recognition, prepositions can be thought of as being introduced by rewriting rules. An apparatus for preposition introduction will be presented in the updated version of the CG.

The CG contains a lexicon of approximately seven thousand

words drawn from the Thorndike-Lorge list on the basis of an occurrence frequency of ten or more per million. The feature classification of these words is still minimal, since it includes only the following:

Syntactic Category

<+N>	boy
<+V>	slay
<+ADJ>	honest
<+M>	must

Strict Subcategorization

Verbs <+V>

<+___NP S>	tempt
<+___NP>	disappoint
<+___PP>	approve
<+___S>	condescend
<+___>	elapse

Nouns <+N>

<+DET___S>	fact
<+DET___>	teapot
<+___S>	it
<+___>	John

Inherent Subcategorization

Nouns <+N>

<+human>	boy
<+animate>	mongoose
<+abstract>	blame
<-animate>	table
<-abstract>	

Selectional Subcategorization

Verbs (+V)

$\langle + \_\_\_\_ \langle + \_\_\_\_ S \rangle \rangle$	suppose
$\langle + \langle + \_\_\_\_ S \rangle S \text{ AUX } \_\_\_\_ \rangle$	bother
$\langle + \_\_\_\_ (\text{DET}) \langle + \text{human} \rangle \text{PREP} \langle + \_\_\_\_ S \rangle \rangle$	remind

Adjectives (+ADJ)

$\langle + \langle + \_\_\_\_ S \rangle S \text{ AUX } \_\_\_\_ \rangle$	obvious

This list of classifications should not be construed as a completed lexicon, for it will be immediately clear that the classifications are much too incomplete to allow the free generation of strings which are free from strict subcategorizational and selectional violations. The lexicon has been constructed with two goals in mind; first, to study the theoretical problems arising in the description of lexical phenomena within the framework of a transformational grammar, and second, to develop an experimental lexicon for the purpose of testing a variety of computational procedures.

3.0 The Transformational Component

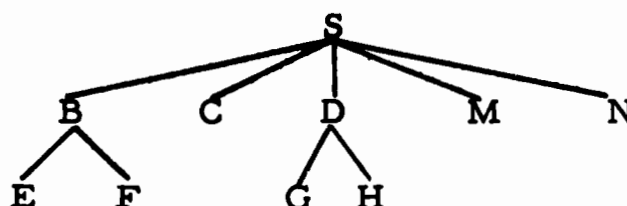
3.1 Introduction

The transformational component of the CG contains rules which transform underlying P-markers into derived P-markers. The domain of a particular transformational rule is provided in terms of conditions on P-markers. Any P-marker or set of P-markers meeting the conditions imposed by a particular transformational rule falls under the domain of that rule. By way of clarification, consider the following hypothetical transformational rule.

(30)	B + C	D	X
	1	2	3====>
	1	Ø	3

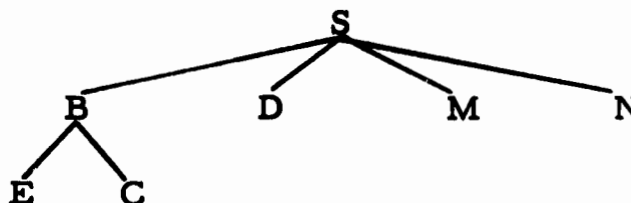
The numbered sequence of elements comprising the first two lines of the above rule (referred to as a structure index) defines the set of P-markers which may undergo the transformational alteration stated in the third line of the rule. The structure index can be interpreted as asserting that any terminal string (last line of a P-marker) falls under the domain of the transformation (30) just in case it can be completely segmented into three consecutive substrings such that the first substring is a (member of the constituent or category sequence) B + C, the second is a D, and the third is anything at all. The diagram (31) contains a P-marker which falls under the domain of the transformation (30).

(31)



The terminal string of (31), i. e., E - F - C - G - H - M - N, can be segmented in such a way that the conditions stipulated by the structure index are met. The diagram (32) contains a P-marker which does not fall under the domain of the transformation (30).

(32)



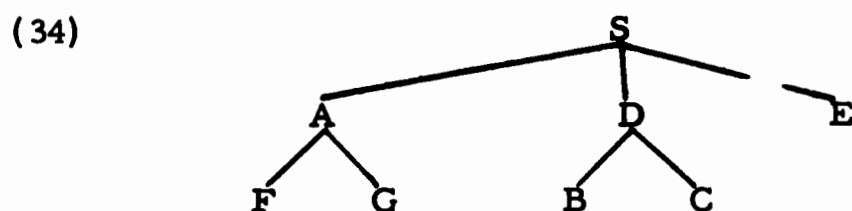
It is clear that transformational rules stated in the above fashion have the power of variable reference, since each structure index characterizes a variety of P-markers. For example, the transformation (30) would be defined on the

P-marker (31) regardless of the constituency dominated by B. If the phrase structure component from which this P-marker was constructed contains the rule $B \rightarrow E + L + S$, then an infinite number of P-markers are provided, all of which will fall under the domain of the transformation (30).

Transformation rules often involve two additional types of restrictions on P-markers. The first is the restriction of dominance where some subtree must either have a certain analysis or must dominate some particular subtree. This information is normally stated in the transformational rule itself by means of a labelled bracketing. Consider, by way of illustration, the following hypothetical rule:

$$\begin{array}{rclcl}
 (33) & A & [B & C]_D & E \\
 & 1 & 2 & 3 & 4 \\
 & 1 & \emptyset & 3 & 4
 \end{array}$$

This transformation is defined on a terminal string just in case the P-marker associated with this string can be segmented into substrings such that the first is an A, the second is a B, the third is a C, the fourth is an E, and, furthermore, the second followed by the third comprises a D, as in the P-marker given in (34).



The transformation (35) is defined on a terminal string just in case

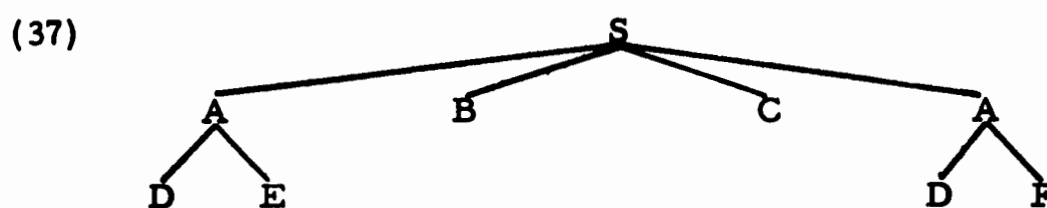
(35)	A	[B C] _D	E
	1	2	3====>
	1	∅	3

the P-marker associated with this string can be segmented into substrings such that the first is an A, the second is a D which dominates a B followed by a C, and the third is an E, as in (34) also.

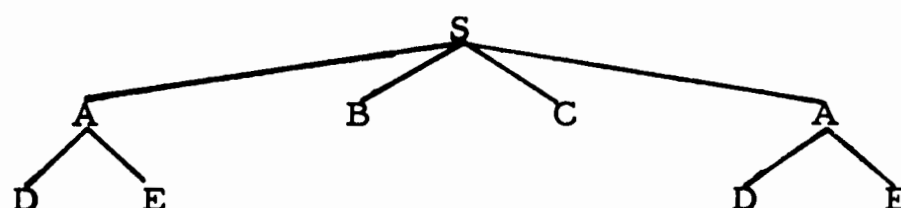
The second type of restriction is the condition of identity where a subtree must be identical or not identical to some other subtree.⁵ Such a restriction is not stated in the structure index or in the line which specifies the structure change, but as a condition on the rule, as in (36).

(36)	A	B	C	A
	1	2	3	4====>
	1	2	∅	4
	Condition: 1 ≠ 4			

This rule is defined on a terminal string just in case the P-marker associated with this string can be segmented into four consecutive substrings, of which the first is an A, the second is a B, the third is a C, and the fourth is an A, but where the subtree A associated with the index 1 is not identical to the subtree A associated with the index 4. For example, this rule is defined on the P-marker in (37), but not on the P-marker in (38).



(38)



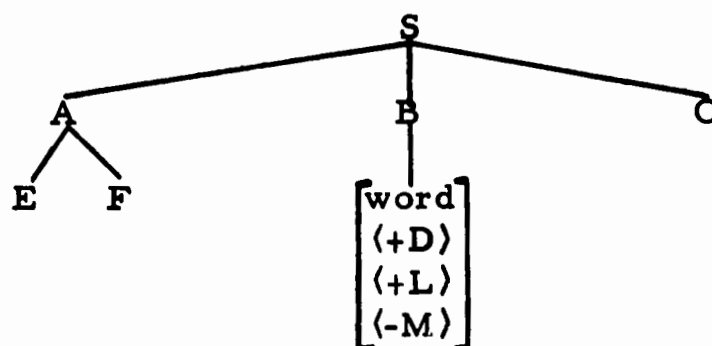
Transformational rules are sensitive to the syntactic feature composition of complex symbols including feature coefficients. In other words, a transformational rule may be defined just in case the syntactic feature vector associated with a complex symbol introduced into a P-marker meets certain internal conditions. Consider, for example, the hypothetical rule given in (39).

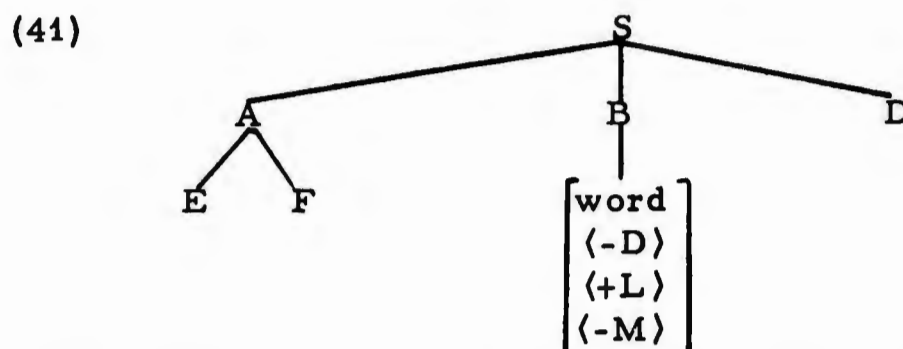
(39) A [(+D)]_B C

1	2	3====>
1	∅	3

A terminal string falls under the domain of the above rule just in case it can be segmented into three consecutive substrings of which the first is an A, the second is a terminal symbol B which dominates a complex symbol in whose syntactic feature vector the feature ⟨D⟩ is marked positively, and the third is a C. Thus, the P-marker in (40) falls under the domain of this transformation whereas the P-marker in (41) does not.

(40)



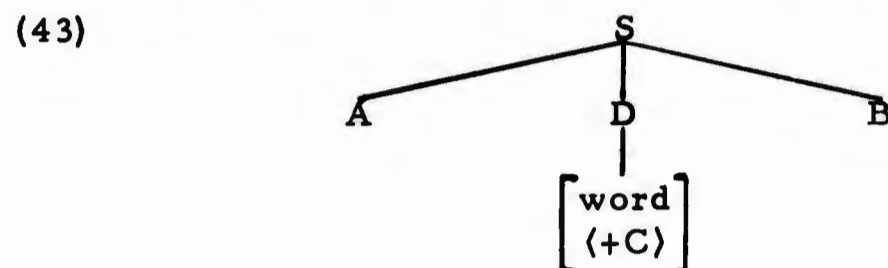


The transformational rule given in (42) requires a special interpretation since $\langle +C \rangle$ is not a constituent but a feature in a vector.

(42)

A	$\langle +C \rangle$	B
1	2	3====>
1	2	$\emptyset$

A P-marker falls under the domain of the above transformation just in case the terminal string can be segmented into three consecutive substrings of which the first is an A, the second is a complex symbol marked positively for the feature C, and the third is a B, as in the P-marker given in (43).



A transformational rule contains a finite sequence of formal operations called elementary transformations. An elementary transformation operates on the n terms of a structure index and determines the alterations to be made in the P-markers which fall under the domain of the transformation. The elementary transformations employed in the CG are substitution, deletion, sister adjunction, and daughter adjunction.

(There is reason to believe that daughter adjunction is theoretically unmotivated. It is likely that this elementary transformation will not appear in the updated version of the CG.)

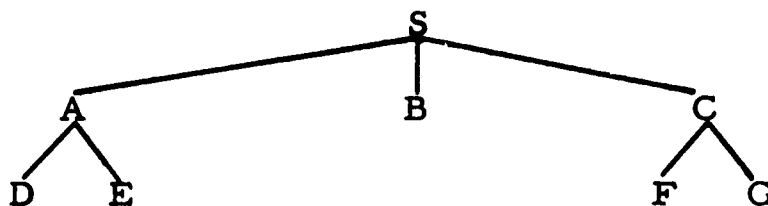
Under the substitution elementary transformation, one subtree is substituted for another subtree. Consider, for example, the hypothetical rule given in (44).

(44)

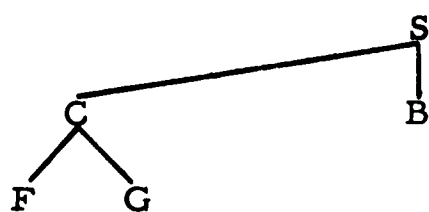
A	B	C
1	2	3====>
3	2	∅

In this rule a subtree having the analysis C is substituted for a subtree having the analysis A in a P-marker which falls under the domain of the transformation. This transformation will, for instance, convert the P-marker in (45) into the P-marker in (46).

(45)

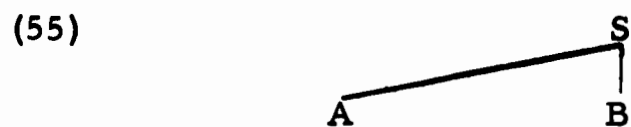


(46)

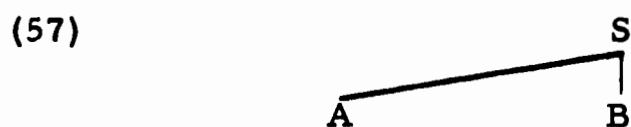
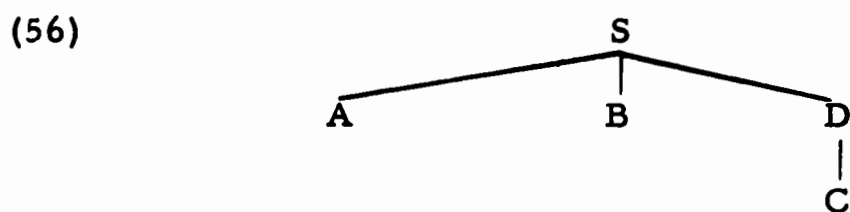


The deletion elementary transformation involves the deletion of a subtree. An example of such a rule is given in (53). This rule converts the P-marker in (54) into the P-marker in (55).

(53)	A	B	C
	1	2	3====>
	1	2	$\emptyset$

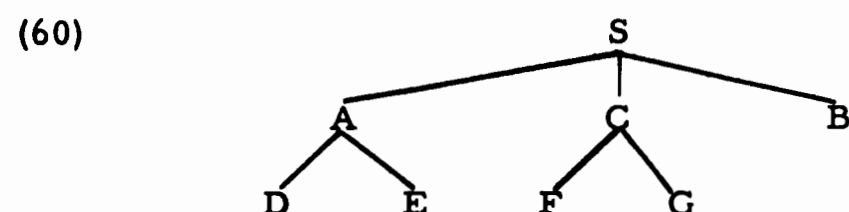
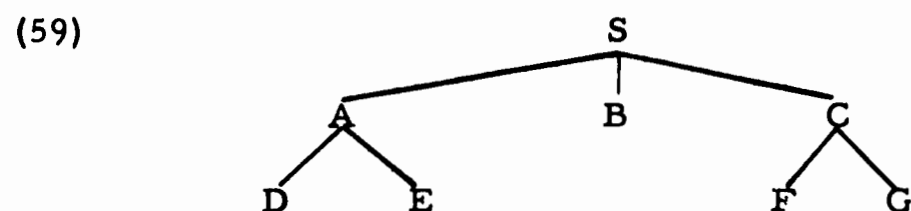


In the event that some constituent X is the sole daughter of some constituent Y and the deletion X is defined, then Y as well is deleted. Thus, the rule (53) converts the P-marker (56) into the P-marker (57).

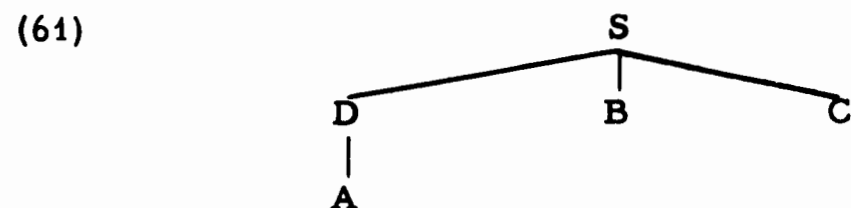


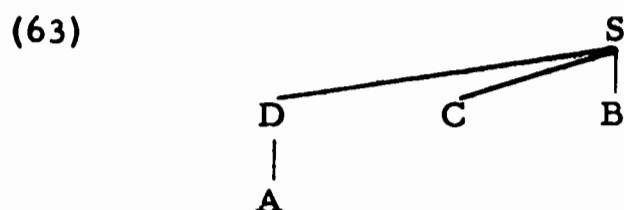
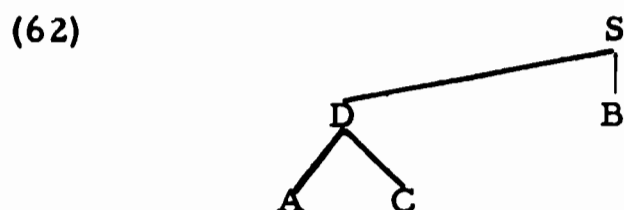
The elementary transformation of sister adjunction involves the introduction of some subtree under the immediate domination of some constituent which has at least one daughter. For example, the rule in (58) converts the P-marker (59) into the P-marker (60).

(58)	A	B	C
	1	2	3====>
	1 + 3	2	$\emptyset$

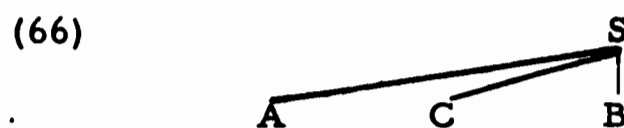
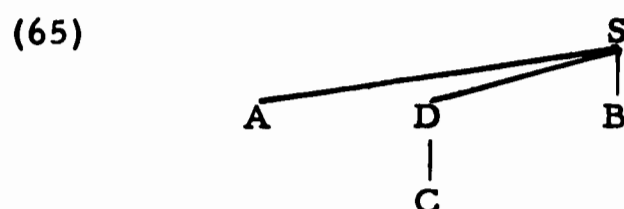
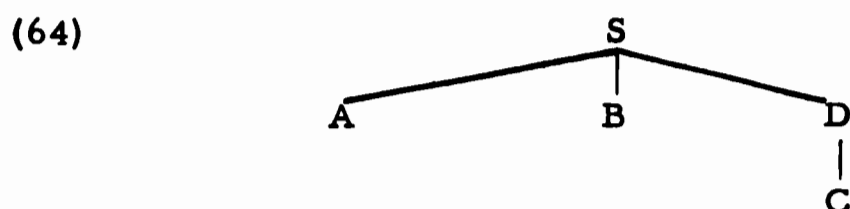


In contradistinction to the substitution transformation, if a constituent X is the sole daughter of a constituent Y, then a transformation which defines the sister adjunction of Z to X does not adjoin Z to Y. The transformation (58) operating on the P-marker (61) yields the P-marker (62) and not the P-marker (63).





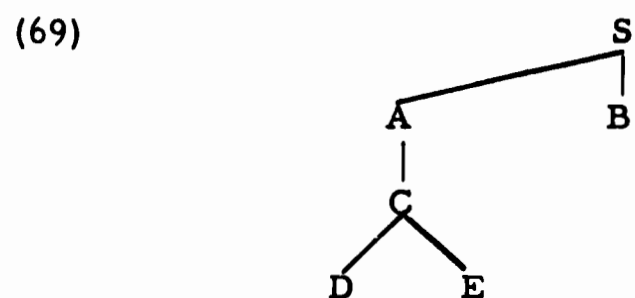
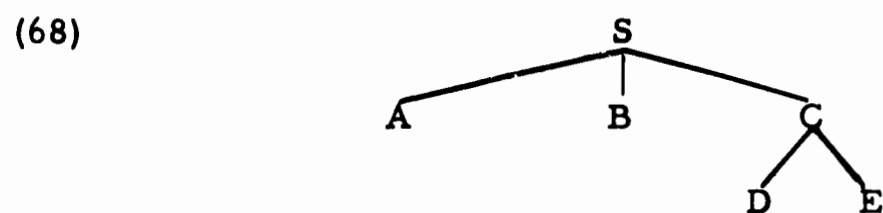
However, if a constituent X, which is to be adjoined as a sister of Y, is the sole daughter of a constituent Z, then Z is adjoined to Y. Thus, the transformation (58) operating on the P-marker (64) yields the P-marker (65) and not the P-marker (66).



The elementary transformation of daughter adjunction involves the introduction of some subtree under the immediate domination of some constituent which dominates no other constituent. For example, the transformation (67) operates on the P-marker in (68) and produces the P-marker in (69).

(67)	A	B	C
	1	2	3====>
	1 < 3	2	∅

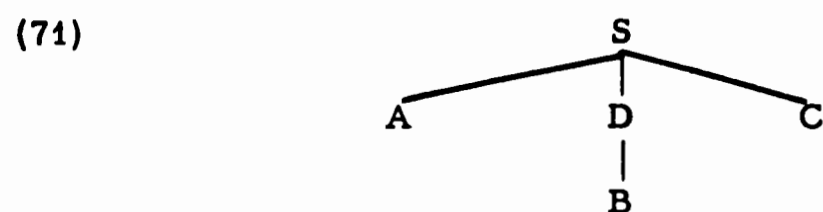
(where "<" is interpreted as immediately dominates)



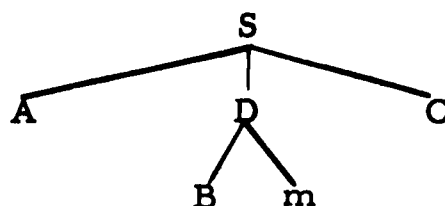
If a constituent X, which is to be adjoined as a daughter to some constituent Y which dominates no other constituents, is the sole daughter of some constituent Z, then Z is adjoined as a daughter to Y.

Transformational rules may introduce morphemes into a P-marker. For instance, the transformation in (70) converts the P-marker in (71) into the P-marker in (72).

(70)	A	B	C
	1	2	3====>
	1	2 + m	3

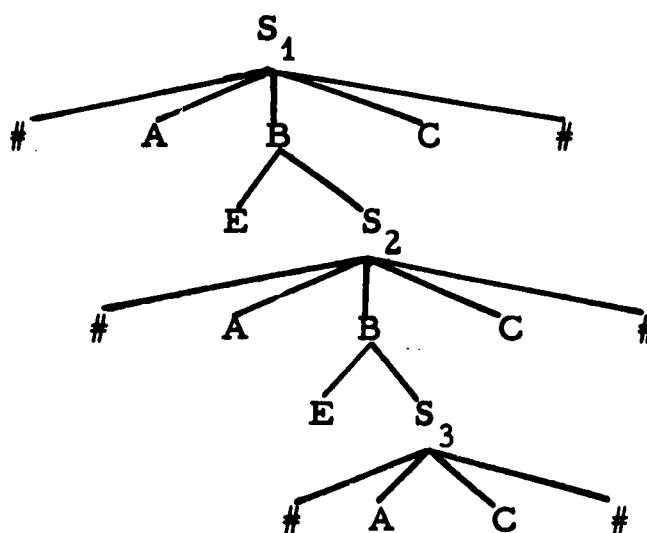


(72)



The transformational component of the CG contains an ordered set of cyclic and post-cyclic transformational rules. The cyclic rules apply to a lowest sentence. A lowest sentence is an S which dominates the sequence # - X - #, where # is a sentence boundary and X is a variable which does not contain #. In the diagram (73), S_3 meets the conditions of a lowest S.

(73)



The cyclical transformational rules apply in sequence to lowest S's. Consider, for instance, the following set of cyclic rules in which the symbol X is a variable ranging over any structure at all.

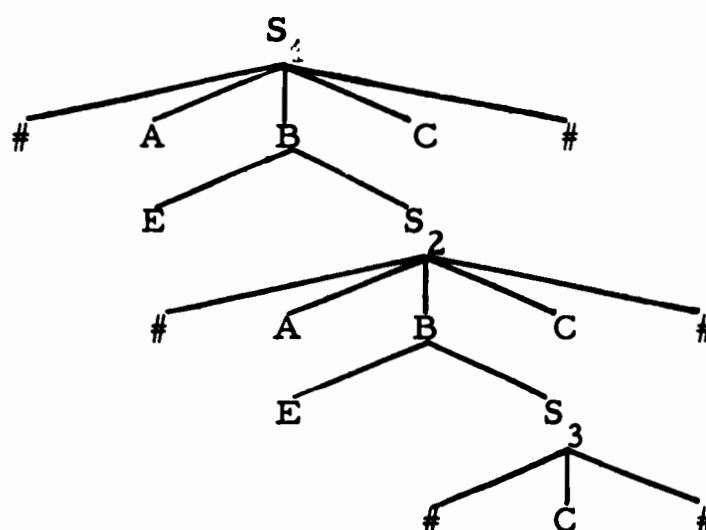
(74)	a.	#	A	X	#
		1	2	3	4====>
		1	∅	3	4

b.

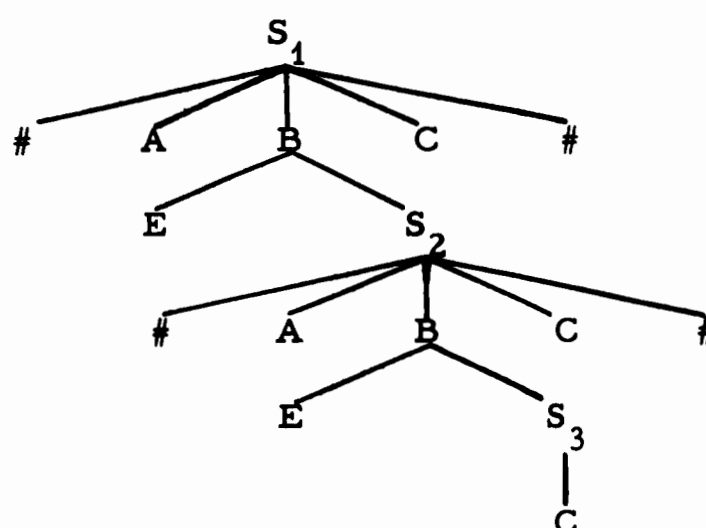
#	X	#
1	2	3====>
∅	2	∅

Applying cyclically, the rules in (74) operate on S_3 in the P-marker given in (73) producing, sequentially, the P-markers (75) and (76).

(75)

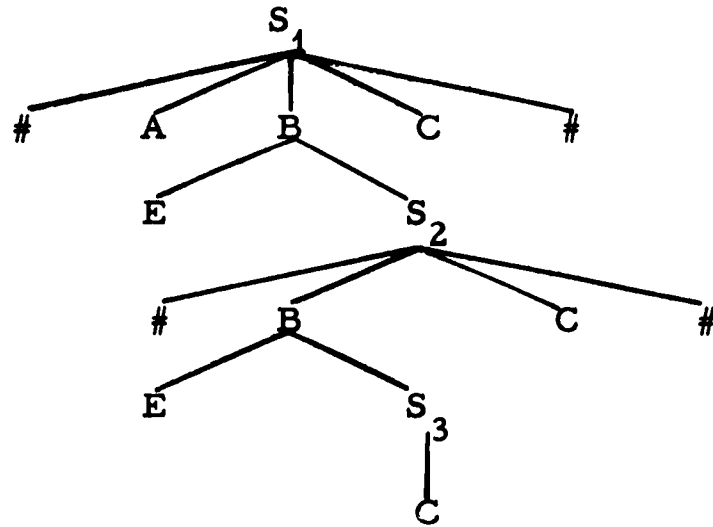


(76)

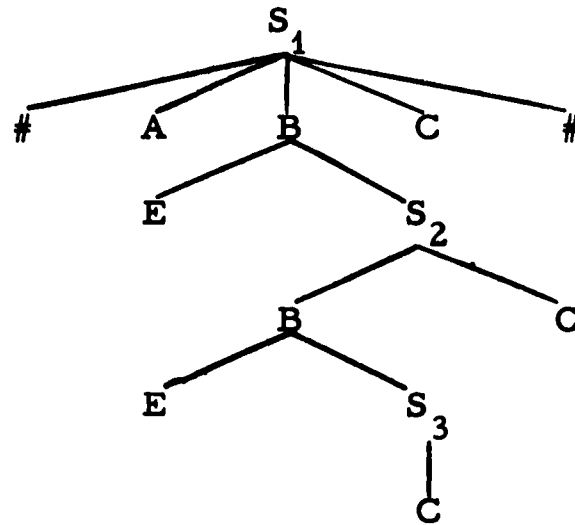


As the result of the application of the rules in (74), S_2 now meets the conditions of a lowest S and the cyclic rules apply again yielding the P-markers (77) and (78).

(77)

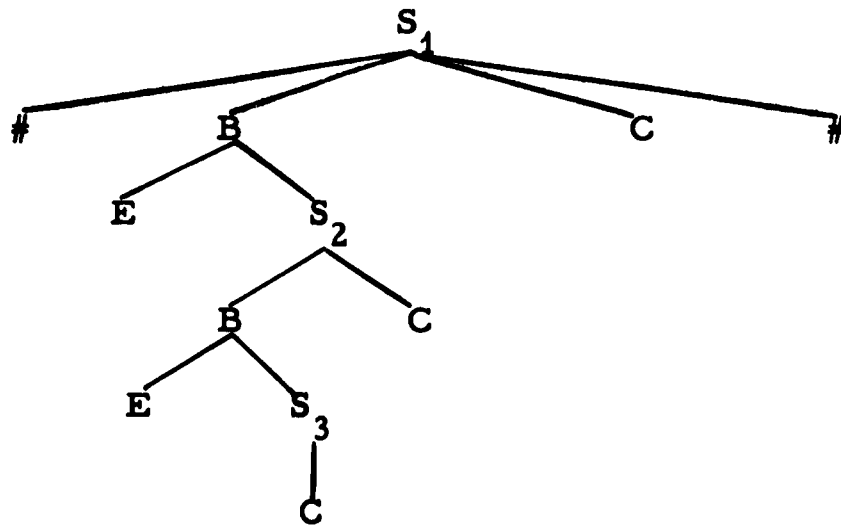


(78)

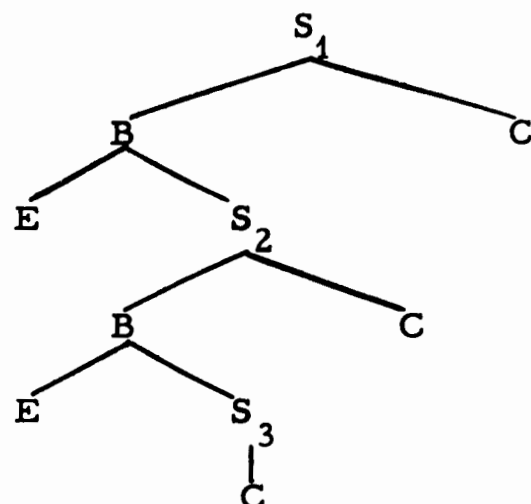


After this cycle S_1 now meets the conditions of the lowest S and the cyclic rules apply once again yielding (79) and (80).

(79)



(80)

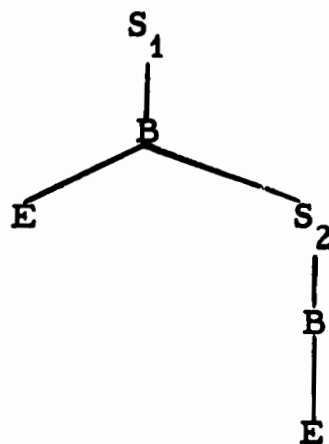


This application of the rules leaves a P-marker in which no S qualifies as a lowest S. Hence, the transformational cycle is terminated.

The P-marker produced by the rules of the transformational cycle is the input to the post-cyclic transformational rules. A possible rule might be that in (81), which deletes every instance of C in (80), thus generating the derived P-marker (82).

(81)	X	C	Y
	1	2	3====>
	1	∅	3

(82)



The post-cyclic rules are ordered. The derivation is terminated after the applicability of the last post-cyclic rule has been tested. The transformations, cyclic and post-cyclic, in the CG are as follows:

3.2 Transformational Rules⁶

I. CYCLIC RULES

1. CP 1 Complementizer Placement 1⁷

#	X	$[\langle \alpha C \rangle]_N$	NP + $\begin{Bmatrix} T \\ \text{be} \\ \text{have} \\ V \end{Bmatrix}$	Y	#
1	2	3	4	5	6====>
1	2	3	$\alpha C + 4$	5	6

2. CP 2 Complementizer Placement 2

OB

#	X	$[\langle +C \rangle]_V$	(NP)	NP + $\begin{Bmatrix} T \\ \text{be} \\ \text{have} \\ V \end{Bmatrix}$	Y	#
1	2	3	4	5	6	7====>
1	2	3	4	$+C + 5$	6	7

3. CP 3 Complementizer Placement 3

OB

#	X	N	$[NP + Y]_S$	Z	#
1	2	3	4 5	6	7====>
1	2	3	that + 4 5	6	7

4. IE Identity Erasure^{8,9} OB

#	W	(NP)	X	αC	NP	Y	(NP)	Z	#
1	2	3	4	5	6	7	8	9	10====>
1	2	3	4	5	$\emptyset$	7	8	9	10

Condition: An NP_j is erased by an identical NP_i if and only if there is an S_n such that

- (i) NP_j is dominated by S_n
- (ii) NP_i neither dominates nor is dominated by S_n
- (iii) for all NP_k neither dominating nor dominated by S_n , the distance between NP_j and NP_k is greater than the distance between NP_j and NP_i where distance between two nodes is defined in terms of the number of branches in the path connecting them.

5. IOI Indirect Object Inversion OP

#	X	V	$\begin{Bmatrix} PP \\ NP \end{Bmatrix}$	to + NP	Y	#
1	2	3	4	5	6	7====>
1	2	3 + 5	4	$\emptyset$	6	7

6. TO To Deletion OB

#	X	V	to	NP	(PREP) + NP	Y	#
1	2	3	4	5	6	7	8====>
1	2	3	$\emptyset$	5	6	7	8

7. PASSIVE Passive OB

#	(PRE)	NP ₁	AUX		V	(PREP)	NP ₂	X	PREP	P	Y	#
1	2	3	4		5	6	7	8	9	10	11	12====>
1	2	7	4	be + en +	5	6	∅	8	9	3	11	12

Condition: 3 ≠ 7

8. EXTRA Extraposition OP

#	X	[<+____S>] _N	S	Y	#
1	2	3	4	5	6====>
1	2	3	∅	5	4 + 6

9. PROREP Pronoun Replacement OB

#	X	[[<+____S>] _N] _{NP}	(AUX (be en) V + (MAN))	aC	NP	Y	#
1	2	3		4	5	6	7 8====>
1	2	6		4	5	∅	7 8

10. WHA WH-Attraction OB

#	U	ART	[NP	W	$\left\{ \begin{array}{l} \text{PREP} + [\text{WH } X]_{\text{NP}} \\ [\text{WH } X]_{\text{NP}} \end{array} \right\}$	Y] _S	Z	#
1	2	3	4	5	6	7	8	9====>
1	2	3	6 + 4	5	∅	7	8	9

11. RELPLACE Relative Placement OB

#	X	ART	S	N	Y	#
1	2	3	4	5	6	7====>
1	2	3	∅	5 + 4	6	7

12. AUXFILL Auxiliary Filler OB

#	X	T	$\left\{ \begin{array}{l} \text{be} \\ \text{have} \end{array} \right\}$	Y	#
1	2	3	4	5	6====>
1	2	3 + 4	∅	5	6

13. AG Agreement OB

#	(PRE)	[(DET)[$\langle \alpha \text{Sg} \rangle$] _N ^X] _{NP}	$\left\{ \begin{array}{l} \text{PRES} \\ \text{PAST} \end{array} \right\}$	Y	#
1	2	3	4	5	6====>
1	2	3	4 < αSg	5	6

Condition: 4 < ∅

14. EVER Ever OP

#	X	INDEF	[$\langle -\text{human.} \rangle$] _N	WH	$\left\{ \begin{array}{l} \text{INDEF} \\ \text{DEF} \end{array} \right\}$	N	Y	#
1	2	3	4	5	6	7	8	9====>
1	2	3	4	5	6 + ever	7	8	9

15. REGDEL Regressive Deletion

a. OP

b. OB

#	X	$\left\{ \begin{array}{l} \text{ART} \quad [\langle +\text{PRO} \rangle]_N \\ \text{INDEF} \quad N \end{array} \right\}$		WH + INDEF + $\left\{ \begin{array}{l} \text{a. } \emptyset \\ \text{b. ever} \end{array} \right\}$		N	Y	#
1	2	3	4	5	6	7	8	====>
1	2	$\emptyset$	$\emptyset$	5	6	7	8	

Condition: 4 = 6

16. DEFI Definitization

OB

#	X	N	[(PREP) + WH		INDEF	N	Y]	Z	#
							_S		
1	2	3	4	5	6	7	8	9	====>
1	2	3	4	DEF	6	7	8	9	

17. WHAG WH-Agreement

OB

#	X	WH	$\left\{ \begin{array}{c} \text{INDEF} \\ \text{DEF} \end{array} \right\}$	(ever)	$[\langle \alpha\text{human} \rangle]_N$	Y	#
1	2	3	4	5	6	7	8====>
1	2	3	4 $\langle \alpha\text{human}$	5	6	7	8

Condition: 4 < $\emptyset$

18. PROGDEL Progressive Deletion

OB

#	X	N	(PREP) + WH + DEF		N	Y	#
1	2	3	4	5	6	7	====>
1	2	3	4	$\emptyset$	6	7	

Condition: 3 = 5

19. RELDEL Relative Deletion OP

#	X	N	[WH Y] _{NP} + be + PRES	ADJ	Z	#
1	2	3	4	5	6	7====>
1	2	3	∅	5	6	7

20. ADJPLACE Adjective Placement OB

#	X	N	ADJ	Y	#
1	2	3	4	5	6====>
1	2	4 + 3	∅	5	6

21. CDUP Complementizer Duplication OB

#	X	αC	NP	Y	#
1	2	3	4	5	6====>
1	2	3	4 + 3	5	6

22. CNEG C Negative Placement OP

#	X	αC	($\left\{ \begin{smallmatrix} \text{be} \\ \text{have} \end{smallmatrix} \right\}$) + T	NEG	Y	#
1	2	3	4	5	6	7====>
1	2	5 + 3	4	∅	6	7

23. CTENSE C Tense OB

#	X	αC	($\left\{ \begin{smallmatrix} \text{be} \\ \text{(+V)} \end{smallmatrix} \right\}$) + PAST	Y	#
1	2	3	4	5	6====>
1	2	3 + have + en	4	5	6

24. TS Tense Suppression

OB

#	X	αC	$\left\{ \begin{array}{l} \text{be} \\ (\text{have} + (\text{en}))(\langle +V \rangle) \end{array} \right\}$	T	Y	#
1	2	3	4	5	6	7====>
1	2	3	4	$\emptyset$	6	7

25. CD Complementizer Deletion

OP

#	X	V	$[\langle +\_S \rangle]_N$	αC	NP	Y	#
1	2	3	4	5	6	7	8====>
1	2	3	4	$\emptyset$	6	7	8

26. TAG Tag Question

OP

#	NEG	Q	NP	AUX	VP	Y	#
1	2	3	4	5	6	7	8====>
1	2	$\emptyset$	4	5	6	7	3+4+5+8

27. NEGPLACE Negative Placement

OB

#	NEG	(Q)	NP	AUX	X	#
1	2	3	4	5	6	7====>
1	$\emptyset$	3	4	5 + 2	6	7

28. NEGTAG Tag Negative Placement OP

#	X	AUX	NEG	VP	(S)	Q	NP	AUX	#
1	2	3	4	5	6	7	8	9	10====>
1	2	3	∅	5	6	7	8	9 + 4	10

29. NEGAUX Negative Auxiliary OP

#	X	T + ($\begin{Bmatrix} \text{be} \\ \text{have} \\ \text{M} \end{Bmatrix}$)	NEG	Y	#
1	2	3	4	5	6====>
1	2	3 + 4	∅	5	6

30. QUES Question OB

#	Q	X	$\left\{ \begin{array}{l} \text{PREP} + [\text{WH} + \text{Y}]_{\text{NP}} \\ [\text{WH} + \text{Y}]_{\text{NP}} \end{array} \right\}$	Z	#
1	2	3	4	5	6====>
1	4 + 2	3	∅	5	6

31. YESNO Yes-No Question OB

#	X	Q	NP	AUX	Y	#
1	2	3	4	5	6	7====>
1	2	5 + 3	4	∅	6	7

32. AF Affix

OB

#	X	$\left\{ \begin{array}{c} T \\ -C \\ \text{ing} \\ \text{en} \end{array} \right\}$	$\left\{ \begin{array}{c} NP \\ \langle +V \rangle \\ \langle +M \rangle \\ \text{be} \\ \text{have} \end{array} \right\}$	Y	#
1	2	3	4	5	6====>
1	2	$\emptyset$	4 + 3	5	6

Condition: 5 = a terminal string $\sigma_1, \sigma_2, \dots, \sigma_n$,
 such that $\sigma_1 \neq \underline{-C}, \underline{\text{ing}}, \underline{\text{en}},$ or $\underline{T}$ and
 2 = a terminal string $\phi_1, \phi_2, \dots, \phi_n$,
 such that $\phi_n \neq \langle +V \rangle$ or $\langle +M \rangle$.

33. PREPDEL Preposition Deletion

OB

#	X	PREP	$\left[\begin{array}{c} \langle + \text{ } \underline{\text{ } S} \rangle \\ \text{not } \langle -C \rangle \end{array} \right]_N$	Y	#
1	2	3	4	5	6====>
1	2	$\emptyset$	4	5	6

34. PD Pronoun Deletion

OB

#	X	$[\langle + \text{ } \underline{\text{ } S} \rangle]_N$	(to + NP)	S	Y	#
1	2	3	4	5	6	7====>
1	2	$\emptyset$	4	5	6	7

35. AGDEL Agent Deletion OP

#	X	[PREP + INDEF + [(+PRO)] _N] _{MAN}	Y	#
1	2	3	4	5====>
1	2	∅	4	5

36. THAT that Deletion OP

#	X	$\begin{Bmatrix} V \\ \text{ADJ} \end{Bmatrix}$	(NP)	[that	Y] _S	Z	#
1	2	3	4	5	6	7	8====>
1	2	3	4	∅	6	7	8

37. VPCOMP VP Complement Placement OP

#	X	MAN	+C + $\begin{Bmatrix} \text{AUX} \\ \text{have} \end{Bmatrix}$	+VP	Y	#
1	2	3	4	5	6	6====>
1	2	4 + 3	∅	5	6	

38. BEDEL be Deletion OB

#	X	M	be	+C	Y	#
1	2	3	4	5	6	7====>
1	2	3	∅	5	6	7

39. MCDEL Modal Complementizer Deletion OB

#	X	M	+C	Y	#
1	2	3	4	5	6====>
1	2	3	Ø	5	6

40. QDEL Q Deletion OB

#	X	Q	Y	#
1	2	3	4	5====>
1	2	Ø	4	5

41. ERASE Boundary Erasure OB

#	X	#
1	2	3====>
Ø	2	Ø

II. POST-CYCLIC RULES

42. PAST Past OB

X	$\left\{ \begin{array}{l} \langle +V \rangle \\ \langle +M \rangle \end{array} \right\}$	PAST	Y
1	2	3	4====>
1	2	ed	4

43. MTDEL Modal Term Deletion OB

X	$\langle +M \rangle$	PRES	Y
1	2	3	4====>
1	2	$\emptyset$	4

44. PLUDEL Plural Deletion OB

X	$\langle +V \rangle$	$[-Sg]_{PRES}$	Y
1	2	3	4====>
1	2	$\emptyset$	4

45. NUM Number OB

X	$[-Sg]_N$	Y
1	2	3====>
1	2 + ■	3

46. NUAG Number Agreement

OB

X	(+V)	+Sg	Y
1	2	3	4====>
1	2	s	4

47. CONTR Contraction

OB

X	[Y NEG]	AUX	Z
1	2	3	4====>
1	2	n't	4

48. NEGSPELL Negative Spelling

OB

X	NEG	Y
1	2	3====>
1	not	3

49. DO 1 Do 1

OB

X	[[aSg] _{PAST}	Y] _{AUX}	Z
1	2	3	4====>
1	did	3	4

50. DO 2 Do 2

OB

X	[[+Sg] _{PRES}	Y] _{AUX}	Z
1	2	3	4====>
1	does	3	4

51. DO 3 Do 3 OB

X	[[-Sg] _{PRES}	Y] _{AUX}	Z
1	2	3	4====>
1	do	3	4

52. BE 1 Be 1 OB

X	be	[+Sg] _{PRES}	Y
1	2	3	4====>
1	∅	is	4

53. BE 2 Be 2 OB

X	be	[-Sg] _{PRES}	Y
1	2	3	4====>
1	∅	are	4

54. BE 3 Be 3 OB

X	be	[+Sg] _{PAST}	Y
1	2	3	4====>
1	∅	was	4

55. BE 4 Be 4 OB

X	be	[-Sg] _{PAST}	Y
1	2	3	4====>
1	∅	were	4

56. HAVE 1 Have 1 OB

X	have	[+ Sg] _{PRES}	Y
1	2	3	4====>
1	∅	has	4

57. HAVE 2 Have 2 OB

X	have	[-Sg] _{PRES}	Y
1	2	3	4====>
1	∅	have	4

58. HAVE 3 Have 3 OB

X	have	PAST	Y
1	2	3	4====>
1	∅	had	4

59. WHPD 1 WH Pronoun Deletion 1

OP

X	DEF	$\left[\begin{array}{c} \langle +\text{PRO} \rangle \\ \langle +\text{Sg} \rangle \end{array} \right]_N$	WH + Y
1	2	3	4====>
1	2	$\emptyset$	4

60. WHPD 2 WH Pronoun Deletion 2

OE

X	WH + INDEF + (ever)	$\left[\begin{array}{c} \langle +\text{PRO} \rangle \\ \langle +\text{Sg} \rangle \end{array} \right]_N$	Y
1	2	3	4====>
1	2	$\emptyset$	4

61. WHDEL WH-Deletion

OP

X	N	$[\text{WH} + \text{Y}]_{NP}$	NP	Z
1	2	3	4	5====>
1	2	$\emptyset$	4	5

62. DEFTHAT Definite that

OB

X	DEF	WH + Y
1	2	3====>
1	that	3

63. WH 1 WH 1 OB

X [WH [<+human>] $\left\{ \begin{array}{c} \text{DEF} \\ \text{INDEF} \end{array} \right\}]_{\text{NP}}$ Y

1 2 3 4====>

1 $\emptyset$ who 4

64. WH 2 WH 2 OB

X WH + DEF Y

1 2 3====>

1 which 3

65. WH 3 WH 3 OB

X WH + INDEF Y

1 2 3====>

1 what 3

66. PLADEL Plural Article Deletion OB

X INDEF [<-Sg>]_N Y

1 2 3 4====>

1 $\emptyset$ 3 4

67. C 1 C 1 OB

X	+C	NP	Y
1	2	3	4====>
1	for	3	4

68. C 2 C 2 OB

X	+C	Y
1	2	3====>
1	to	3

69. C 3 C 3 OB

X	NP	-C	Y
1	2	3	4====>
1	2	's	4

70. C 4 C 4 OB

X	-C	X
1	2	3====>
1	ing	3

71. BY By

OB

X [PREP NP]_{MAN} Y

1 2 3 4====>

1 by 3 4

72. INDEF Indefinite

OB

X INDEF Y

1 2 3====>

1 a 3

73. DEF Definite

OB

X DEF Y

1 2 3====>

1 the 3

4.0 Derivations on the CG

4.1 Conventions

A derivation consists of four parts: An Actual String, a Derived String, an Underlying Structure (P-marker), and a Derivation.

An Actual String is an orthographic representation of a grammatical sentence in English. It identifies, in other words, one of the sentences for which the CG offers a partial characterization. Lacking a phonological component, the characterization offered by the CG is necessarily incomplete. The CG is limited to syntactic characterization.

A Derived String provides the sequence of morphemes generated by the grammar. The immediate constituency of this sequence is not listed in the derived string but is uniquely determined, nonetheless, by the transformational rules applying to the structure underlying the string.

The Underlying Structure provides a record of the particular application of rewriting and lexical introduction rules which, upon the application of the transformational rules, yields the sequence in the derived string. In an underlying structure, complex symbols are represented by a word, e. g., John, which is an abbreviation for the phonological distinctive feature matrix associated with this word, and by a list of syntactic features, e. g., $\langle +N \rangle$, $\langle +Sg \rangle$, which are relevant to the application of certain transformational rules.

A Derivation is a record of the transformational rules which apply to the Underlying Structure. Each Derivation contains as many cycles of transformational application as there are S's in the Underlying Structure, followed by one application

of the post-cyclic rules. A line of derivation is a record of the structure produced as the result of the application of the transformation specified at the right of the line of derivation and also a record of the structure requisite to the application of the transformation specified at the right of the succeeding line. Structure introduced by conventions on elementary transformations is not incorporated into a line of derivation. For example, in Sentence Type No. 30, the application of the Affix Transformation in the First Cycle incorporates AUX as well as T into V, since T is the sole daughter of AUX. Since AUX is adjoined to the complex symbol by convention, and not as the result of transformational specification, AUX is not represented in the line of derivation characterizing the output of the Affix Transformation. The incorporation or elimination of structure in a line of derivation does not necessarily imply that these alterations are the result of transformational application. When structure ceases to be relevant to the application of subsequent transformations, it is no longer listed in a line of derivation (although, of course, the structure continues to exist in the full P-marker). Similarly, structure is listed in a line of derivation only when it becomes relevant to the application of subsequent transformations.

4.2 Sentence Types

1. the boy likes the girl
2. the boys like the girl
3. the boy liked the girl
4. the boy does not like the girl
5. the boy will like the girl
6. the boy would like the girl

7. the boy will not like the girl
8. the boy is admiring the girl
9. the boy isn't admiring the girl
10. the boy has been admiring the girl
11. the boy will have been admiring the girl
12. does the boy like the girl?
13. doesn't the boy like the girl?
14. John likes the girl doesn't he?
15. John does not like Mary does he?
16. is John admiring Mary?
17. the books were purchased by John
18. must Mary be tormented by John?
19. John gave the book to Mary
20. John offered Mary the book
21. the book was offered to Mary by John
22. Mary was offered the book by John
23. who sleeps?
24. what boy sleeps?
25. which things slip?
26. what slips?
27. what book has John not taken?
28. about what did John speak?
29. the boy who must leave will leave
30. the book of which John speaks is awful
31. the book John speaks of is awful
32. John touched that which annoys Bill
33. Bill can visualize what will fall
34. whatever falls will bounce
35. a tall boy arrived

36. which tall boy did John see?
37. John would like for Mary to leave
38. John wants Mary to leave
39. John wants Mary to be loved by Bill
40. John prefers for Bill not to leave
41. Bill would prefer for John not to have dreamed
42. for John not to drown would be preferred
43. it is required for John to stand
44. Bond was believed to be dead by Goldfinger
45. John loves to run
46. John likes to be taken
47. John thinks Bill to be silly
48. John decided for Bill to represent Harry
49. John decided on Bill to represent Harry
50. John appears to have fallen
51. it embarrasses Bill to trip
52. John may resemble Bill
53. John dislikes Bill's annoying Mary
54. John dislikes Bill annoying Mary
55. John dislikes annoying Mary
56. John decided on going
57. John thinks that Bill will go
58. John thinks Bill smokes
59. that Bill smokes was mentioned by John
60. Bill mentioned to Mary that John smokes
61. it was mentioned by John that Bill smokes
62. Bill tells Mary John smokes
63. Bill reminded Mary to go
64. John tempted Mary to go

65. John condescended to go

66. John stops wondering

4. 3 Derivations

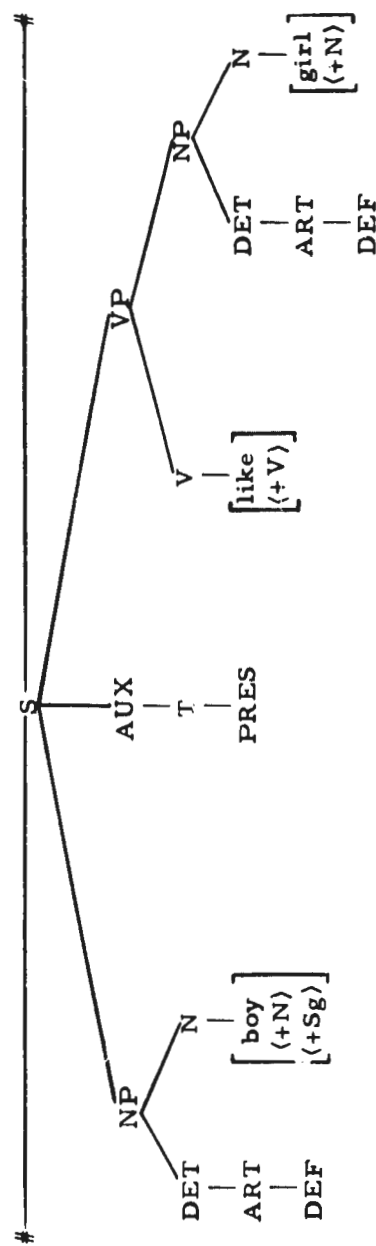
(see following pages)

Sentence Type No. 1

Actual string: the boy likes the girl

Derived string: the boy like s the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[[DEF \left[\begin{smallmatrix} \text{boy} \\ [+Sg] \end{smallmatrix} \right]]_N \left[\begin{smallmatrix} [+Sg] PRES]_T \end{smallmatrix} \right]_V \left[\begin{smallmatrix} \text{like} \\ [+V] \end{smallmatrix} \right]_V DEF [girl]_N]_S$
2. $[DEF [boy]_N \left[\begin{smallmatrix} \text{like} \\ [+V] \end{smallmatrix} \right]_V \left[\begin{smallmatrix} [+Sg] PRES]_T \end{smallmatrix} \right]_V DEF [girl]_N]_S$

T:AG

T:AF

Post Cycle:

1. $[DEF [boy]_N \left[\begin{smallmatrix} \text{like} [s]_T \end{smallmatrix} \right]_V DEF [girl]_N]_S$
2. $[the [boy]_N [like s]_V the [girl]_N]_S$

T:NUAG

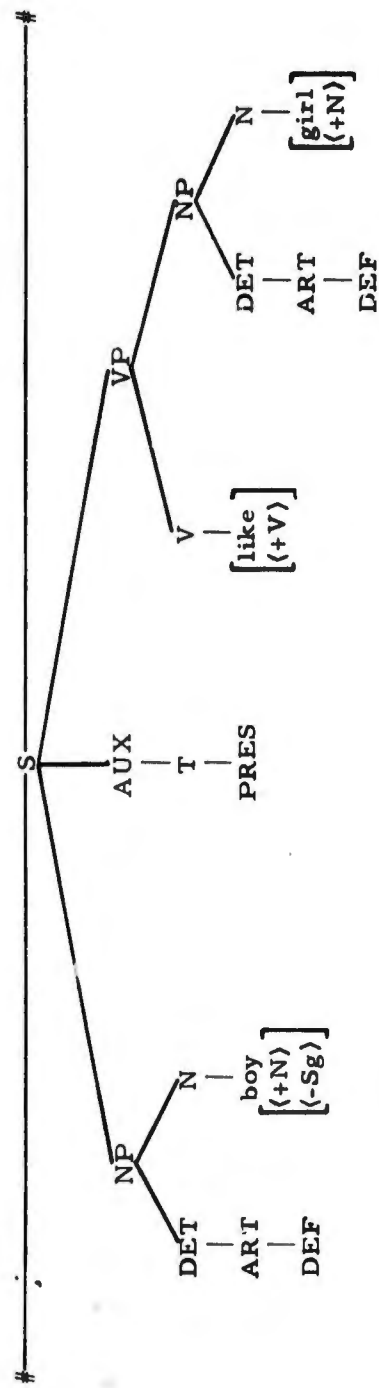
T:DEF

Sentence Type No. 2

Actual string: the boys like the girl

Derived string: the boy s like the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[DEF [boy]_{(-Sg)}]_N]_{NP} [[(-Sg) PRES]_T]_{VP} [like]_{(+V)} [DEF [girl]_{(+N)}]_N]_S$ T:AG
2. $[DEF [boy]_N]_{NP} [[like]_{(+V)} [(-Sg) PRES]_T]_{VP} [DEF [girl]_{(+N)}]_N]_S$ T:AF

Post Cycle:

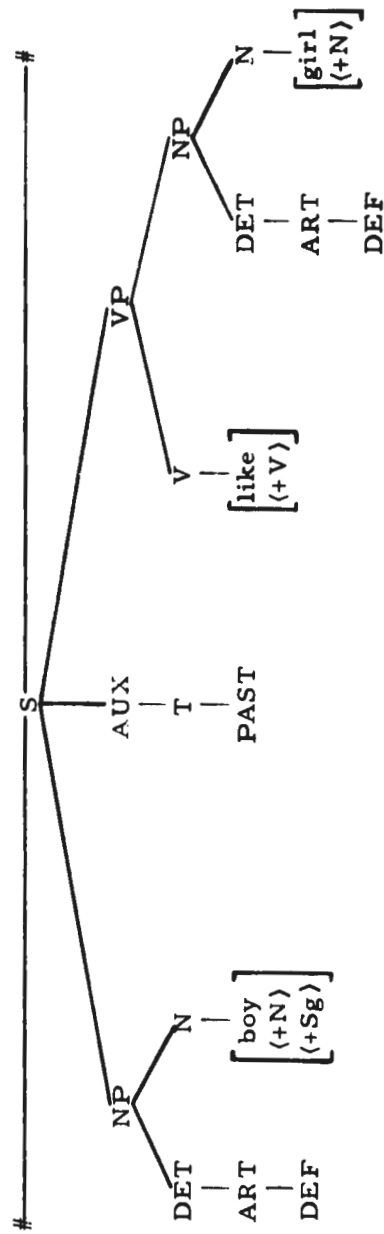
1. $[DEF [boy]_{(-Sg)}]_N [like]_{(+V)} [DEF [girl]_{(+N)}]_N]_S$ T:PLUDEL
2. $[DEF [boy + s]_{(-Sg)}]_N [like]_{(+V)} [DEF [girl]_{(+N)}]_N]_S$ T:NUM
3. $[the [boy s]_N [like]_{(+V)} the [girl]_{(+N)}]_S$ T:DEF

Sentence Type No. 3

Actual string: the boy liked the girl

Derived string: the boy like ed the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[\text{DEF } [\text{boy}]_{\text{N}}]_{\text{NP}} [[+ \text{Sg}]_{\text{PAST}}]_{\text{T}} [\text{like}]_{\text{V}}]_{\text{VP}} \text{ DEF } [\text{girl}]_{\text{N}}]_{\text{S}}$
2. $[\text{DEF } [\text{boy}]_{\text{N}} [\text{like } [[+ \text{Sg}]_{\text{PAST}}]_{\text{T}}]_{\text{VP}}]_{\text{VP}} \text{ DEF } [\text{girl}]_{\text{N}}]_{\text{S}}$

T:AG

T:AF

Post Cycle:

1. $[\text{DEF } [\text{boy}]_{\text{N}} [\text{like } [\text{ed}]_{\text{T}}]_{\text{VP}}]_{\text{VP}} \text{ DEF } [\text{girl}]_{\text{N}}]_{\text{S}}$
2. $[\text{the } [\text{boy}]_{\text{N}} [\text{like ed}]_{\text{VP}}]_{\text{VP}} \text{ the } [\text{girl}]_{\text{N}}]_{\text{S}}$

T:PAST

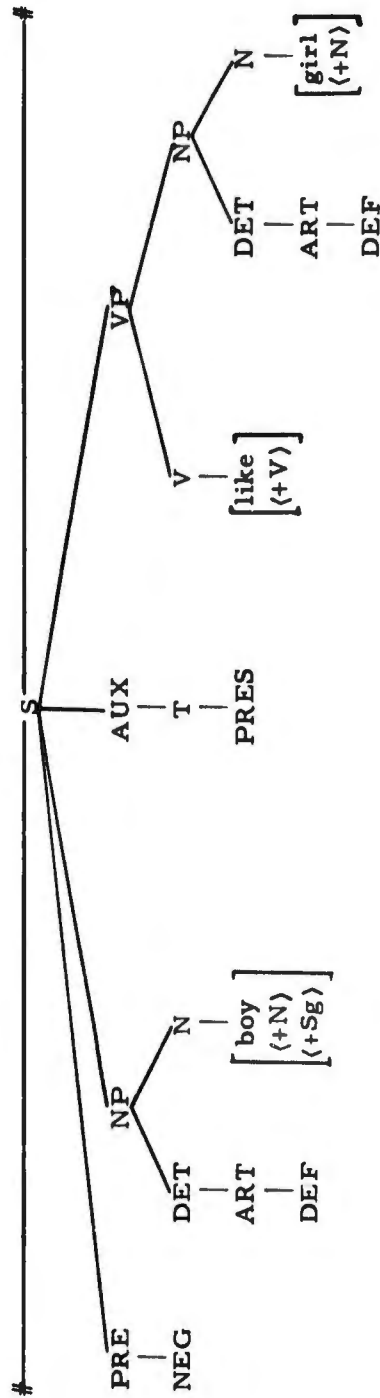
T:DEF

Sentence Type No. 4

Actual string: the boy does not like the girl

Derived string: the boy does not like the girl

Underlying structure:



Derivation:

First Cycle:

1. [NEG [DEF [boy]_NNP]NP]NP [[[+Sg]PRES]_TAUX] like_V DEF [girl]_NS

2. [[[DEF [boy]_NNP]NP]NP]NP [[[+Sg]PRES]_TAUX] NEG [like]_V DEF [girl]_NS

Post Cycle:

1. [[[DEF [boy]_NNP]NP]NP]NP [[[+Sg]PRES]_TAUX] not [like]_V DEF [girl]_NS

2. [[[DEF [boy]_NNP]NP]NP]NP [does]_{AUX} not [like]_V DEF [girl]_NS

3. [the [boy]_N]N [does]_{AUX} not [like]_V the [girl]_NS

T:AG

T:NEGPLACE

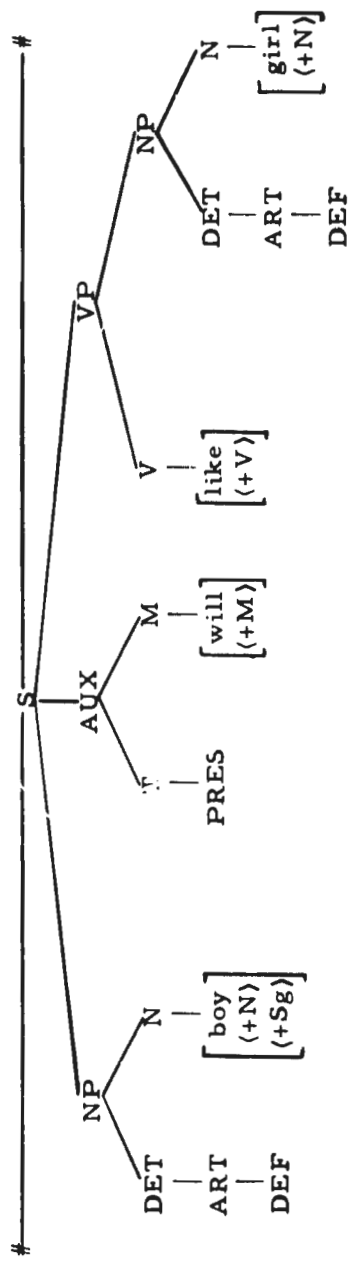
T:NEGSPELL

T:DO 2

T:DEF

Actual string: the boy will like the girl
 Derived string: the boy will like the girl

Underlying structure:



Derivation:

First Cycle:

1. $[(DEF \left[\begin{smallmatrix} \text{boy} \\ (+Sg) \end{smallmatrix} \right]_N]_{NP} \left[\begin{smallmatrix} [+Sg] \\ PRES \end{smallmatrix} \right]_T \left[\begin{smallmatrix} will \\ (+M) \end{smallmatrix} \right]_M \left[\begin{smallmatrix} like \\ V \end{smallmatrix} \right]_{DEF} \left[\begin{smallmatrix} girl \\ N \end{smallmatrix} \right]_S]$ T:AG
2. $[DEF \left[\begin{smallmatrix} boy \\ N \end{smallmatrix} \right]_N \left[\begin{smallmatrix} will \\ (+M) \end{smallmatrix} \right]_M \left[\begin{smallmatrix} [+Sg] \\ PRES \end{smallmatrix} \right]_T \left[\begin{smallmatrix} like \\ V \end{smallmatrix} \right]_{DEF} \left[\begin{smallmatrix} girl \\ N \end{smallmatrix} \right]_S]$ T:AF

Post Cycle:

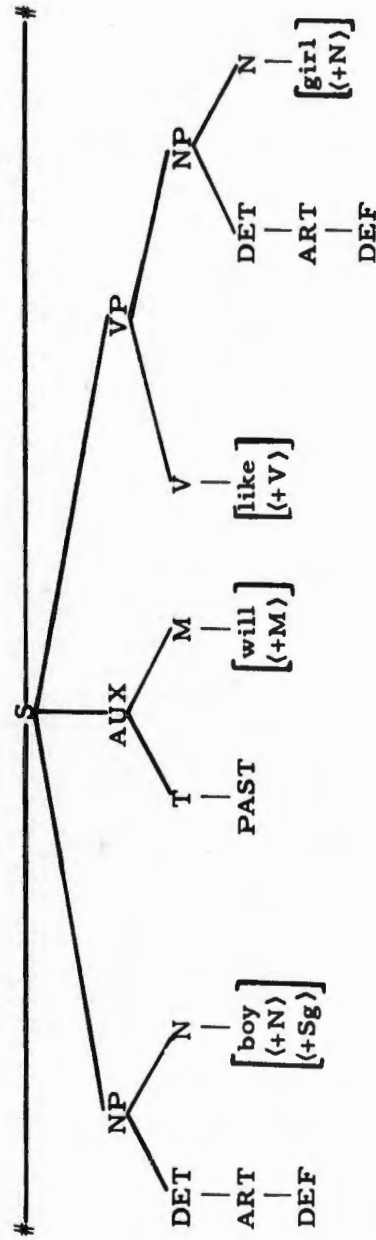
1. $[DEF \left[\begin{smallmatrix} boy \\ N \end{smallmatrix} \right]_N \left[\begin{smallmatrix} will \\ (+M) \end{smallmatrix} \right]_M \left[\begin{smallmatrix} like \\ V \end{smallmatrix} \right]_{DEF} \left[\begin{smallmatrix} girl \\ N \end{smallmatrix} \right]_S]$ T:MTDEL
2. $[the \left[\begin{smallmatrix} boy \\ N \end{smallmatrix} \right]_N \left[\begin{smallmatrix} will \\ M \end{smallmatrix} \right]_M \left[\begin{smallmatrix} like \\ V \end{smallmatrix} \right]_{the} \left[\begin{smallmatrix} girl \\ N \end{smallmatrix} \right]_S]$ T:DEF

Sentence Type No. 6

Actual string: the boy would like the girl

Derived string: the boy will ed like the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[\text{DEF} \left[\begin{smallmatrix} \text{boy} \\ (+Sg) \end{smallmatrix} \right]]_{NP} [\text{PAST}]_T [\text{will}]_{(+M)}]_M [\text{like}]_V \text{DEF} [\text{girl}]_{N'}]_S$ T:AG
2. $[\text{DEF} [\text{boy}]_N [\text{will} [(+Sg) \text{PAST}]_T]_{(+M)}]_M [\text{like}]_V \text{DEF} [\text{girl}]_{N'}]_S$ T:AF

Post Cycle:

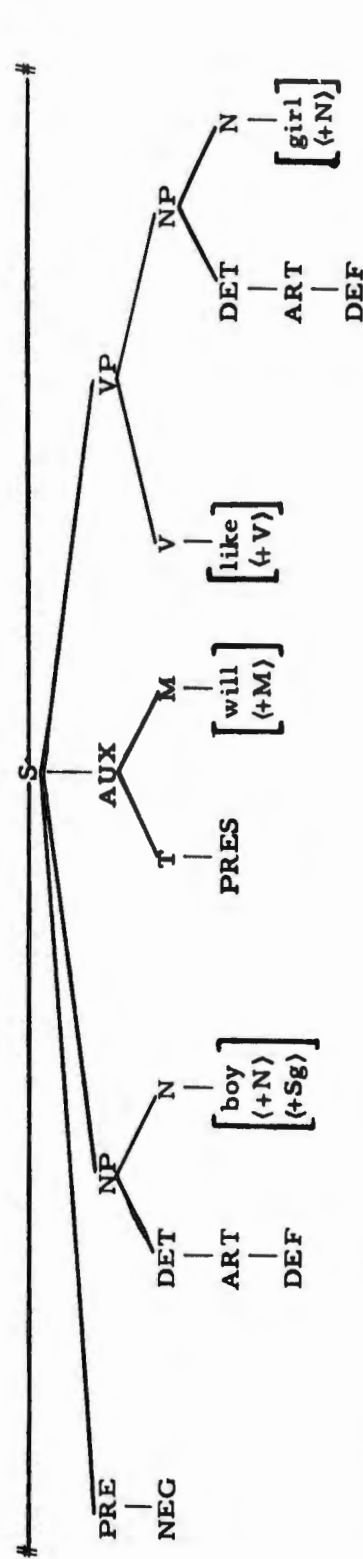
1. $[\text{DEF} [\text{boy}]_N [\text{will} [\text{ed}]_T]_{(+M)}]_M [\text{like}]_V \text{DEF} [\text{girl}]_{N'}]_S$ T:PAST
2. $[\text{the} [\text{boy}]_N [\text{will ed}]_M [\text{like}]_V \text{the} [\text{girl}]_{N'}]_S$ T:DEF

Sentence Type No. 7

Actual string: the boy will not like the girl

Derived string: the boy will not like the girl

Underlying structure:



Derivation:

First Cycle:

1. [NEG [DEF [boy]_N]_{NP}]_N [[+Sg]_{PRES}]_T]_M]_{AUX} [like]_V DEF [girl]_N]_S T:AG
2. [DEF [boy]_N]_N [[+Sg]_{PRES}]_T]_M]_{AUX} NEG [like]_V DEF [girl]_N]_S T:NEGPLACE
3. [DEF [boy]_N]_N [[+Sg]_{PRES}]_T]_M]_{AUX} [like]_V DEF [girl]_N]_S T:AF

Post Cycle:

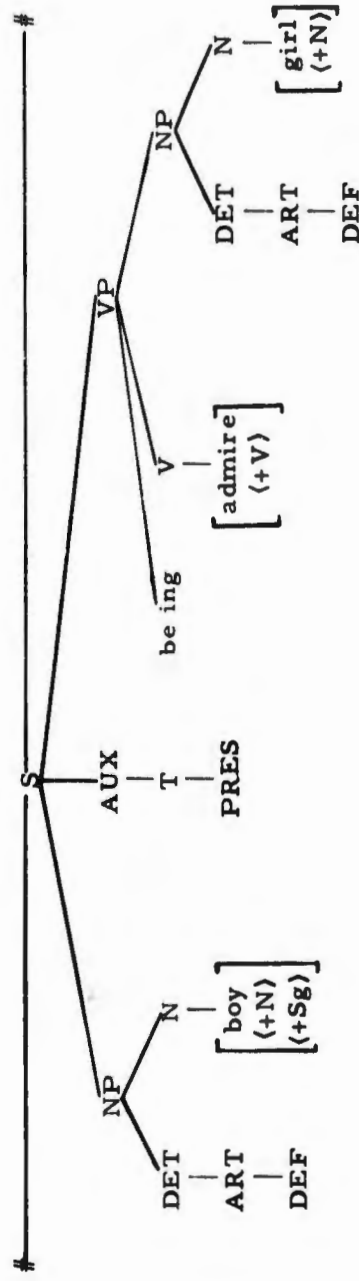
1. [DEF [boy]_N]_N [will]_M]_M]_{AUX} NEG [like]_V DEF [girl]_N]_S T:MTDEL
2. [DEF [boy]_N]_N [will]_M]_M]_{AUX} not [like]_V DEF [girl]_N]_S T:NEGSPELL
3. [the [boy]_N]_N [will]_M]_M]_{AUX} not [like]_V the [girl]_N]_S T:DEF

Sentence Type No. 8

Actual string: the boy is admiring the girl

Derived string: the boy is admire ing the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[[DEF \text{ [boy]}_N]_{NP} \text{ ing } [[PRES]_T \text{ be}]_{AUX} \text{ ing } [admire]_{(V)}]_V \text{ DEF } [girl]_{N'S}]_{T:AUXFILL}$
2. $[[[DEF \text{ [boy]}_N]_{NP} \text{ ing } [[[[+Sg]PRES]_T \text{ be}]_{AUX} \text{ ing } [admire]_{(V)}]_V \text{ DEF } [girl]_{N'S}]_{T:AG} \text{ DEF } [girl]_{N'S}]_{T:AF}$
3. $[[DEF \text{ [boy]}_N]_{NP} \text{ ing } [[be + [[+Sg]PRES]_T]_{AUX} \text{ ing } [admire + ing]_{(V)}]_V \text{ DEF } [girl]_{N'S}]_{T:BE 1}$

Post Cycle:

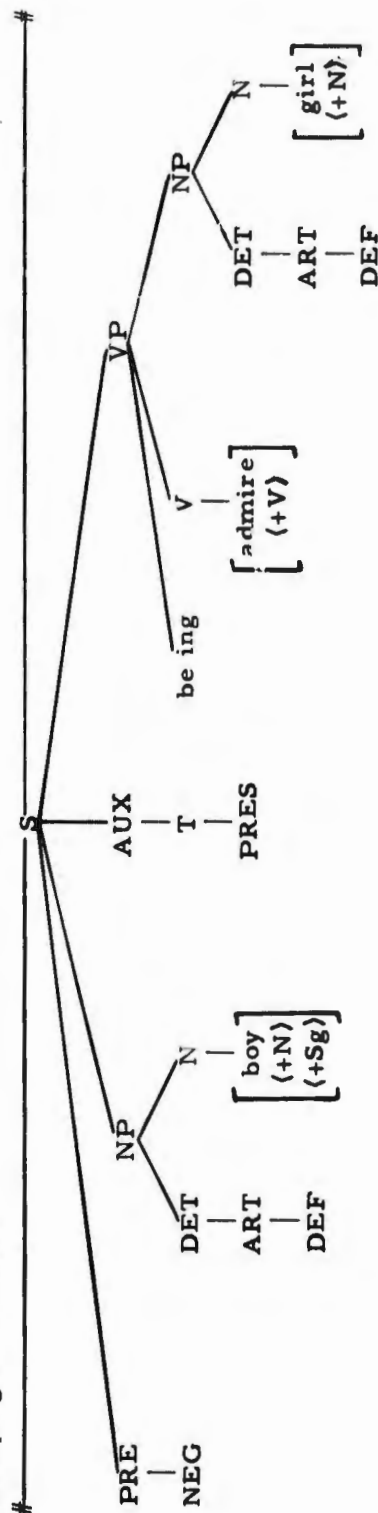
1. $[[DEF \text{ [boy]}_N]_{NP} \text{ ing } [is]_{AUX} \text{ ing } [admire]_{(V)} \text{ DEF } [girl]_{N'S}]_{T:DEF}$
2. $[[the \text{ [boy]}_N]_{NP} \text{ ing } [is]_{AUX} \text{ ing } [admire]_{(V)} \text{ the } [girl]_{N'S}]_{T:DEF}$

Sentence Type No. 9

Actual string: the boy isn't admiring the girl

Derived string: the boy is n't admire ing the girl

Underlying structure:



Derivation:

First Cycle:

1. [NEG[DEF [boy]N]NP [[PRES]T be]AUX ing [admire]V DEF [girl]N'S] T:AUXFILL
2. [NEG [DEF [boy]N]NP [[[+Sg]PRES]T be]AUX ing [admire]V DEF [girl]N'S] T:AG
3. [DEF [boy]N [[[+Sg]PRES]T be]AUX NEG ing [admire]V DEF [girl]N'S] T:NEGPLACE
4. [DEF [boy]N [[[+Sg]PRES]T be NEG]AUX ing [admire]V DEF [girl]N'S] T:NEGAUX
5. [DEF [boy]N [be + [[+Sg]PRES]T NEG]AUX [admire + ing]V DEF [girl]N'S] T:AF

Post Cycle:

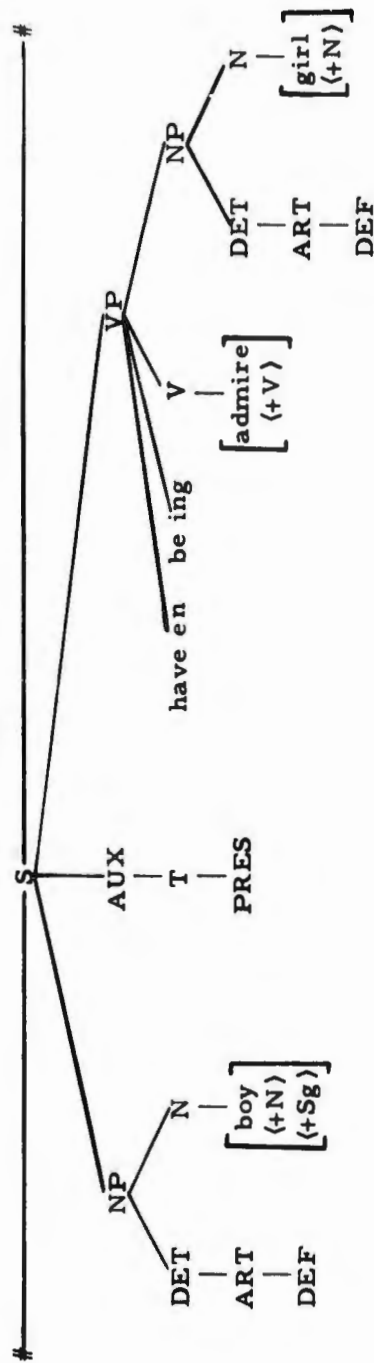
1. [DEF [boy]N [be [[+Sg]PRES]T n't]AUX [admire ing]V DEF [girl]N'S] T:CONTR
2. [DEF [boy]N [[is]T n't]AUX [admire ing]V DEF [girl]N'S] T:BE 1
3. [the [boy]N [[is]T n't]AUX [admire ing]V the [girl]N'S] T:DEF

Sentence Type No. 10

Actual string: the boy has been admiring the girl

Derived string: the boy has been admire ing the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[DEF [boy]_{N}]]_{NP} [[PRES]_T have]_{AUX} en be ing [admire]_V DEF [girl]_{N}S$ T:AUXFILL
2. $[[DEF [boy]_{N}]]_{NP} [[[+Sg]PRES]_T have]_{AUX} en be ing [admire]_{N}S$ T:AG
3. $[DEF [boy]_{N} have + [[+Sg]PRES]_T]_{AUX} be + en [admire + ing]_{N}S$ T:AF

Post Cycle:

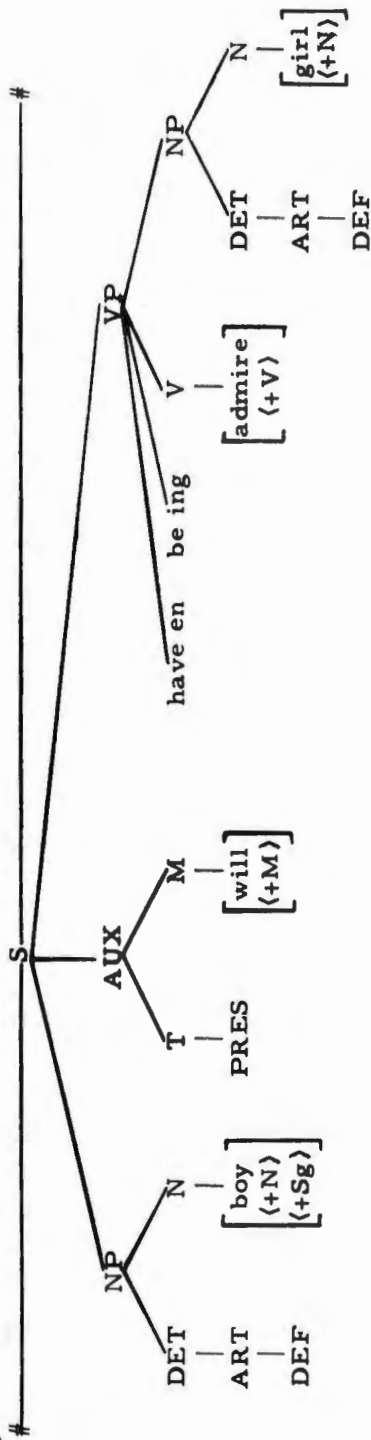
1. $[DEF [boy]_{N} [has]_{AUX} be en [admire ing]_V DEF [girl]_{N}S]$ T:HAVE 1
2. $[the [boy]_{N} [has]_{AUX} be en [admire ing]_V the [girl]_{N}S]$ T:DEF

Sentence Type No. 11

Actual string: the boy will have been admiring the girl

Derived string: the boy will have been admire ing the girl

Underlying structure:



Derivation:

First Cycle:

1. $[[DEF [boy]_N]_{NP} [[+Sg] PRES]_T]_{AUX} have\ en\ be\ ing [admire [girl]_N]_S$ T:AG
2. $[DEF [boy]_N]_{NP} [will + [+Sg] PRES]_T [have\ be + en [admire + ing [girl]_N]_S]_V$ T:AF

Post Cycle:

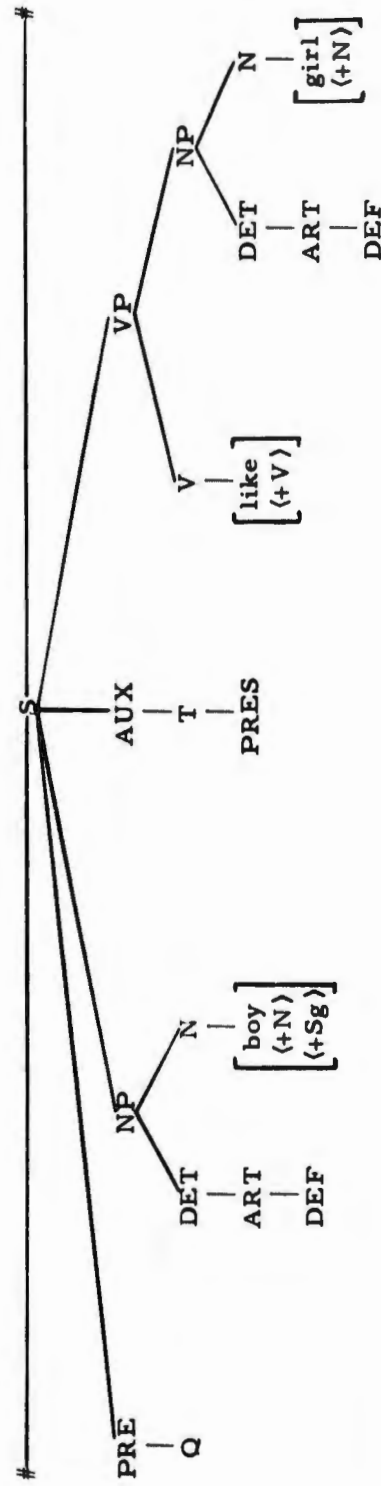
1. $[DEF [boy]_N]_{NP} [[will]_M]_{AUX} have\ be\ en [admire\ ing]_V DEF [girl]_N]_S$ T:MTDEL
2. $[the [boy]_N]_{NP} [will]_M have\ be\ en [admire\ ing]_V the [girl]_N]_S$ T:DEF

Sentence Type No. 12

Actual string: does the boy like the girl

Derived string: does the boy like the girl

Underlying structure:



Derivation:

First Cycle:

1. [Q[DEF [boy] [+Sg]]]_N NP [[+Sg]PRES]_TAUX [like]_V DEF [girl]_NS T:AG
2. [[[[+Sg]PRES]_TAUX Q [DEF [boy]_NNP [like]_V DEF [girl]_NS] T:YESNO
3. [[[[+Sg]PRES]_TAUX DEF [boy]_N [like]_V DEF [girl]_NS] T:QDEL

Post Cycle:

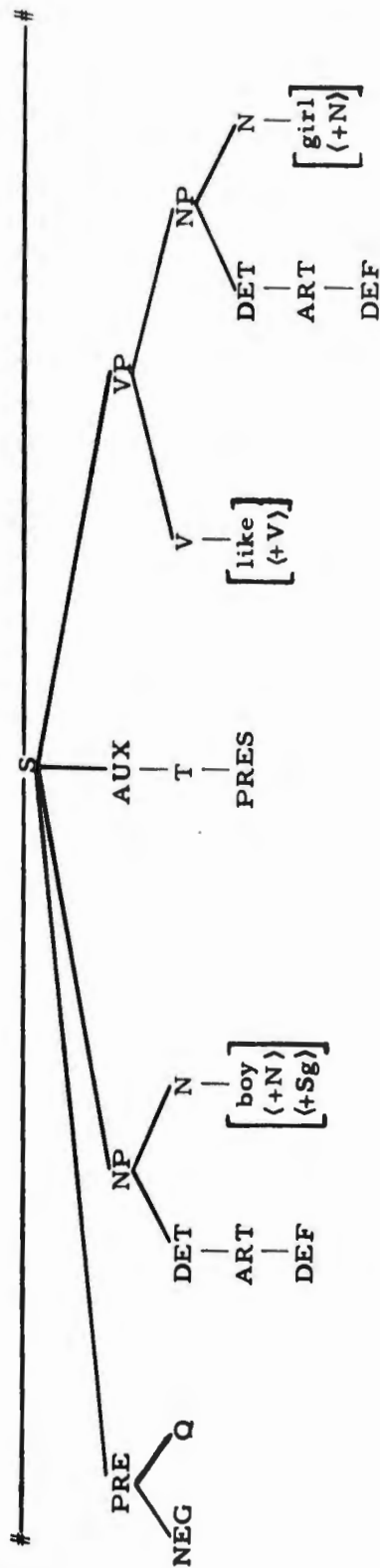
1. [[does]_{AUX} DEF [boy]_N [like]_V DEF [girl]_NS] T:DO 2
2. [[does]_{AUX} the [boy]_N [like]_V the [girl]_NS] T:DEF

Sentence Type No. 13

Actual string: doesn't the boy like the girl

Derived string: does n't the boy like the girl

Underlying structure:



Derivation:

First Cycle:

1. [NEG Q [DEF [boy] NP] NP] NP [[+Sg] PRES] T AUX [like] V DEF [girl] N] S T:AG
2. [Q [DEF [boy] NP] NP] NP [[+Sg] PRES] T AUX NEG [like] V DEF [girl] N] S T:NEGPLACE
3. [Q [DEF [boy] NP] NP] NP [[+Sg] PRES] T NEG] AUX [like] V DEF [girl] N] S T:NEGAUX
4. [[[+Sg] PRES] T NEG] AUX Q DEF [boy] NP [like] V DEF [girl] N] S T:YESNO
5. [[[+Sg] PRES] T NEG] AUX DEF [boy] NP [like] V DEF [girl] N] S T:QDEL

Post Cycle:

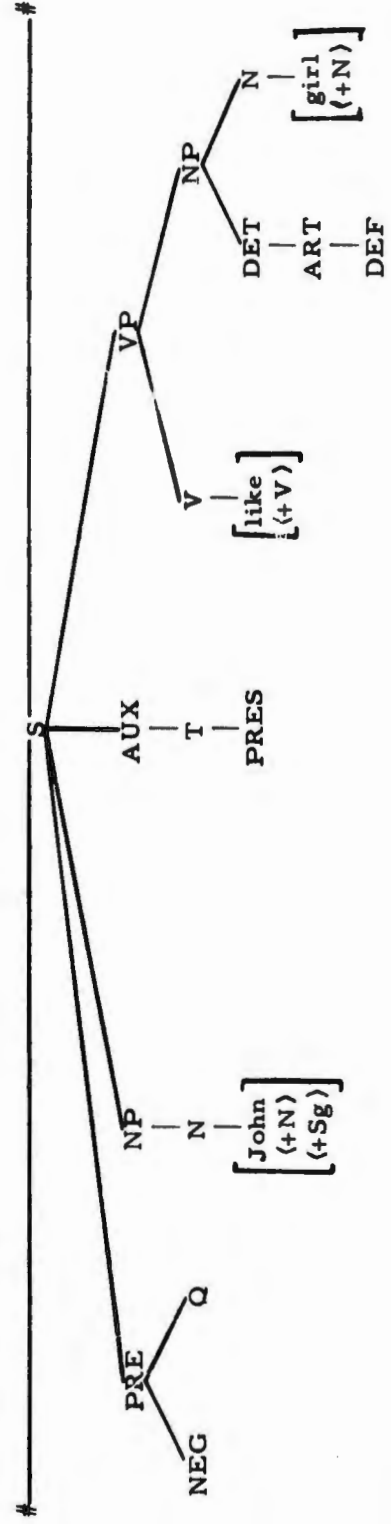
1. [[[+Sg] PRES] T n't] AUX DEF [boy] NP [like] V DEF [girl] N] S T:CONTR
2. [[[does] T n't] AUX DEF [boy] NP [like] V DEF [girl] N] S T:DO 2
3. [[[does] T n't] AUX the [boy] NP [like] V the [girl] N] S T:DEF

Sentence Type No. 14

Actual string: John likes the girl doesn't he

Derived string: John like s the girl does n't John

Underlying structure:



Derivation:

First Cycle:

1. [NEG Q [[John] NP [[+Sg] PRES] T] AUX [like] V DEF [girl] NP S T:AG
2. [NEG [[John] NP [[+Sg] PRES] T] AUX [like] V DEF [girl] NP Q [[John] NP NP [[+Sg] PRES] T] AUX S T:TAG
3. [[[John] NP [[+Sg] PRES] T] AUX NEG [like] V DEF [girl] NP Q [[John] NP NP [[+Sg] PRES] T] AUX S T:NEGPLACE
4. [[[John] NP [[+Sg] PRES] T] AUX [[like] V DEF [girl] NP VP Q [[John] NP NP [[+Sg] PRES] T] AUX NEG S T:NEGTAG

5. $[[John]_N [[+Sg]PRES]_T]_{AUX} [like]_V DEF [girl]_N Q [[John]_N]_{NP} [[+Sg]PRES]_T NEG]_{AUX} S$ T:NEGAUX
6. $[[John]_N [[+Sg]PRES]_T]_{AUX} \left[\begin{array}{c} like \\ (+V) \end{array} \right]_V DEF [girl]_N [[+Sg]PRES]_T NEG]_{AUX} Q [[John]_N]_{NP} S$ T:YESNO
7. $[[John]_N \left[\begin{array}{c} like [[+Sg]PRES]_T \\ (+V) \end{array} \right]_V DEF [girl]_N [[+Sg]PRES]_T NEG]_{AUX} Q [John]_N S$ T:AF
8. $[[John]_N \left[\begin{array}{c} like [[+Sg]PRES]_T \\ (+V) \end{array} \right]_V DEF [girl]_N [[+Sg]PRES]_T NEG]_{AUX} [John]_N S$ T:QDEL

Post Cycle:

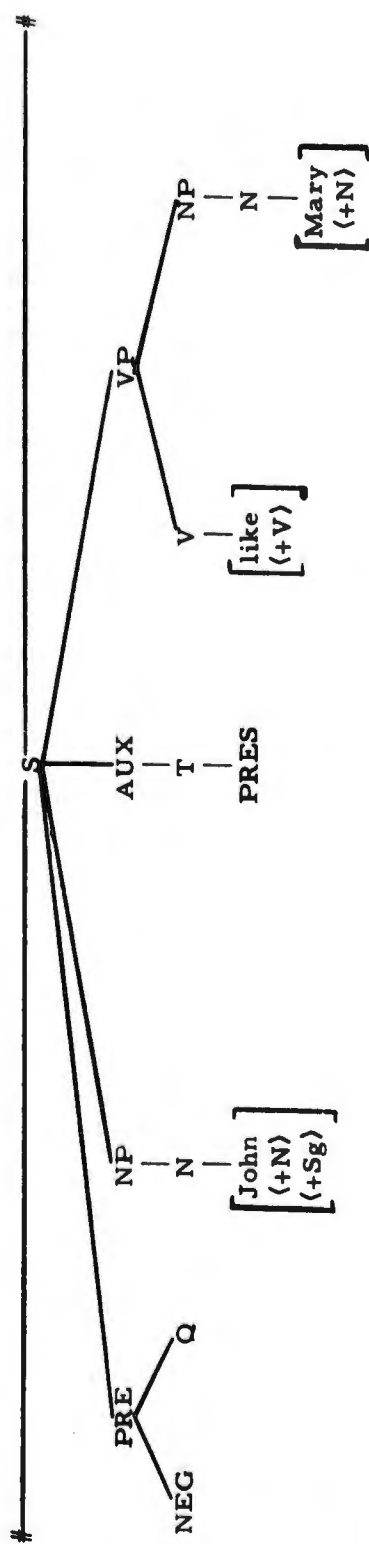
1. $[[John]_N [like[s]_T]_V DEF [girl]_N [[+Sg]PRES]_T NEG]_{AUX} [John]_N S$ T:NUAG
2. $[[John]_N [like s]_V DEF [girl]_N [[+Sg]PRES]_T n't]_{AUX} [John]_N S$ T:CONTR
3. $[[John]_N [like s]_V DEF [girl]_N [[does]_T n't]_{AUX} [John]_N S$ T:DO 2
4. $[[John]_N [like s]_V the [girl]_N [[does]_T n't]_{AUX} [John]_N S$ T:DEF

Sentence Type No. 15

Actual string: John does not like Mary does he

Derived string: John does not like Mary does John

Underlying structure:



Derivation:

First Cycle:

1. [NEG Q [[John] NP [[[+Sg] PRES] T] AUX [like] V [Mary] N] S T:AG
2. [NEG [[John] N] NP [[[+Sg] PRES] T] AUX [like] V [Mary] N Q [[John] N] NP [[[+Sg] PRES] T] AUX S T:TAG
3. [[[John] N] NP [[[+Sg] PRES] T] AUX NEG [like] V [Mary] N Q [[John] N] NP [[[+Sg] PRES] T] AUX S T:NEGPLACE
4. [[[John] N] [[[+Sg] PRES] T] AUX NEG [like] V [Mary] N [[[+Sg] PRES] T] AUX Q [[John] N] NP S T:YESNO
5. [[[John] N] [[[+Sg] PRES] T] AUX NEG [like] V [Mary] N [[[+Sg] PRES] T] AUX [John] N] S T:QDEL

Post Cycle:

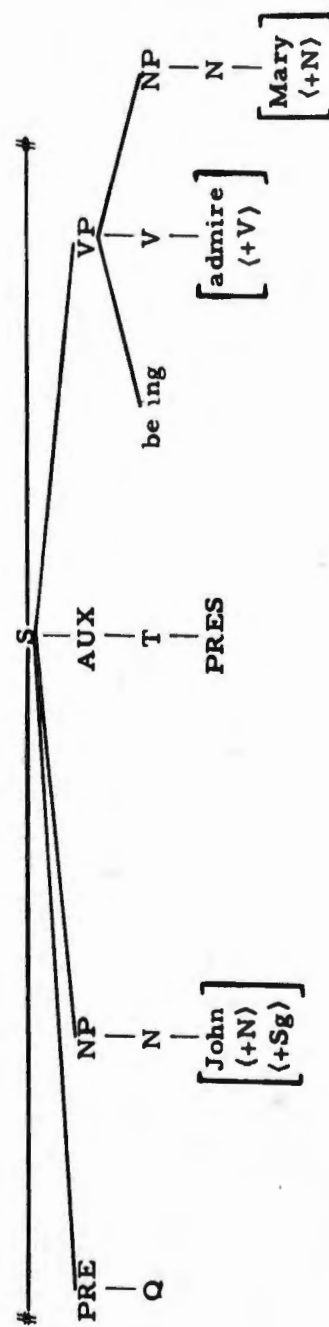
1. [[John] N [[[+Sg] PRES] T] AUX not [like] V [Mary] N [[[+Sg] PRES] T] AUX [John] N] S T:NEGSPELL
2. [[[John] N] [does] AUX not [like] V [Mary] N does [John] N] S T:DO 2

Sentence Type No. 16

Actual string: is John admiring Mary

Derived string: is John admire ing Mary

Underlying structure:



Derivation:

First Cycle:

1. $[Q [[John]]_{NP} [[PRES]_T + be]_{AUX} ing [admire]_V [Mary]_{N'}S]$ T:AUXFILL
2. $[Q [[John]]_{NP} [[[+Sg]PRES]_T be]_{AUX} ing [admire]_V [Mary]_{N'}S]$ T:AG
3. $[[[[+Sg]PRES]_T be]_{AUX} Q [[John]_{N'}NP ing [admire] [Mary]_{N'}S]]_V$ T:YESNO
4. $[[[be + [[[+Sg]PRES]_T]_{AUX} Q [John]_N [admire + ing] [Mary]_{N'}S]]_V]$ T:AF
5. $[[[be [[[+Sg]PRES]_T]_{AUX} [John]_N [admire ing]_V [Mary]_{N'}S]]]$ T:QDEL

Post Cycle:

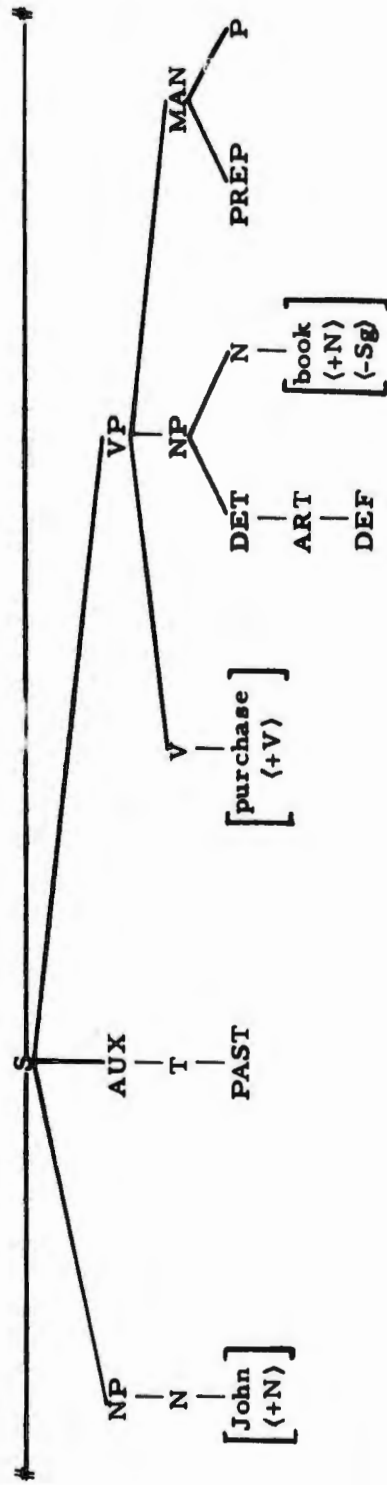
1. $[[[is]_{AUX} [John]_N [admire ing]_V [Mary]_{N'}S]]$ T:BE 1

Sentence Type No. 17

Actual string: the books were purchased by John

Derived string: the book s were purchase en by John

Underlying structure:



Derivation:

First Cycle:

1. [[DEF [book]_NNP] [[PAST]_TAUX be + en + [purchase]_V [PREP [[John]_NNP]MAN]_S T:PASSIVE
2. [[DEF [book]_NNP] [[PAST]_T + be]_{AUX} en [purchase]_V [PREP [[John]_NNP]MAN]_S T:AUXFILL
3. [[DEF [book]_NNP] [[[-Sg]PAST]_T be]_{AUX} en [purchase]_V [PREP [[John]_NNP]MAN]_S T:AG
4. [DEF [book]_NNP] [be + [[[-Sg]PAST]_TAUX [purchase + en]_V] [PREP [[John]_NNP]MAN]_S T:AF

Post Cycle:

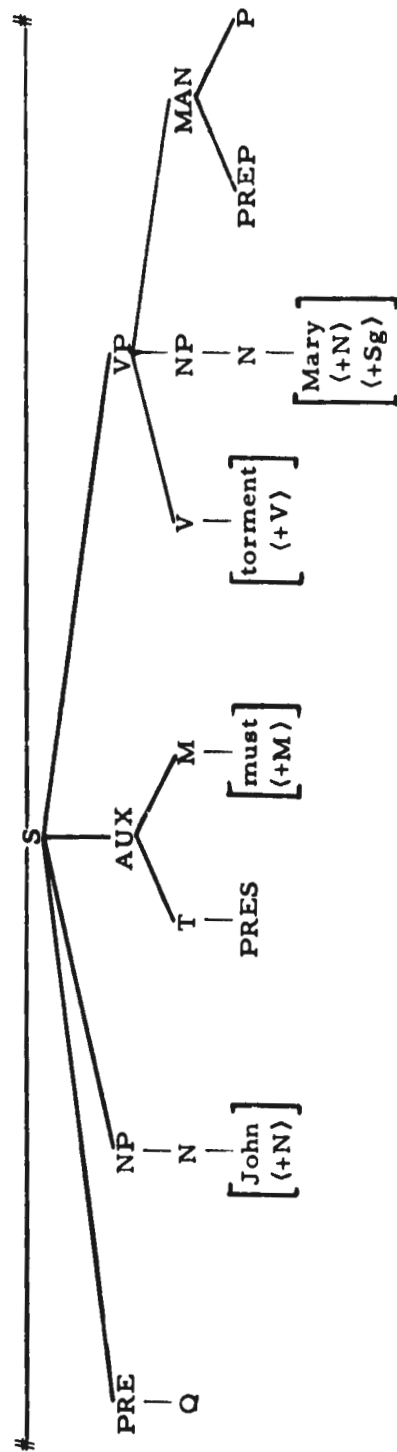
1. [DEF [book + s]_N] [be [[[-Sg]PAST]_TAUX [purchase en]_V [PREP [[John]_NNP]MAN]_S T:NUM
2. [DEF [book s]_N] [were]_{AUX} [purchase en]_V [PREP [[John]_NNP]MAN]_S T:BE 4
3. [DEF [book s]_N] [were]_{AUX} [purchase en]_V [by [[John]_NNP]MAN]_S T:BY
4. [the [book s]_N] [were]_{AUX} [purchase en]_V by [John]_NS T:DEF

Sentence Type No. 18

Actual string: must Mary be tormented by John

Derived string: must Mary be torment en by John

Underlying structure:



Derivation:

First Cycle:

1. $[Q [[Mary]_{NP} [[PRES]_T [must]_M]_{AUX} be + en + [torment]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:PASSIVE
2. $[Q [[Mary]_{NP} [[[+Sg]]_{PRES}]_T [must]_M]_{AUX} be en [torment]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:AG
3. $[[[[[+Sg]]_{PRES}]_T [must]_M]_{AUX} Q [[Mary]_{NP}]_{NP} be en [torment]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:YESNO
4. $[[[[must + [[+Sg]]_{PRES}]_T]_{AUX} Q [[Mary]_{NP}]_{NP} be [torment + en]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:AF
5. $[[[[must [[+Sg]]_{PRES}]_T]_{AUX} [Mary]_{NP} be [torment en]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:QDEL

Post Cycle:

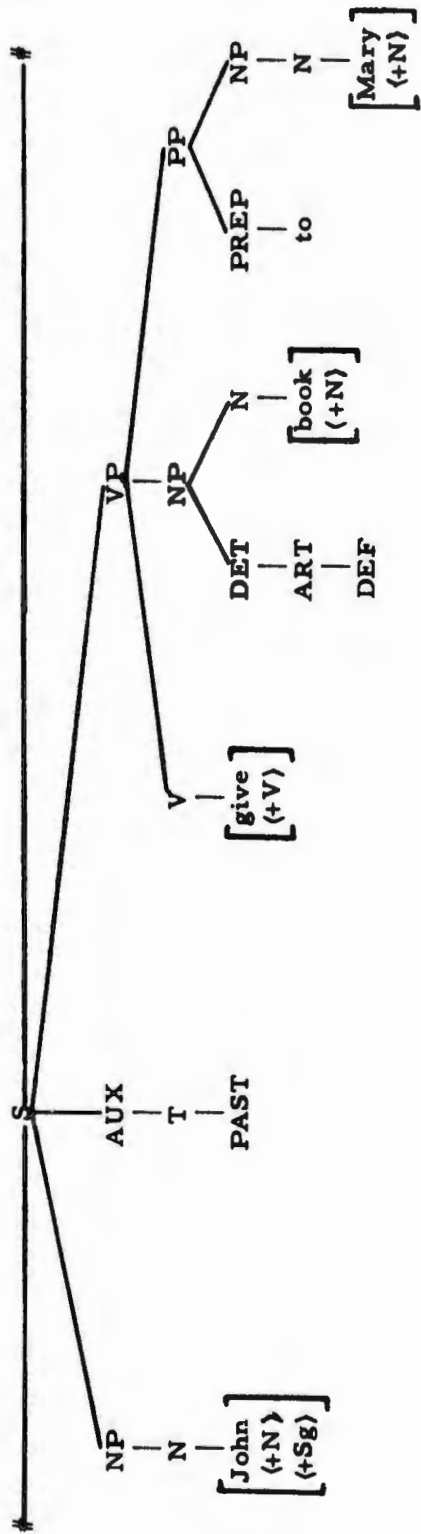
1. $[[[[must]_M]_{AUX} [Mary]_{NP} be [torment en]_V [PREP [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:MTDEL
2. $[[[[must]_M [Mary]_{NP} be [torment en]_V]_{BY} [[John]_{NP}]_{NP}]_{MAN}]_S]$ T:BY

Sentence Type No. 19

Actual string: John gave the book to Mary

Derived string: John give ed the book to Mary

Underlying structure:



Derivation:

First Cycle:

1. $[[[John]]_{NP} [[+Sg]_{PAST}]_T [give]_{DEF [book]_N to [Mary]_{N'S} }]_V$
2. $[[[John]_N [give [[+Sg]_{PAST}]_T]_V DEF [book]_N to [Mary]_{N'S}]_V$

T:AG

T:AF

Post Cycle:

1. $[[[John]_N [give [ed]]_T]_V DEF [book]_N to [Mary]_{N'S}]_V$
2. $[[[John]_N [give ed]_V the [book]_N to [Mary]_{N'S}]_V$

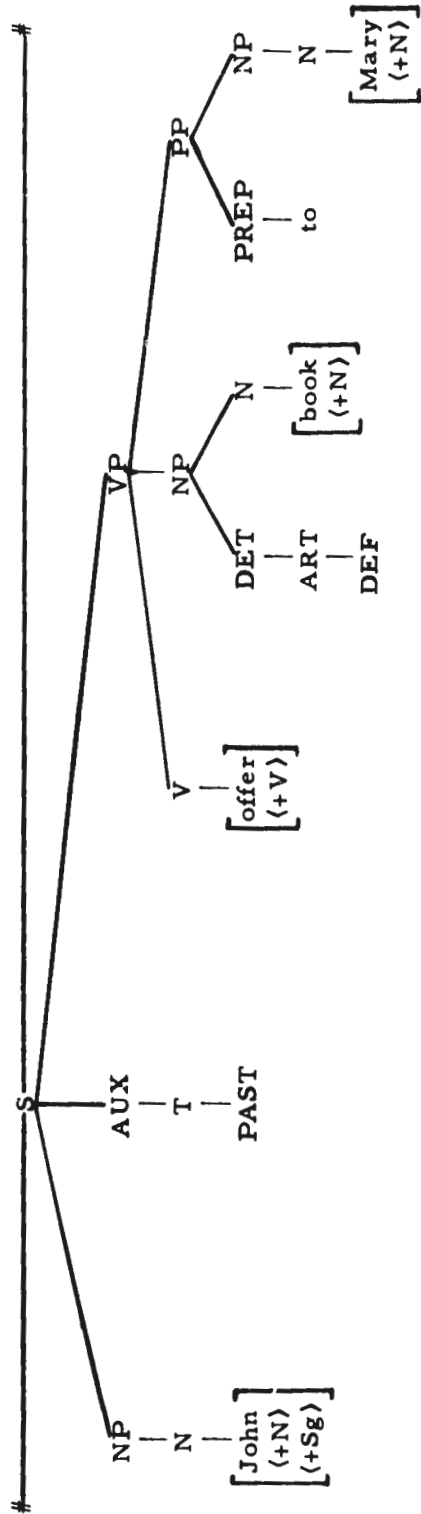
T:PAST

T:DEF

Actual string: John offered Mary the book

Derived string: John offer ed Mary the book

Underlying structure:



Derivation:

First Cycle:

1. $[[[[\text{John}]_{NP} \text{PAST} [\text{offer}]_V [\text{to} [[\text{Mary}]_{NP} \text{PP} [\text{DEF} [\text{book}]_{NP}]_S]]]]_N$ T:IOI
2. $[[[[\text{John}]_{NP} \text{PAST} [\text{offer}]_V [[[\text{Mary}]_{NP} \text{PP} [\text{DEF} [\text{book}]_{NP}]_S]]]]_N$ T:TO
3. $[[[[\text{John}]_{NP} [[[+Sg] \text{PAST}]_T \text{AUX} [\text{offer}]_V [\text{Mary}]_N \text{DEF} [\text{book}]_N]_S]]]_N$ T:AG
4. $[[[\text{John}]_N [\text{offer}]_{(+V)} [[[+Sg] \text{PAST}]_T]_V [\text{Mary}]_N \text{DEF} [\text{book}]_N]_S$ T:AF

Post Cycle:

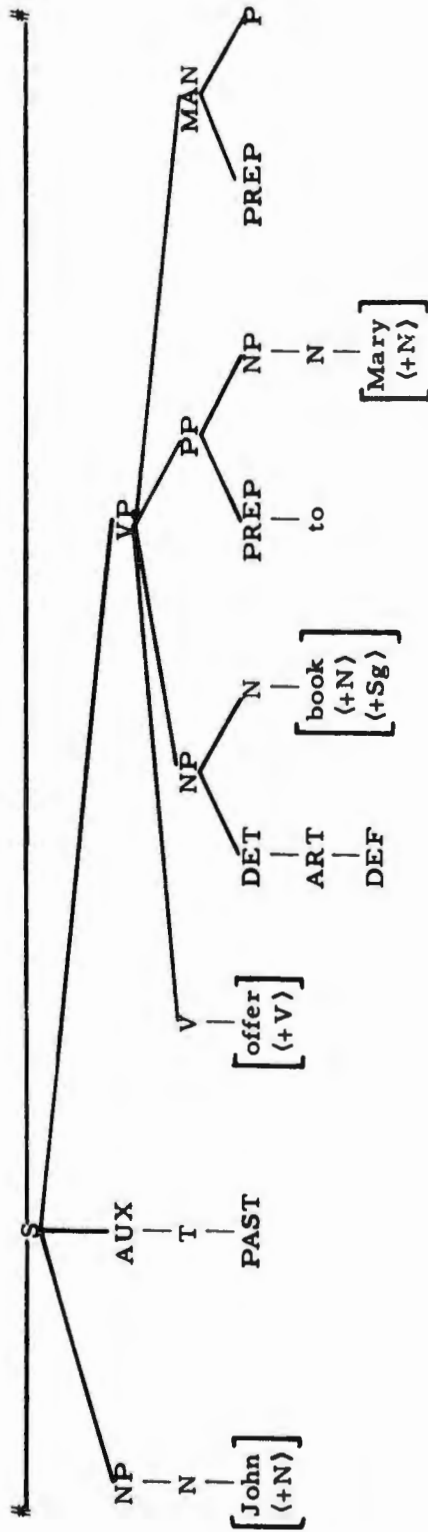
1. $[[[\text{John}]_N [\text{offer}]_{ed}]_T]_V [\text{Mary}]_N \text{DEF} [\text{book}]_N]_S$ T:PAST
2. $[[[\text{John}]_N [\text{offer ed}]_V [\text{Mary}]_N \text{the} [\text{book}]_N]_S$ T:DEF

Sentence Type No. 21

Actual string: the book was offered to Mary by John

Derived string: the book was offer en to Mary by John

Underlying structure:



Derivation:

First Cycle:

1. $[[[DEF \text{ book}]_N]_{NP} [[PAST]_T]_{AUX} be + en + [offer]_V \text{ to } [Mary]_N [PREP [[John]_N]_{NP}]_{MAN}]_S$ T:PASSIVE
2. $[[[DEF [book]_{<+Sg>}]_N]_{NP} [[PAST]_T + be]_{AUX} en [offer]_V \text{ to } [Mary]_N [PREP [[John]_N]_{NP}]_{MAN}]_S$ T:AUXFILL
3. $[DEF [book]_{<+Sg>}]_N [[[[+Sg]_{PAST}]_T be]_{AUX} en [offer]_{<+V>}]_V \text{ to } [Mary]_N [PREP [[John]_N]_{NP}]_{MAN}]_S$ T:AG
4. $[DEF [book]_N [be + [[+Sg]_{PAST}]_T]_{AUX} [offer + en]_{<+V>}]_V \text{ to } [Mary]_N [PREP [[John]_N]_{NP}]_{MAN}]_S$ T:AF

Post Cycle:

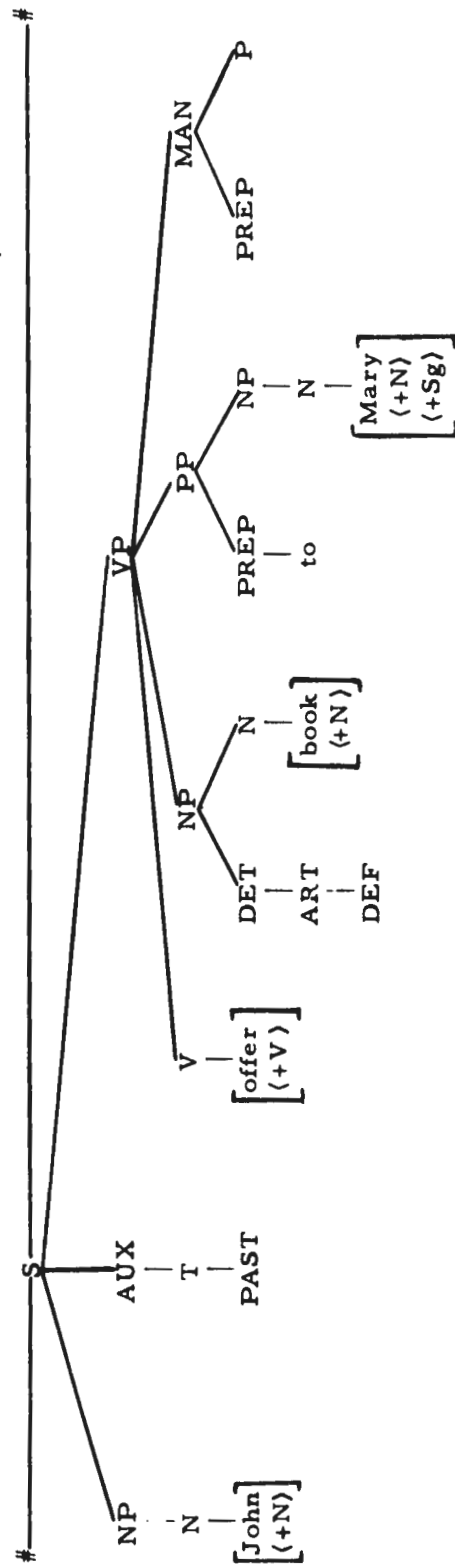
1. $[DEF [book]_N [was]_{AUX} [offer en]_V \text{ to } [Mary]_N [PREP [[John]_N]_{NP}]_{MAN}]_S$ T:BE 3
2. $[DEF [book]_N [was]_{AUX} [offer en]_V \text{ to } [Mary]_N [by [[John]_N]_{NP}]_{MAN}]_S$ T:BY
3. $[the [book]_N [was]_{AUX} [offer en]_V \text{ to } [Mary]_N \text{ by } [John]_N]_S$ T:DEF

Sentence Type No. 22

Actual string: Mary was offered the book by John

Derived string: Mary was offer en the book by John

Underlying structure:



Derivation:

First Cycle:

1. $[[[John]_N \text{ PAST } [offer]_V \text{ to } [[Mary]_N \text{ NP } PP \text{ DEF } [book]_N \text{ NP } PREP } P]_S]$ T:IOI
2. $[[[[John]_N \text{ NP } [[PAST]_T \text{ AUX } [offer]_V \text{ to } [[Mary]_N \text{ NP } [DEF [book]_N \text{ NP } PREP } P]_S]]]]]$ T:TO
3. $[[[[Mary]_N \text{ NP } [[PAST]_T \text{ AUX } be + en + [offer]_V \text{ DEF } [book]_N \text{ NP } [PREP [[John]_N \text{ NP } MAN]_S]]]]]$ T:PASSIVE
4. $[[[[Mary]_N \text{ NP } [[PAST]_T + be]_{AUX} en [offer]_V \text{ DEF } [book]_N \text{ NP } [PREP [[John]_N \text{ NP } MAN]_S]]]]]$ T:AUXFILL
5. $[[[[Mary]_N \text{ NP } [[+Sg]_{PAST} \text{ T } be]_{AUX} en [offer]_V \text{ DEF } [book]_N \text{ NP } [PREP [[John]_N \text{ NP } MAN]_S]]]]]$ T:AG
6. $[[[[Mary]_N \text{ NP } [be + [[+Sg]_{PAST} \text{ T } AUX]_{offer + en}]_V \text{ DEF } [book]_N \text{ NP } [PREP [[John]_N \text{ NP } MAN]_S]]]]]$ T:AF

Post Cycle:

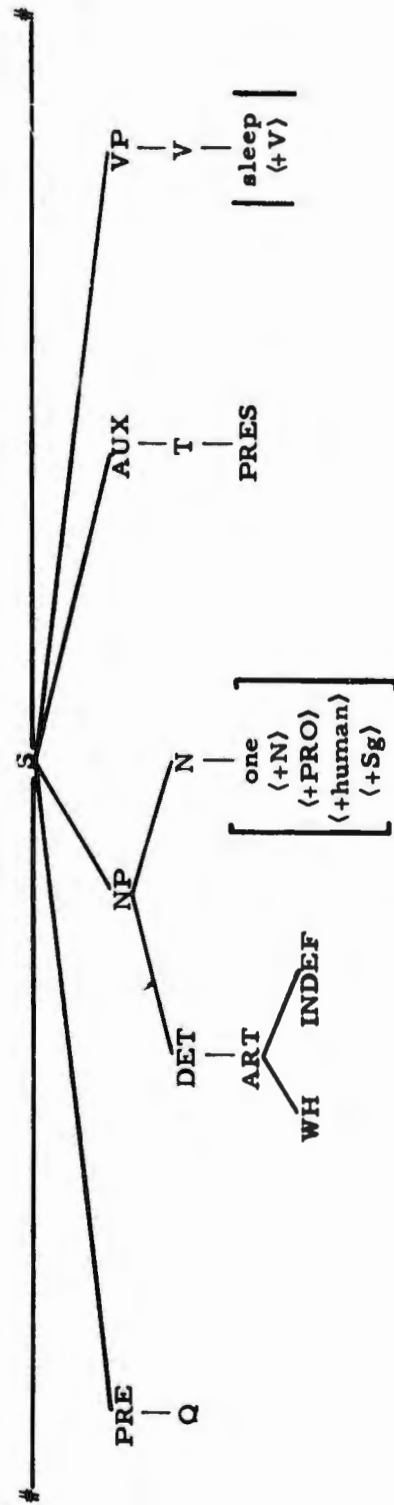
1. $[[[Mary]_N \text{ [was]}_{AUX} [offer en]_V \text{ DEF } [book]_N \text{ NP } [PREP [[John]_N \text{ NP } MAN]_S]]]$ T:BE 3
2. $[[[Mary]_N \text{ [was]}_{AUX} [offer en]_V \text{ DEF } [book]_N \text{ NP } [by [[John]_N \text{ NP } MAN]_S]]]$ T:BY
3. $[[[Mary]_N \text{ [was]}_{AUX} [offer en]_V \text{ the } [book]_N \text{ NP } by [John]_N \text{ NP } MAN]_S]]]$ T:DEF

Sentence Type No. 23

Actual string: who sleeps?

Derived string: who sleep s

Underlying structure:



Derivation:

First Cycle:

1. $[[Q]_{PRE}[[[WH\ INDEF]_{ART}]_{DET} \left[\begin{array}{c} one \\ \langle +Sg \rangle \\ \langle +human \rangle \end{array} \right]_{NP} [+Sg]_{PRES} [sleep]_{V}]_S$ T:AG
2. $[Q [WH [+human]_{INDEF} \left[\begin{array}{c} one \\ \langle +human \rangle \end{array} \right]_{NP} [+Sg]_{PRES} [sleep]_{V}]_S$ T:WHAG
3. $[[[WH [+human]_{INDEF} [one]_{NP} Q [[[+Sg]_{PRES}]_{T} AUX \left[\begin{array}{c} sleep \\ \langle +V \rangle \end{array} \right]_{V}]_S$ T:QUES
4. $[[[WH [+human]_{INDEF} [one]_{NP} Q [sleep [[[+Sg]_{PRES}]_{T}]_S$ T:AF
5. $[[[WH [+human]_{INDEF} [one]_{NP} Q [sleep [[[+Sg]_{PRES}]_{T}]_S$ T:QDEL

Post Cycle:

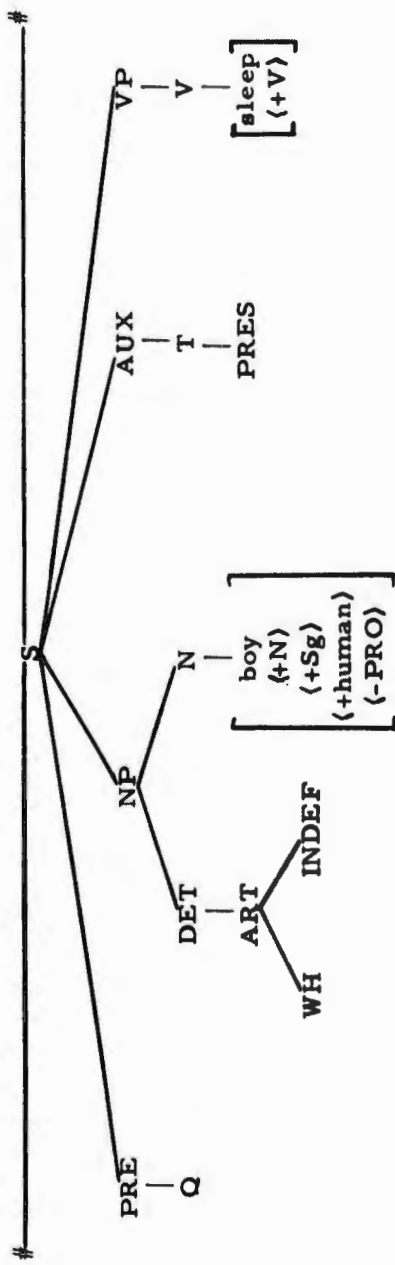
1. $[[[WH [+human]_{INDEF} \left[\begin{array}{c} one \\ \langle +PRO \rangle \\ \langle +Sg \rangle \end{array} \right]_{NP} [sleep[s]]_{T}]_V]_S$ T:NUAG
2. $[[[WH [+human]_{INDEF}]_{NP} [sleep s]_{V}]_S$ T:WHPD 2
3. $[[who]_{NP} [sleep s]_{V}]_S$ T:WH 1

Sentence Type No. 24

Actual string: what boy sleeps?

Derived string: **what boy sleep s**

Underlying structure:



Derivation:

First Cycle:

1.	[Q [WH INDEF [boy (+Sg) (+human)]]] _{NP} [+Sg] PRES [sleep] _V] _S	T:AG
2.	[Q [WH [+human]] INDEF [boy (+human)]] _{NP} [+Sg] PRES [sleep] _V] _S	T:WHAG
3.	[[[WH [+human]] INDEF [boy] _N] _{NP} Q [[[+Sg] PRES] _T] _{AUX} [sleep] _S (+V)]] _V	T:QUES
4.	[[[WH [+human]] INDEF [boy] _N] _{NP} Q [sleep (+V)]] _S [[+Sg] PRES] _T] _V	T:AF
5.	[[[WH [+human]] INDEF [boy] _N] _{NP} [sleep [[+Sg] PRES] _T] _S (+V)]] _V	T:QDEL
Post Cycle:		
1.	[[[WH [+human]] INDEF [boy] _N] _{NP} [sleep] _S] _V] _S	T:NUAG
2.	[[[what [boy] _N] _{NP} [sleep] _S] _V] _S	T:WH 3

Post Cycle:

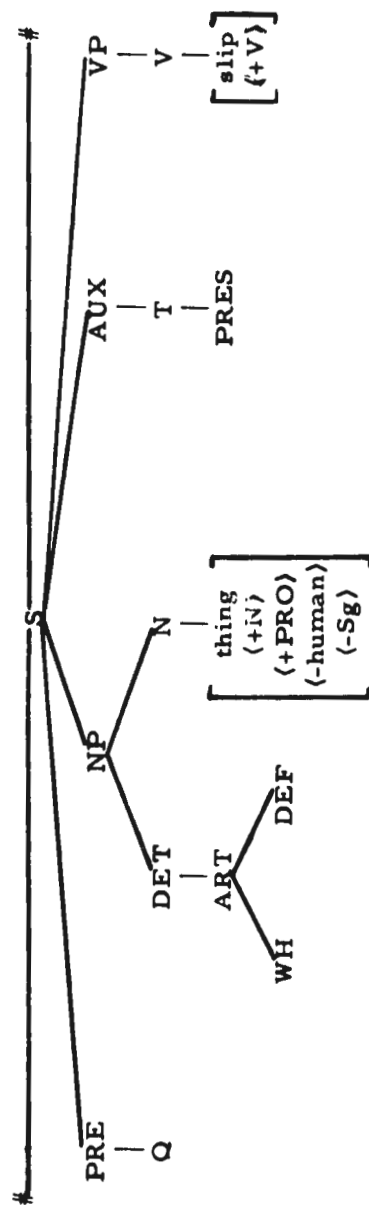
1. $[[w1] [+human]_{INDEX} [boy]_{NP} [sleep[s]_T]_V]_S$
2. $[[what [boy]_{NP} [sleep[s]_V]_S$

Sentence Type No. 25

Actual string: which things slip?

Derived string: which thing s slip

Underlying Structure:



Derivation:

First Cycle:

1. [Q [WH DEF [thing (-Sg) (-human)]_N] NP [-Sg] PRES [slip]_V] S T:AG
2. [Q [WH [-human] DEF [thing (-human)]_N] NP [-Sg] PRES [slip]_V] S T:WHAG
3. [[[WH [-human] DEF [thing]_N] NP Q [[[-Sg] PRES]_T] AUX [slip (+V)]_V] S T:QUES
4. [WH [-human] DEF [thing]_N] Q [slip (+V)]_V T:AF
5. [WH [-human] DEF [thing]_N] [slip [-Sg] PRES]_V T:QDEL

Post Cycle:

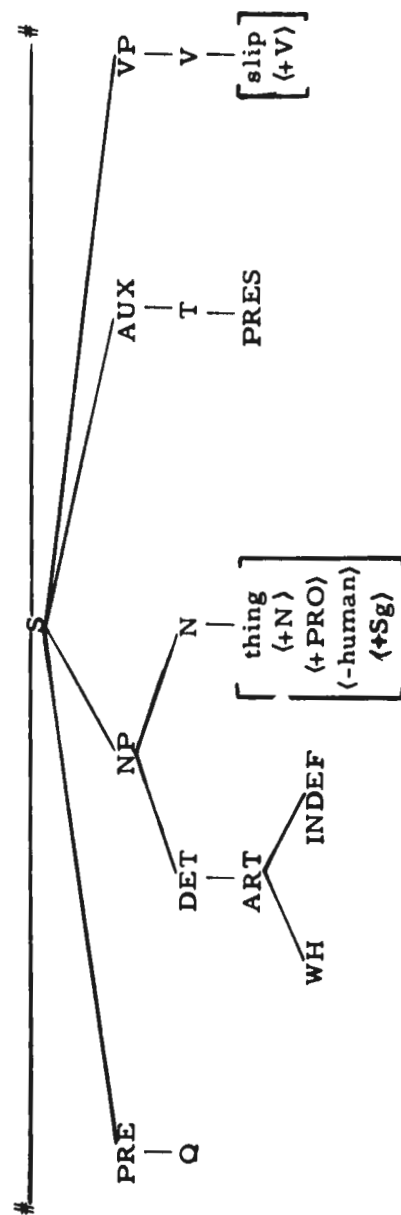
1. [WH [-human] DEF [thing (-Sg)]_N] [slip]_V] S T:PLUDEL
2. [[[WH [-human] DEF [thing + s] (-Sg)]_N] NP [slip]_V] S T:NUM
3. [which [thing s]_N] [slip]_V] S T:WH 2



Actual string: what slips?

Derived string: what slip s

Underlying structure:



Derivation:

First Cycle:

1. [Q [WH INDEF [thing (+Sg)]_{NP} [+Sg] PRES [slip]_V]_S T:AG
2. [Q [WH [-human] INDEF [thing]_N]_{NP} [+Sg] PRES [slip]_V]_S T:WHAG
3. [[WH [-human] INDEF [thing]_N]_{NP} Q [[+Sg] PRES]_T AUX [slip]_V]_S T:QUES
4. [WH [-human] INDEF [thing]_N Q [slip]_(+V) [[+Sg] PRES]_T]_V]_S T:AF
5. [WH [-human] INDEF [thing]_N [slip + [[+Sg] PRES]_T]_V]_S T:QDEL

Post Cycle:

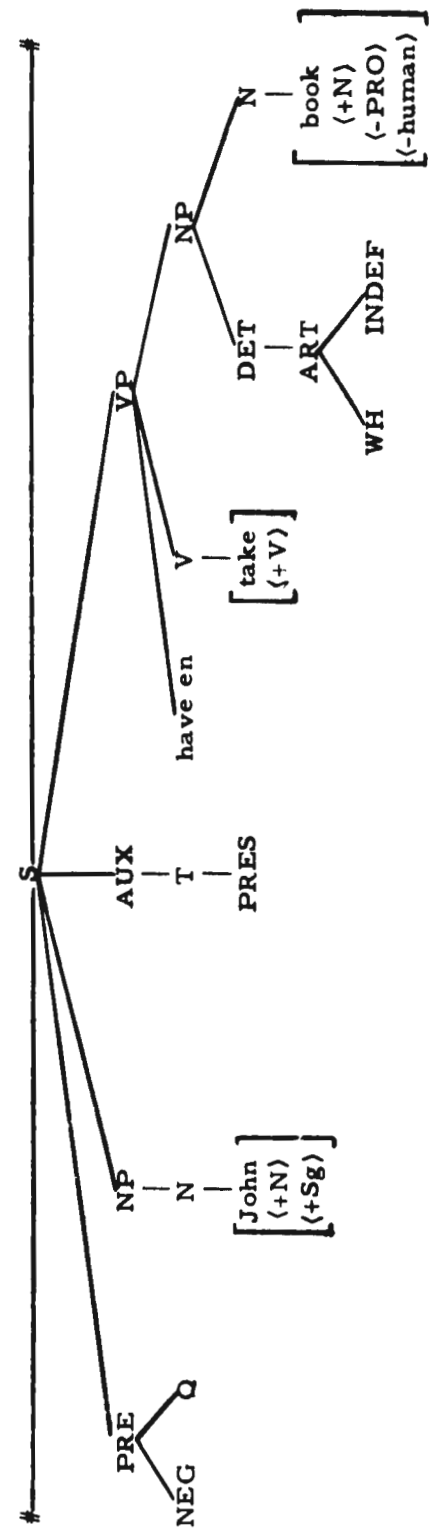
1. [[WH [-human] INDEF [thing (+PRO)]_{NP} [slip]_T]_V]_S T:NUAG
2. [[WH [-human] INDEF]_{NP} [slip s]_V]_S T:WHPD 2
3. [[what]_{NP} [slip [s] PRES]_V]_S T:WH 3

Sentence Type No. 27

Actual string: what book has John not taken?

Derived string: what book has John not taken en

Underlying structure:



Derivation:

First Cycle:

1. [NEG Q [[John]_{NP} [[[PRES]_T + have]_{AUX} en [take]_V WH INDEF [book]_S]_N]_N T:AUXFILL
2. [NEG Q [[John]_{NP} [[[[+Sg]_{PRES}]_T have]_{AUX} en [take]_V WH INDEF [book]_S]_N]_N T:AG
3. [NEG Q [[John]_{NP} [[[[+Sg]_{PRES}]_T have]_{AUX} en [take]_V WH [-human]_{INDEF} [book]_S]_N]_N T:WHAG
4. [Q [[John]_{NP} [[[[+Sg]_{PRES}]_T have]_{AUX} NEG en [take]_V [WH [-human]_{INDEF} [book]_{NP}]_S]_N]_N T:NEGPLACE
5. [[WH [-human]_{INDEF} [book]_{NP} Q [[John]_{NP} [[[[+Sg]_{PRES}]_T have]_{AUX} NEG en [take]_V]_S]_N]_N T:QUES

6. [[[WH [-human]]_{INDEF} [book]_N]_{NP} [[[+Sg]PRES]_T have]_{AUX} Q [[[John]_N]_{NP} NEG [_(+V) [_V take]]_S]]_V
T: YESNO
7. [[WH [-human]]_{INDEF} [book]_N [have + [[+Sg]PRES]_T]]_{AUX} Q [[John]_N NEG [_(+V) [_V take + en]]_S]]_V
T: AF
8. [[WH [-human]]_{INDEF} [book]_N [have [+Sg]PRES]_{AUX} [[John]_N NEG [take en]_V]]_S
T: QDEL

Post Cycle:

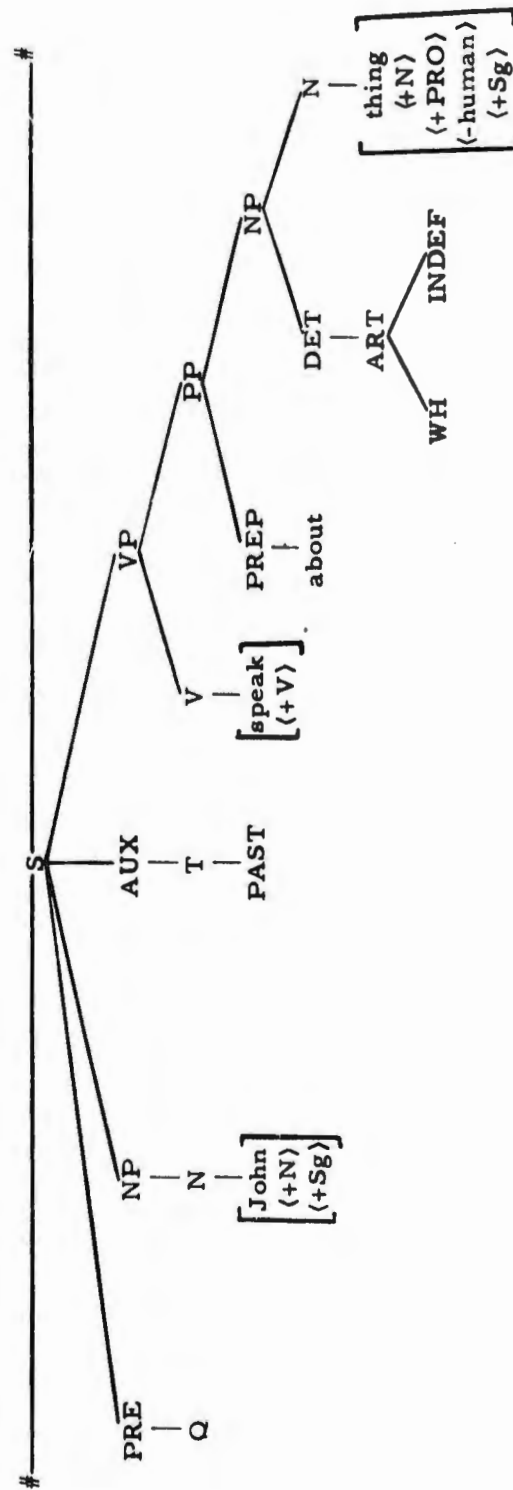
1. [[WH [-human]]_{INDEF} [book]_N [have [+Sg]PRES]_{AUX} [[John]_N not [take en]_V]]_S
T: NEGSPELL
2. [[[WH [-human]]_{INDEF} [book]_N]_{NP} [has]_{AUX} [[John]_N not [take en]_V]]_S
T: HAVE 1
3. [[[what [book]_N]_{NP} [has]_{AUX} [[John]_N not [take en]_V]]_S
T: WH 3

Sentence Type No. 28

Actual string: about what did John speak?

Derived string: about what did John speak

Underlying structure:



Derivation:

First Cycle:

1. $[Q [[John]_{NP} [+Sg]_{PAST} [speak]_V [about]_{PREP} WH_{INDEF} [thing]_{NP}]_S]_N$ T:AG
2. $[Q [[John]_{NP} [+Sg]_{PAST} [speak]_V [about]_{PREP} [WH [-human]_{INDEF} [thing]_{NP}]_N]_{NP}]_S$ T:WHAG
3. $[[[about]_{PREP} [WH [-human]_{INDEF} [thing]_{NP}]_{NP}]_{PP} Q [[John]_{NP}]_{NP}]_{T_{AUX} [speak]_V]_S$ T:QUES
4. $[[[about]_{PREP} [WH [-human]_{INDEF} [thing]_{NP}]_{NP}]_{T_{AUX} Q [John]_{NP} [speak]_V]_S]_S$ T:YESNO

5. [[about]_{PREP} [WH [-human]_{INDEF} [thing]_NNP [[[+Sg]_{PAST}]_TAUX [John]_N [speak]_V]_S T:QDEL

Post Cycle:

1. [[about]_{PREP} [WH [-human]_{INDEF} $\left[\begin{array}{c} \text{thing} \\ \langle +\text{PRO} \rangle \\ \langle +\text{Sg} \rangle \end{array} \right]_{\text{N}} \text{]}_{\text{NP}}$ [did]_{AUX} [John]_N [speak]_V]_S T:DO 1

2. [[about]_{PREP} [WH [-human]_{INDEF}]_{NP} [did]_{AUX} [John]_N [speak]_V]_S T:WHPD 2

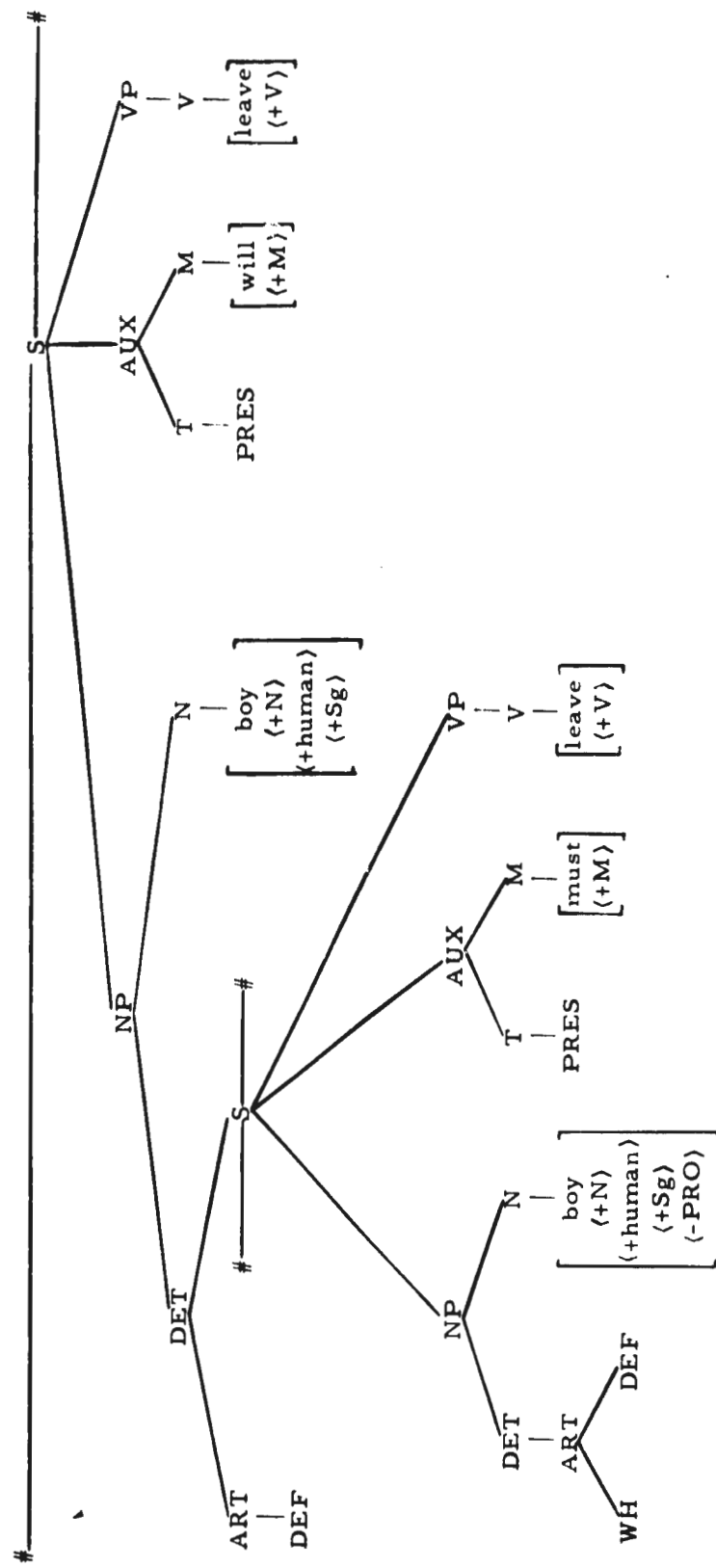
3. [[about]_{PREP} [what]_{NP} [did]_{AUX} [John]_N [speak]_V]_S T:WH 3

Sentence Type No. 29

Actual string: the boy who must leave will leave

Derived string: the boy who must leave will leave

Underlying structure:



Derivation:

First Cycle:

1. $X[[WH DEF \left[\begin{array}{c} \text{boy} \\ (+Sg) \\ (+human) \end{array} \right]]_{NP} [[+Sg] PRES]_T \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M$ T:AG
2. $X[[WH [+human]_{DEF} \left[\begin{array}{c} \text{boy} \\ (+human) \end{array} \right]]_N \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M$ T:WHAG
3. $X[[WH [+human]_{DEF} [\text{boy}]_{NP} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M$ T:AF

Second Cycle:

1. $[[[DEF]_{ART} [\text{boy}]_N [WH [+human]_{DEF} \left[\begin{array}{c} \text{boy} \\ (+Sg) \end{array} \right]]_N \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:RELPLACE
2. $[[[DEF]_{ART} \left[\begin{array}{c} \text{boy} \\ (+Sg) \end{array} \right]]_N [WH [+human]_{DEF} [\text{boy}]_N \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:AG
3. $[DEF [\text{boy}]_N [WH [+human]_{DEF} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:PROGDEL
4. $[DEF [\text{boy}]_N [WH [+human]_{DEF} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:AF

Post Cycle:

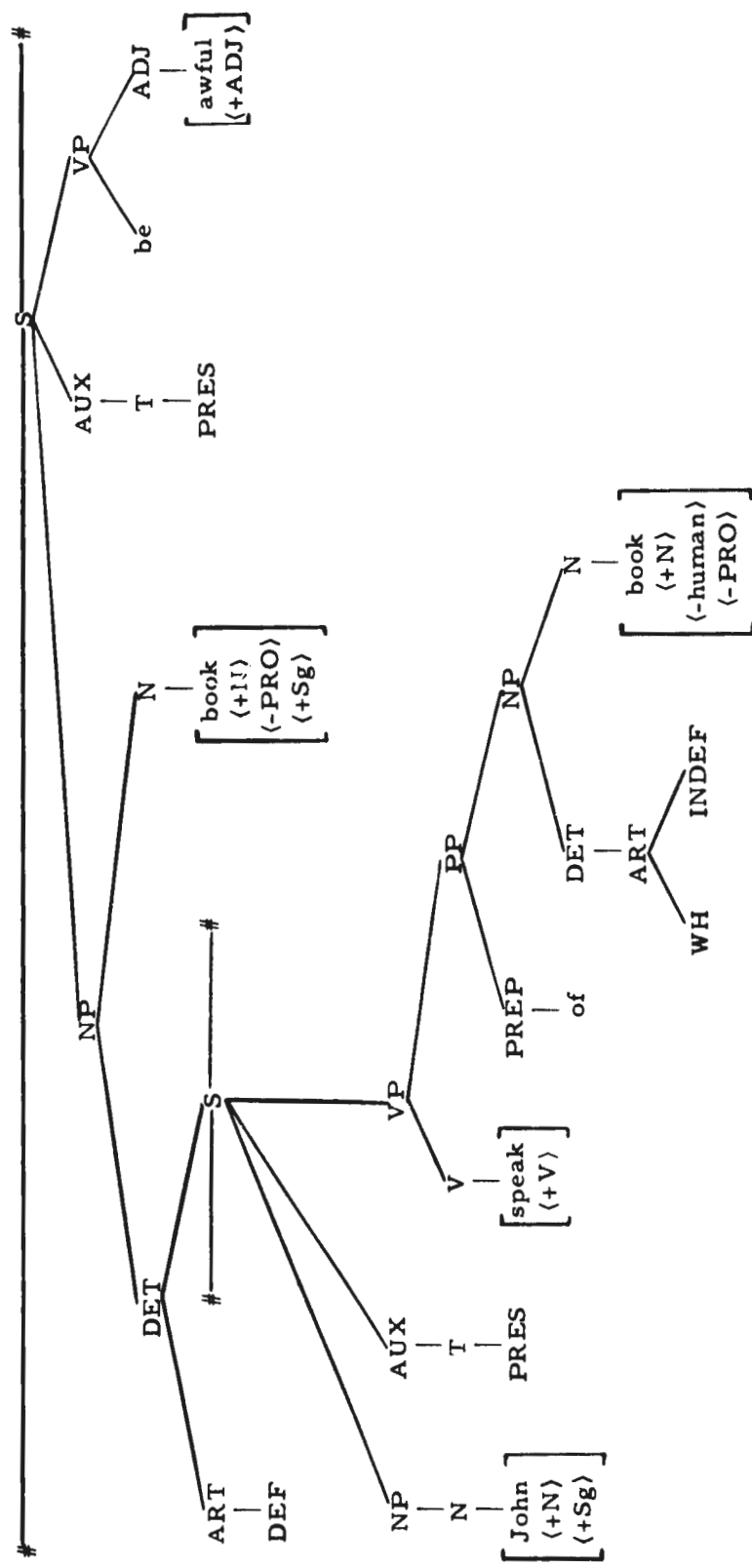
1. $[DEF [\text{boy}]_N [[WH [+human]_{DEF} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:MTDEL
2. $[DEF [\text{boy}]_N [[who]_{NP} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:WH 1
3. $[the [\text{boy}]_N [[who]_{NP} \left[\begin{array}{c} \text{must} \\ (+M) \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S Y \end{array} \right]]_M \left[\begin{array}{c} \text{will} \\ M \end{array} \right]]_M \left[\begin{array}{c} \text{leave} \\ V]_S \end{array} \right]]_M$ T:DEF

Actual string:	the book of which John speaks is awful
Derived string:	the book of which John speak s is awful

Actual string: the book of which John speaks is awful

Derived string: the book of which John speak s is awful

Underlying structure:



First Cycle:

XII

- Second Cycle:**

- Post Cycle:**

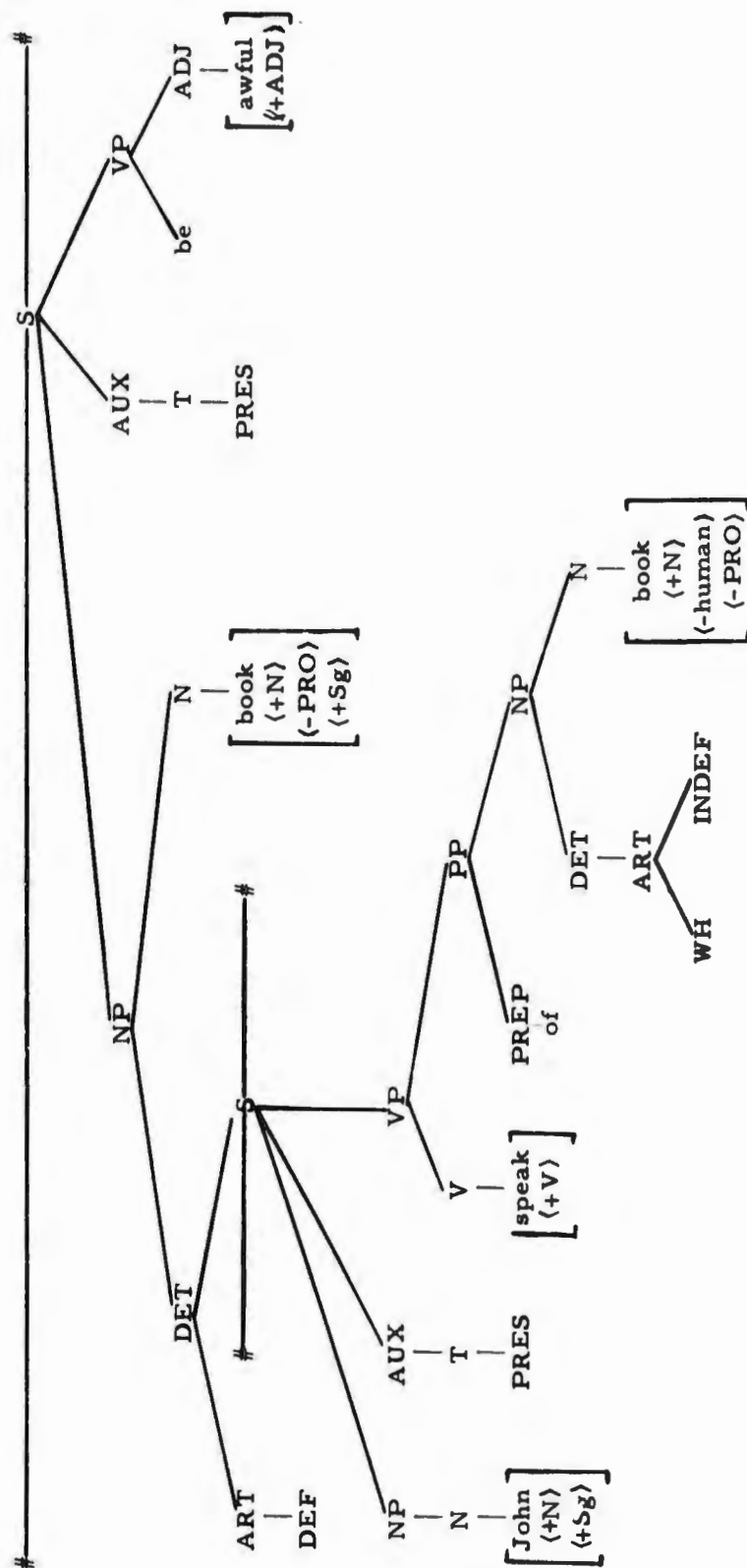
1. [DEF [book]_N [of wñi [-human]_{DEF} [John]_N [speak[s]_T]_VS [be [[+Sg]PRES]_TAUX [awful]_{ADJ}]_S T:NUAG
2. [DEF [book]_N [of [WH [-human]_{DEF}NP [John]_N [speak[s]_T]_VS [is]_{AUX} [awful]_{ADJ}]_S T:BE 1
3. [DEF [book]_N [of [which]_{NP} [John]_N [speak[s]_T]_VS [is]_{AUX} [awful]_{ADJ}]_S T:WH 2
4. [the [book]_N [of [which]_{NP} [John]_N [speak[s]_T]_VS [is]_{AUX} [awful]_{ADJ}]_S T:DEF

Sentence Type No. 31

Actual string: the book John speaks of is awful

Derived string: the book John speak s of is awful

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_N]_{NP} [+Sg] PRES [speak]_V of WH INDEF [book
(-human)]_N]_S Y T:AG
2. X [[[John]_N]_{NP} [+Sg] PRES]_T [speak
(+V)]_V of WH [-human] INDEF [book
(-human)]_N]_S Y T:WHAG
3. X [[[John]_N]_{NP} [speak
(+V)]_V of [WH [-human] INDEF [book]_N]_{NP}]_S Y T:AF

Second Cycle:

1. [[[DEF] ART [[WH[-human] INDEF [book]_N]_{NP} [John]_N [speak[+Sg] PRES]_V of] PREP]_S [book]_N]_{NP} PRES be[awful] ADJ]_S T:WHA
2. [[[DEF] ART [book]_N [WH[-human] INDEF [book]_N [John]_N [speak[+Sg] PRES]_V of]]_S]_{NP} [[PRES]_T]_T AUX be[awful] ADJ]_S T:RELPLACE
3. [[[DEF] [book
(+Sg)]_N [WH[-human] INDEF [book]_N [speak[+Sg] PRES]_V of]]_S]_{NP} [[PRES]_T +be] AUX [awful] ADJ]_S T:AUXFILL
4. [[[DEF] [book
(+Sg)]_N [WH[-human] INDEF [book]_N [John]_N [speak[+Sg] PRES]_V of]]_S]_{NP} [[[+Sg] PRES]_T be] AUX [awful] ADJ]_S T:AG
5. [DEF [book]_N [WH DEF [book
(-human)]_N] [John]_N [speak[+Sg] PRES]_V of]]_S [+Sg] PRES be [awful] ADJ]_S T:DEFT
6. [DEF [book]_N [[WH[-human] DEF [book
(-human)]_N] [John]_N [speak[+Sg] PRES]_V of]]_S [+Sg] PRES be[awful] ADJ]_S T:WHAG
7. [DEF [book]_N [[WH[-human] DEF]_{NP} [John]_N [speak[+Sg] PRES]_V of]]_S [[+Sg] PRES]_T be [awful] ADJ]_S T:PROGDEL
8. [DEF [book]_N [[WH[-human] DEF]_{NP} [John]_N [speak[+Sg] PRES]_V of]]_S [be + [[+Sg] PRES]_T]_T AUX [awful] ADJ]_S T:AF

Post Cycle:

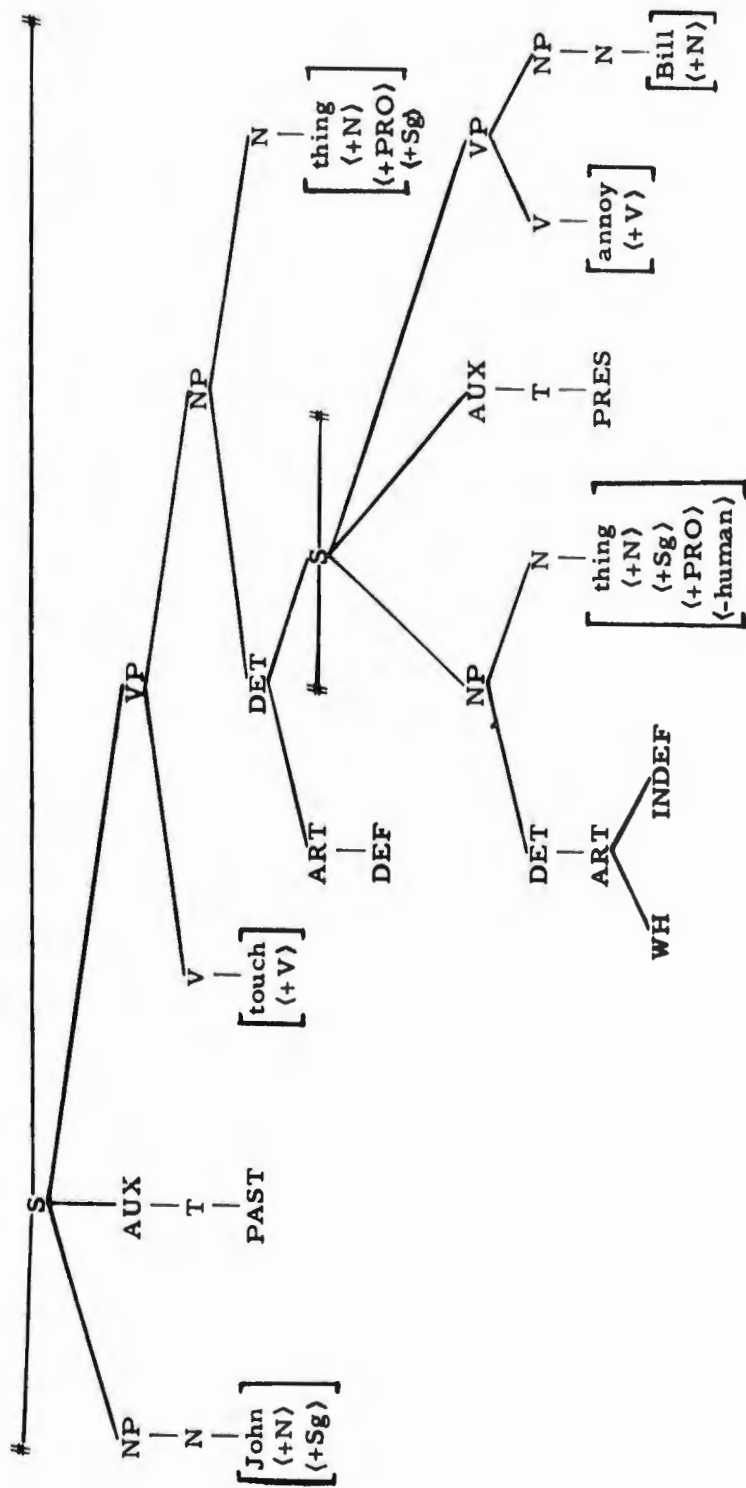
1. [DEF [book]_N [[WH[-human] DEF]_{NP} [John]_N [speak[s]]_T]_V of]]_S [be[+Sg] PRES] AUX [awful] ADJ]_S T:NUAG
2. [DEF [book]_N [[WH[-human] DEF]_{NP} [[John]_N]_{NP} [speak[s] PRES]_V of]]_S [is] AUX [awful] ADJ]_S T:BE 1
3. [DEF [book]_N [[[John]_N]_{NP} [speak[s] PRES]_V of]]_S [is] AUX [awful] ADJ]_S T:WHDEL
4. [the [book]_N [[John]_N [speak s]]_V of]]_S [is] AUX [awful] ADJ]_S T:DEF

Sentence Type No. 32

Actual string: John touched that which annoys Bill

Derived string: John touch ed that which annoy s Bill

Underlying structure:



Derivation:

First Cycle:

1. X [[WH INDEF $\left[\begin{array}{c} \text{thing} \\ (+Sg) \\ (-human) \end{array} \right]_N$]NP [+Sg]PRES [annoy]_V [Bill]_N'S Y T:AG
2. X [WH [-human]_NINDEF $\left[\begin{array}{c} \text{thing} \\ (-human) \end{array} \right]_N$] [+Sg]PRES_T [annoy]_V [Bill]_N'S Y T:WHAG
3. X [WH [-human]_NINDEF [thing]_N [annoy $\left[\begin{array}{c} (+V) \end{array} \right]_V$] [+Sg]PRES_T] [Bill]_N'S Y T:AF

Second Cycle:

1. [[[John]_N]NP PAST[touch]_V [[DEF]ART [thing]_N [WH[-human]INDEF [thing]_N [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:RELPLACE
2. [[[John]_N]NP [+Sg]PAST[touch]_V [DEF]ART [thing]_N [WH[-human]INDEF [thing]_N [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:AG
3. [[[John]_N]NP [+Sg]PAST[touch]_V [[DEF]ART [thing]_N [WH DEF $\left[\begin{array}{c} \text{thing} \\ (-human) \end{array} \right]_N$] [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:DEFT
4. [[[John]_N]NP [+Sg]PAST[touch]_V [DEF]ART [thing]_N [WH[-human]DEF $\left[\begin{array}{c} \text{thing} \\ (-human) \end{array} \right]_N$] [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:WHAG
5. [[[John]_N]NP [+Sg]PAST_T [touch]_V $\left[\begin{array}{c} (+V) \end{array} \right]_V$] [[DEF]ART [thing]_N [WH[-human]DEF [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:PROGDEL
6. [[[John]_N]NP touch $\left[\begin{array}{c} (+V) \end{array} \right]_V$] [+Sg]PAST_T [touch]_V [DEF [thing]_N [WH[-human]DEF [annoy [+Sg]PRES]_V [Bill]_N'S]NP'S T:AF

Post Cycle:

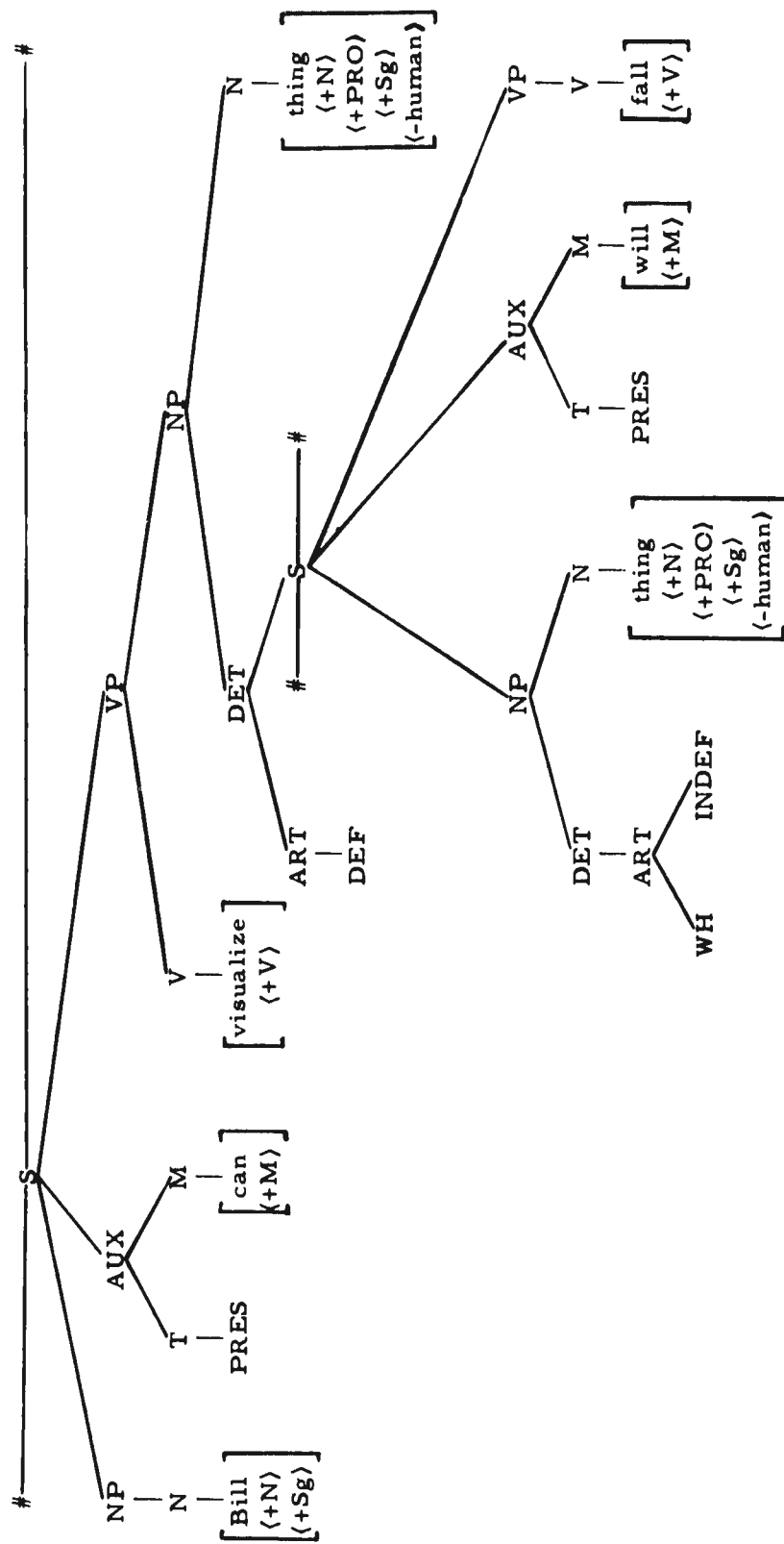
1. [[[John]_N]touch [ed]_T]V DEF [thing]_N [WH[-human]DEF $\left[\begin{array}{c} \text{annoy} [+Sg] \text{PRES} \\ (+V) \end{array} \right]_V$] [Bill]_N'S]NP'S T:PAST
2. [[[John]_N]touch ed]_V DEF $\left[\begin{array}{c} \text{thing} \\ (+PRO) \\ (+Sg) \end{array} \right]_N$] [WH[-human]DEF [annoy [s]_T]V [Bill]_N'S]NP'S T:NUAG
3. [[[John]_N]touch ed]_V DEF [WH[-human]DEF [annoy s]_V [Bill]_N'S]NP'S T:WHPD 1
4. [[[John]_N]touch ed]_V that [[[WH[-human]DEF]_{NP} [annoy s]_V [Bill]_N'S]NP'S T:DEFTHAT
5. [[[John]_N]touch ed]_V that [[[which]_{NP} [annoy s]_V [Bill]_N'S]NP'S T:WH 2

Sentence Type No. 33

Actual string: Bill can visualize what will fall

Derived string: Bill can visualize what will fall

Underlying structure:



Derivation:

First Cycle:

1. X [[WH INDEF $\left[\begin{array}{c} \text{thing} \\ (+Sg) \\ (-human) \end{array} \right]_N$] NP [+Sg] PRES will [fall]_V S] Y
T:AG
2. X [WH [-human] INDEF $\left[\begin{array}{c} \text{thing} \\ (-human) \end{array} \right]_N$] [[+Sg] PRES T $\left[\begin{array}{c} \text{will} \\ (+M) \end{array} \right]_M$] [fall]_V S] Y
T:WHAG
3. X [WH [-human] INDEF [thing]_N [will + [[+Sg] PRES] T] $\left[\begin{array}{c} \text{fall} \\ (+M) \end{array} \right]_M$] S] Y
T:AF

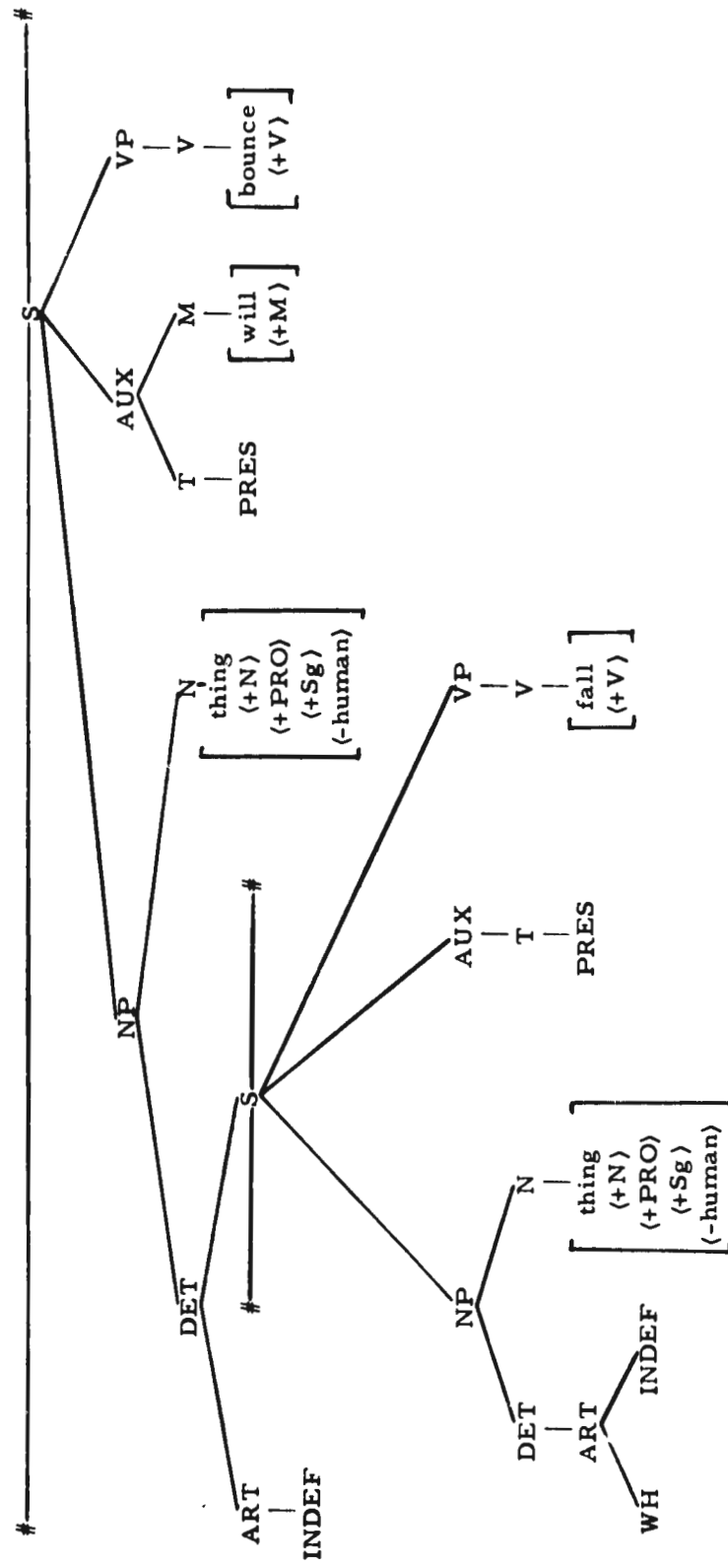
Second Cycle:

1. [[[Bill] NP PRES [can]_M [visualize]_V [[DEF] ART [thing]_N [WH [-human] INDEF [thing]_N [will [+Sg] PRES]_M [fall]_V S] NP] S]
T:RELPLACE
2. [[[Bill] NP [+Sg] PRES [can]_M [visualize]_V [DEF] ART $\left[\begin{array}{c} \text{thing} \\ (+PRO) \end{array} \right]_N$] [WH [-human] INDEF [thing]_N [will [+Sg] PRES]_M [fall]_V S] S]
T:AG
3. [[[Bill] NP [[+Sg] PRES] T $\left[\begin{array}{c} \text{can} \\ (+M) \end{array} \right]_M$] [visualize]_V [WH [-human] INDEF [thing]_N [will [+Sg] PRES]_M [fall]_V S] S]
T:REGDEL
4. [[[Bill] NP [can + [[+Sg] PRES] T] $\left[\begin{array}{c} \text{visualize} \\ (+M) \end{array} \right]_M$] [WH [-human] INDEF [thing]_N [will [+Sg] PRES]_M [fall]_V S] S]
T:AF

Post Cycle:

1. [[[Bill] NP [can]_M [visualize]_V [WH [-human] INDEF $\left[\begin{array}{c} \text{thing} \\ (+PRO) \end{array} \right]_N$] [will]_M [fall]_V S] S]
T:MTDEL
2. [[[Bill] NP [can]_M [visualize]_V [[WH [-human] INDEF] NP [will]_M [fall]_V S] S]
T:WHPD 2
3. [[[Bill] NP [can]_M [visualize]_V [will]_M [fall]_V S] S]
T:WH 3

Underlying structure:



Derivation:

First Cycle:

1. X [[WH INDEF $\left[\begin{array}{c} \text{thing} \\ \langle +Sg \rangle \\ \langle -human \rangle \end{array} \right]]_{NP} [+Sg]_{PRES} [fall]_{V'}]_S^Y$ T:AG
2. X [WH [-human] INDEF $\left[\begin{array}{c} \text{thing} \\ \langle -human \rangle \end{array} \right]]_N [[+Sg]_{PRES}]_T \left[\begin{array}{c} \text{fall} \\ \langle +V \rangle \end{array} \right]]_S^Y$ T:WHAG
3. X [WH [-human] INDEF $\left[\begin{array}{c} \text{fall} \\ \langle +V \rangle \end{array} \right]]_N [[+Sg]_{PRES}]_T]_S^Y$ T:AF

Second Cycle:

1. [[[INDEF] ART $\left[\begin{array}{c} \text{thing} \\ \langle +Sg \rangle \end{array} \right]]_N$ [WH [-human] INDEF $\left[\begin{array}{c} \text{thing} \\ \langle -human \rangle \end{array} \right]]_N$ [fall [+Sg] PRES] V] NP PRES [will] M [bounce] V] S T:RELPLACE
2. [[[INDEF] ART $\left[\begin{array}{c} \text{thing} \\ \langle +Sg \rangle \\ \langle -human \rangle \end{array} \right]]_N$ [WH [-human] INDEF $\left[\begin{array}{c} \text{thing} \\ \langle -human \rangle \end{array} \right]]_N$ [fall [+Sg] PRES] V] NP [+Sg] PRES [will] M [bounce] V] S T:AG
3. [[[INDEF] ART $\left[\begin{array}{c} \text{thing} \\ \langle -human \rangle \end{array} \right]]_N$ [WH [-human] INDEF ever [thing] N [fall [+Sg] PRES] V] S [+Sg] PRES [will] M [bounce] V] S T:EVER
4. [[[WH [-human] INDEF ever [thing] N [fall [+Sg] PRES] V] S] NP [[+Sg] PRES] T $\left[\begin{array}{c} \text{will} \\ \langle +M \rangle \end{array} \right]]_M$ [bounce] V] S T:REGDEL
5. [[[WH [-human] INDEF ever [thing] N [fall [+Sg] PRES] V] S] $\left[\begin{array}{c} \text{will} \\ \langle +M \rangle \end{array} \right]]_M$ [+Sg] PRES [will] M [bounce] V] S T:AF

Post Cycle:

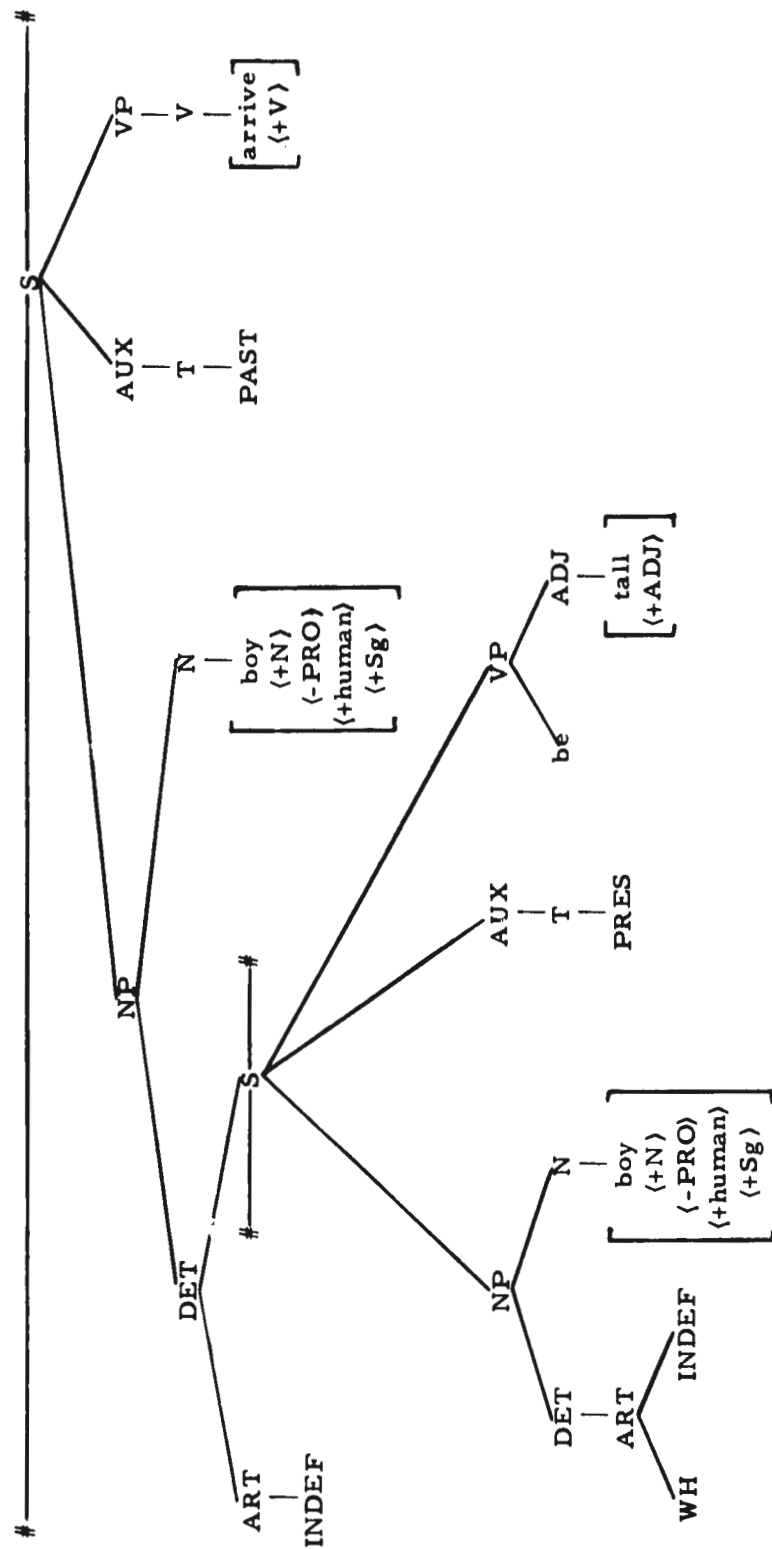
1. [[[WH [-human] INDEF ever [thing] N $\left[\begin{array}{c} \text{fall} \\ \langle +V \rangle \end{array} \right]]_N$ [+Sg] PRES] V] S [will] M [bounce] V] S T:MTDEL
2. [[[WH [-human] INDEF ever $\left[\begin{array}{c} \text{thing} \\ \langle +PRO \rangle \\ \langle +Sg \rangle \end{array} \right]]_N$ [fall [s] T] V] S [will] M [bounce] V] S T:NUAG
3. [[[WH [-human] INDEF ever] NP [fall [s] T] V] S [will] M [bounce] V] S T:WHPD 2
4. [[[what ever] NP [fall [s] T] V] S [will] M [bounce] V] S T:WH 3

Sentence Type No. 35

Actual string: a tall boy arrived

Derived string: a tall boy arrive ed

Underlying structure:



Derivation:

First Cycle:

1. X [WH INDEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +Sg \rangle \end{smallmatrix} \right]_N$ [[PRES]_T + be]_{AUX} [tall]_{ADJ}_S Y] T:AUXFILL
2. X [WH INDEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +Sg \rangle \\ \langle +human \rangle \end{smallmatrix} \right]_N$ [[[+Sg]PRES]_T be]_{AUX} [tall]_{ADJ}_S Y] T:AG
3. X [WH [+human]_{INDEF} $\left[\begin{smallmatrix} \text{boy} \\ \langle +human \rangle \end{smallmatrix} \right]_N$ [[[+Sg]PRES]_T be]_{AUX} [tall]_{ADJ}_S Y] T:WHAG
4. X [WH [+human]_{INDEF} [boy]_N [be + [[+Sg]PRES]_T]_{AUX} [tall]_{ADJ}_S Y] T:AF

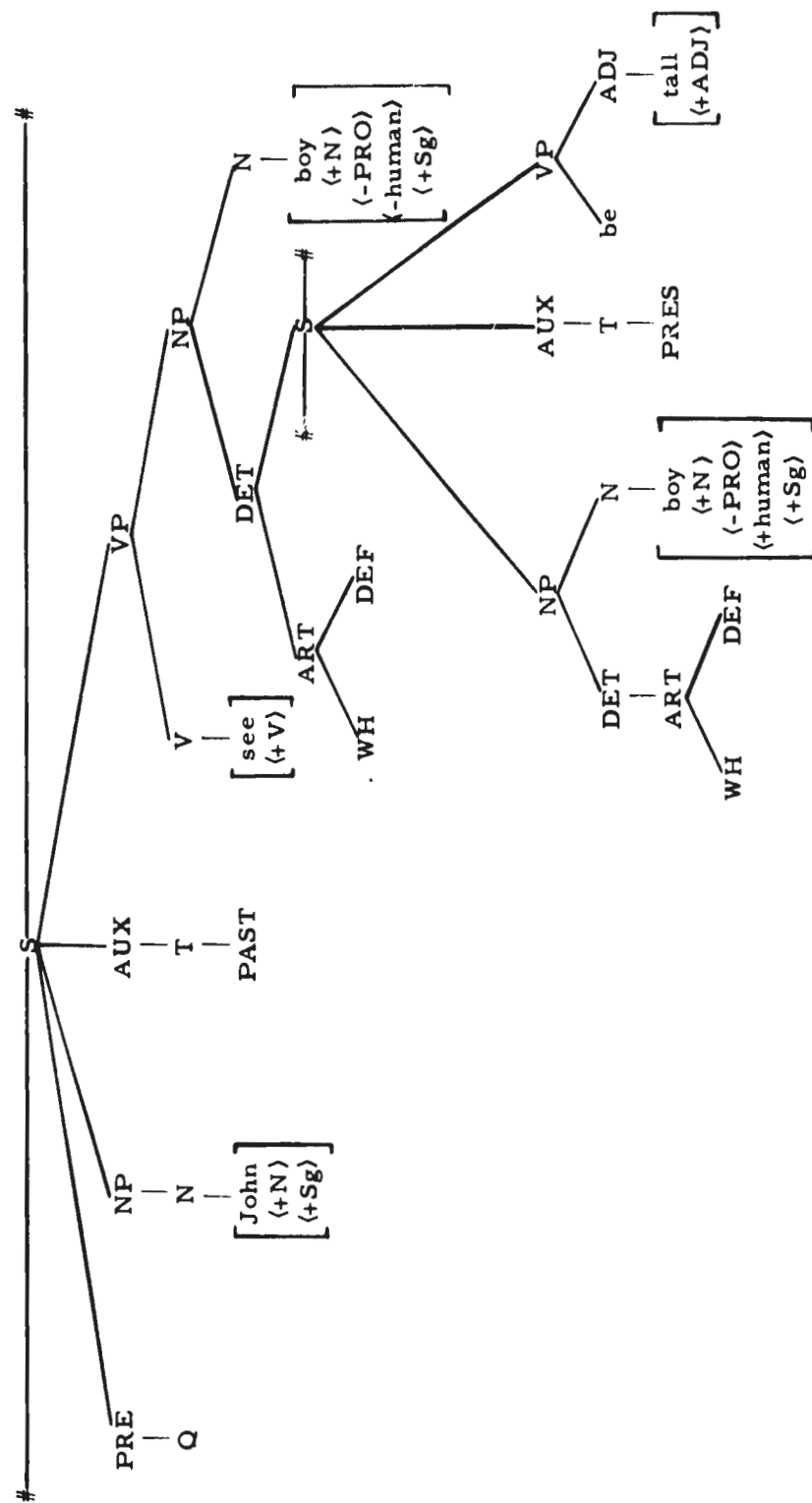
Second Cycle:

1. [[[INDEF]_{ART} $\left[\begin{smallmatrix} \text{boy} \\ \langle +Sg \rangle \end{smallmatrix} \right]_N$ [WH[+human]_{INDEF} [boy]_N [be[+Sg]PRES]_{AUX} [tall]_{ADJ}_S]_{NP}PAST[arrive]_V]_S T:RELPLACE
2. [[INDEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +Sg \rangle \end{smallmatrix} \right]_N$ [WH[+human]_{INDEF} [boy]_N [be[+Sg]PRES]_{AUX} [tall]_{ADJ}_S]_{NP} [+Sg]PAST[arrive]_V]_S T:AG
3. [INDEF [boy]_N [WH DEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +human \rangle \end{smallmatrix} \right]_N$ [be[+Sg]PRES]_{AUX} [tall]_{ADJ}_S [+Sg]PAST[arrive]_V]_S T:DEFI
4. [INDEF [boy]_N [[WH[+human]_{DEF} $\left[\begin{smallmatrix} \text{boy} \\ \langle +human \rangle \end{smallmatrix} \right]_N$]_{NP} [be[+Sg]PRES]_{AUX} [tall]_{ADJ}_S [+Sg]PAST[arrive]_V]_S T:WHAG
5. [INDEF [boy]_N [[WH[+human]_{DEF}]_{NP} [be[+Sg]PRES]_{AUX} [tall]_{ADJ}_S [+Sg]PAST[arrive]_V]_S T:PROGDEL
6. [[INDEF [boy]_N [[tall]_{ADJ}_S]_{NP} [+Sg]PAST[arrive]_V]_S T:RELDEL
7. [[[INDEF [tall]_{ADJ} [boy]_N]_{NP} [[+Sg]PAST]_T $\left[\begin{smallmatrix} \text{arrive} \\ \langle +V \rangle \end{smallmatrix} \right]_V$]_S T:ADJPLACE
8. [INDEF [tall]_{ADJ} [boy]_N [arrive [[+Sg]PAST]_T]_V]_S T:AF

Post Cycle:

1. [INDEF [tall]_{ADJ} [boy]_N [arrive [ed]_T]_V]_S T:PAST
2. [a [tall]_{ADJ} [boy]_N [arrive ed]_V]_S T:INDEF

Underlying structure:



Derivation:

First Cycle:

1. X [WH DEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +\text{Sg} \rangle \end{smallmatrix} \right]_N$ [[PRES]_T + be]_{AUX} [tall]_{ADJ}]_S Y
T:AUXFILL
2. X [WH DEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +\text{Sg} \rangle \end{smallmatrix} \right]_N$ [[[[+Sg]PRES]_T be]_{AUX} [tall]_{ADJ}]_S Y
T:AG
3. X [WH [+human]DEF $\left[\begin{smallmatrix} \text{boy} \\ \langle +\text{human} \rangle \end{smallmatrix} \right]_N$ [[[[+Sg]PRES]_T be]_{AUX} [tall]_{ADJ}]_S Y
T:WHAG
4. X [WH [+human]DEF [boy]_N [be + [[+Sg]PRES]_T]_{AUX} [tall]_{ADJ}]_S Y
T:AF

Second Cycle:

1. [Q [[John]]_{NP} PAST[see]_V [[WH DEF]_{ART} [boy]_N [WH[+human]DEF [boy]_N [be [+Sg]PRES]_{AUX} [tall]_{ADJ}]_S]_{NP}]_S
T:RELPLACE
2. [Q [[John]]_{NP} [+Sg]PAST[see]_V [[WH DEF]_{ART} [boy]_N [WH[+human]DEF [boy]_N [be [+Sg]PRES]_{AUX} [tall]_{ADJ}]_S]_{NP}]_S
T:AG
3. [Q [[John]]_{NP} [+Sg]PAST[see]_V [[WH DEF]_{ART} [boy]_N [[WH[+human]DEF]_{NP} [be [+Sg]PRES]_{AUX} [tall]_{ADJ}]_S]_{NP}]_S
T:PROGDEL
4. [Q [[John]]_{NP} [+Sg]PAST[see]_V [[WH DEF]_{ART} [boy]_N [[tall]_{ADJ}]_S]_{NP}]_S
T:RELDEL
5. [Q [[John]_N]_{NP} [+Sg]PAST [see]_V [[WH DEF]_{ART} [tall]_{ADJ} [boy]_N]_{NP}]_S
T:ADJPLACE
6. [[WH[+human]DEF [tall]_{ADJ} [boy]_N]_{NP} Q [[John]_N]_{NP} [[[[+Sg]PAST]_T]_{AUX} [see]_V]_S
T:QUES
7. [[WH[+human]DEF [tall]_{ADJ} [boy]_N]_{NP} [[[[+Sg]PAST]_T]_{AUX} Q [[John]_N]_{NP} [see]_V]_S
T:YESNO
8. [[WH[+human]DEF [tall]_{ADJ} [boy]_N]_{NP} [[[[+Sg]PAST]_T]_{AUX} [John]_N [see]_V]_S
T:QDEL

Post Cycle:

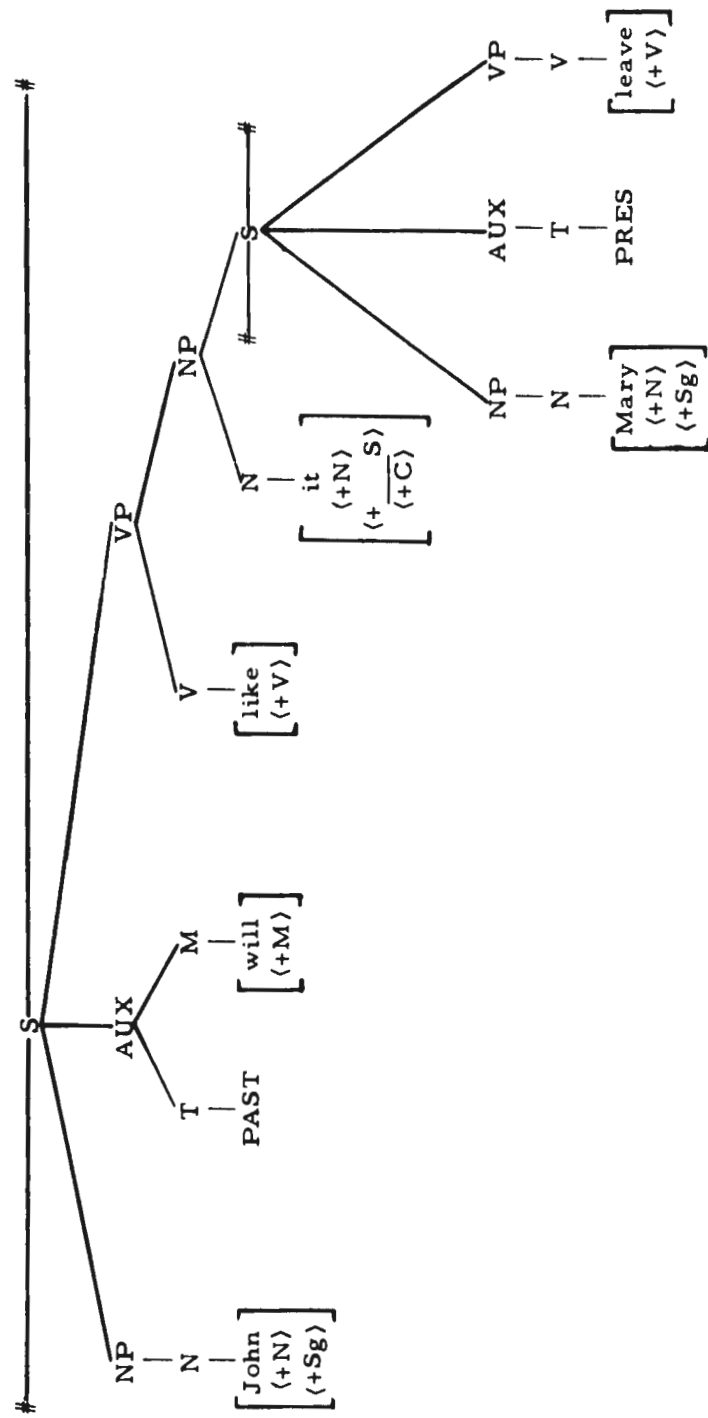
1. [[WH[+human]DEF [tall]_{ADJ} [boy]_N]_{NP} [did]_{AUX} [John]_N [see]_V]_S
T:DO 1
2. [[which [tall]_{ADJ} [boy]_N]_{NP} [did]_{AUX} [John]_N [see]_V]_S
T:WH 2

Sentence Type No. 37

Actual string: John would like for Mary to leave

Derived string: John will ed like for Mary to leave

Underlying structure:



Derivation:

First Cycle:

1. X [[[Mary]_N]_{NP} [+Sg] PRES]_T [leave]_S]_V T:AG
2. X [[Mary]_N]_{NP} [+Sg] PRES]_T]_S]_V T:AF

Second Cycle:

1. [[[John]_N]_{NP} PAST [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} [leave [+Sg] PRES]_V]_S]_S T:CP 1
2. [[[John]_N]_{NP} [+Sg] PAST [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} [leave [+Sg] PRES]_V]_S]_S T:AG
3. [[[John]_N]_{NP} [+Sg] PAST [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} +C [leave [+Sg] PRES]_T]_S]_S T:CDUP
4. [[[John]_N]_{NP} [+Sg] PAST]_T [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S T:TS
5. [[[John]_N]_{NP} [+Sg] PAST]_T [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} +C [leave]_V]_S]_S T:AF
6. [[[John]_N]_{NP} [+Sg] PAST]_T [will]_M [like]_V [it]_S]_V]_N [+C [[Mary]_N]_{NP} +C [leave]_V]_S]_S T:PD

Post Cycle:

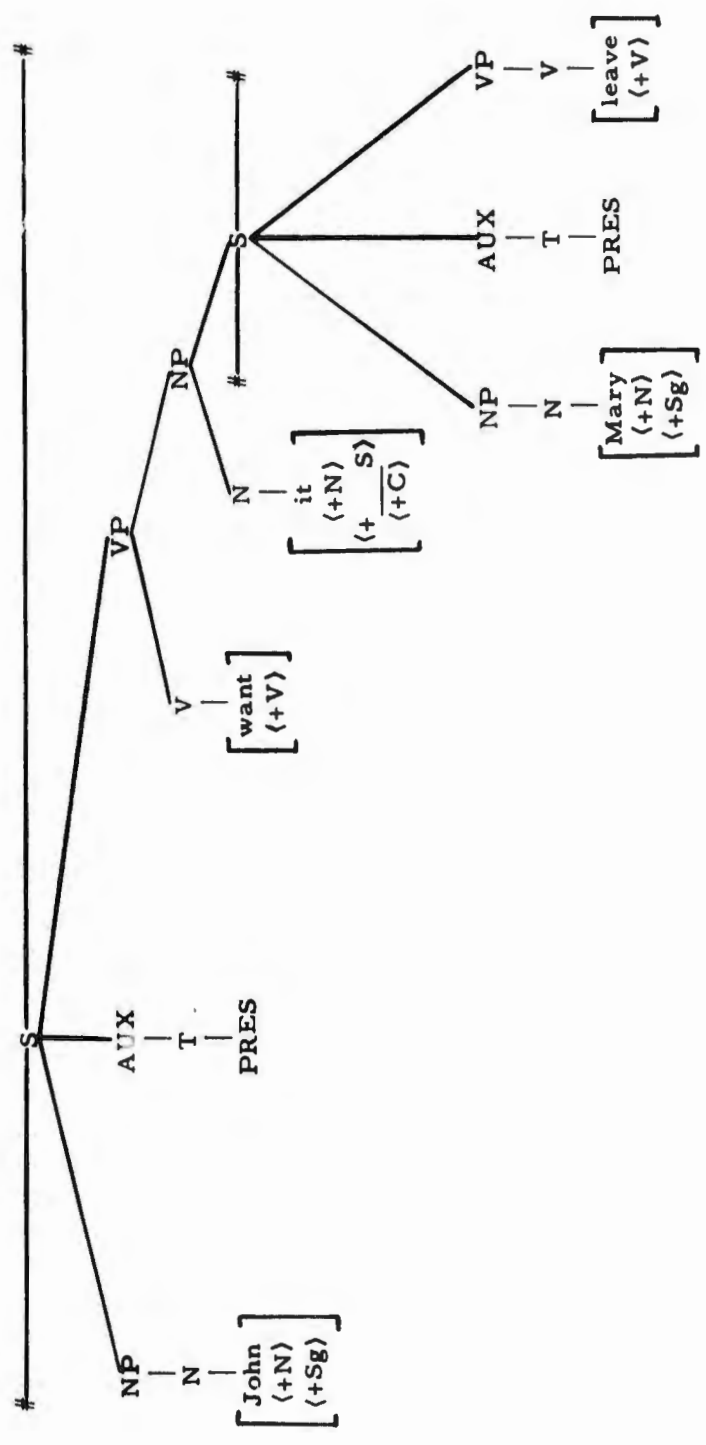
1. [[[John]_N]_{NP} [will ed]_T]_M [like]_V [+C [[Mary]_N]_{NP} +C [leave]_V]_S]_S T:PAST
2. [[[John]_N]_{NP} [will ed]_T]_M [like]_V [for [[Mary]_N]_{NP} +C [leave]_V]_S]_S T:C 1
3. [[[John]_N]_{NP} [will ed]_T]_M [like]_V [for [[Mary]_N]_{NP} to [leave]_V]_S]_S T:C 2

Sentence Type No. 38

Actual string: John wants Mary to leave

Derived string: John want s Mary to leave

Underlying structure:



Derivation:

First Cycle:

1. X [[[Mary]_N]_{NP} [[+Sg]PRES]_T [leave]_S]_S ^Y
 [(+V)]_V
2. X [[Mary]_N]_{NP} [leave]_S ^Y
 [(+V)]_V

T:AG

T:AF

Second Cycle:

1. [[[John]_N]_{NP} PRES [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} [leave]_S]_S
2. [[[John]_N]_{NP} [+Sg]PRES [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} [leave]_S]_S
3. [[[John]_N]_{NP} [+Sg]PRES [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S
4. [[[John]_N]_{NP} [+Sg]PRES [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S
5. [[[John]_N]_{NP} [[+Sg]PRES]_T [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S
6. [[[John]_N]_{NP} [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S
7. [[[John]_N]_{NP} [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S

T:CP 1

T:AG

T:CDUP

T:TS

T:CD

T:AF

T:PD

Post Cycle:

1. [[[John]_N]_{NP} [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S
2. [[[John]_N]_{NP} [want]_S]_S ^Y
 [(+V)]_V [it]_S ^Y
 [(+C)]_N [+C [[Mary]_N]_{NP} +C [leave]_S]_S

T:NUAG

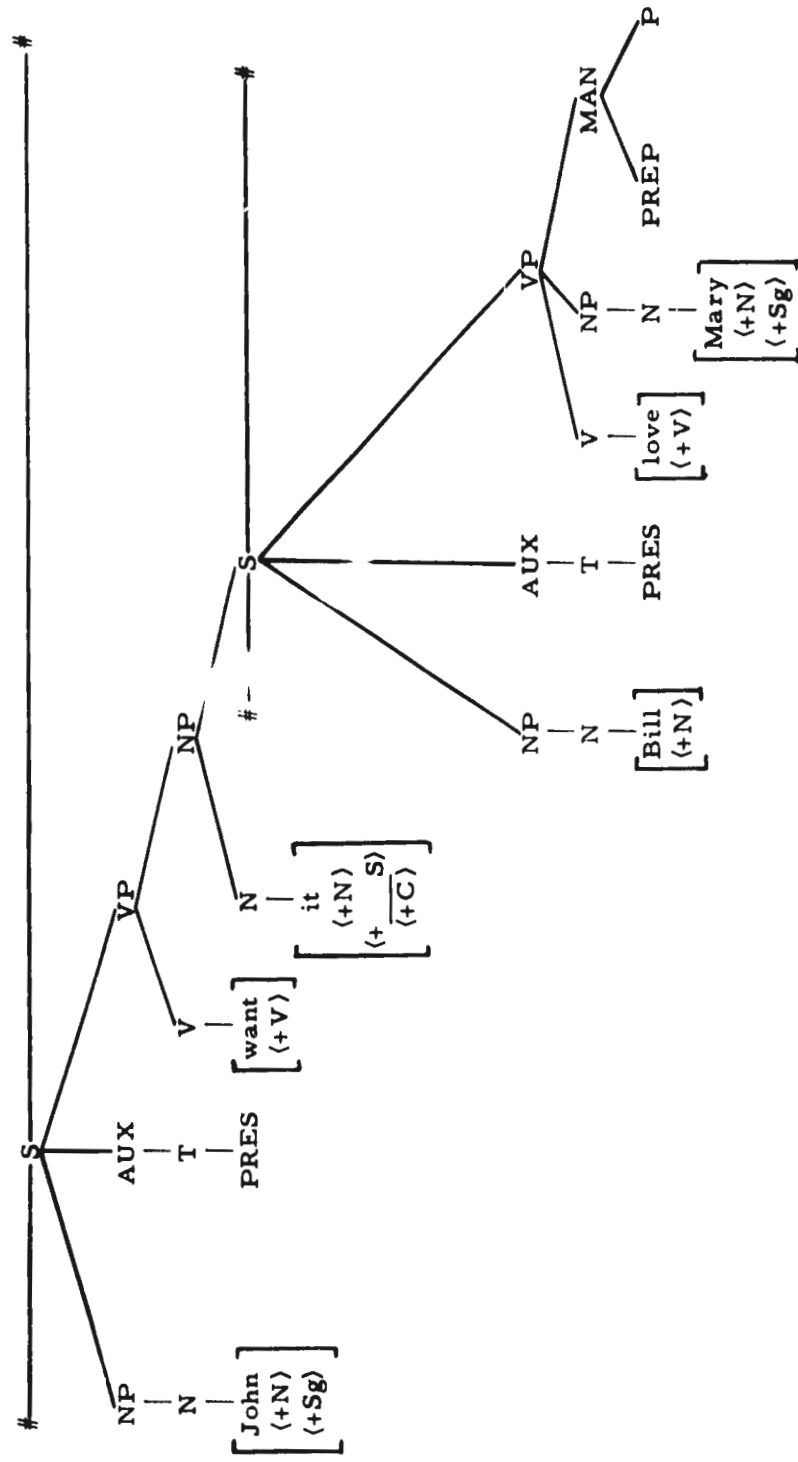
T:C 2

Sentence Type No. 39

Actual string: John wants Mary to be loved by Bill

Derived string: John want s Mary to be love en by Bill

Underlying structure:



Derivation:

First Cycle:

1. X [[[Mary]_NNP] PRES]_T AUX be + en + love [PREP[[Bill]_NNP]MAN]_S Y T:PASSIVE
2. X [[Mary]_NNP] PRES]_T + be] AUX en love PREP [Bill]_NS Y T:AUXFILL
3. X [[Mary]_NNP] PRES]_T be] AUX en [love]_(+V) PREP [Bill]_NS Y T:AG
4. X [[Mary]_N be + [[+Sg]PRES]_T AUX [love + en]_(+V) PREP [Bill]_NS Y T:AF

Second Cycle:

1. [[[John]_NNP] PRES [want]_N [it]_(+C)]_N [+C [[Mary]_NNP] be [+Sg]PRES]_T AUX [love en]_V PREP [Bill]_NS]_S T:CP 1
2. [[John]_NNP] [+Sg]PRES [want]_V [it]_N [+C [[Mary]_NNP] be [+Sg]PRES]_T AUX [love en]_V PREP [Bill]_NS]_S T:AG
3. [[John]_NNP] [+Sg]PRES [want]_V [it]_N [+C [[Mary]_NNP] +C [be [[+Sg]PRES]_T AUX [love en]_V PREP [Bill]_NS]_S T:CDUP
4. [[John]_NNP] [+Sg]PRES [want]_V [it]_(+S)]_N [+C [[Mary]_NNP] +C [be] AUX [love en]_V PREP [Bill]_NS]_S T:TS
5. [[John]_NNP] [+Sg]PRES [want]_(+V)]_V [it]_(+S)]_N [[[Mary]_NNP] +C [be] AUX [love en]_V PREP [Bill]_NS]_S T:CD
6. [[John]_NNP] want [+Sg]PRES]_T [it]_(+S)]_V [[[Mary]_NNP] +C [be] AUX [love en]_V PREP [Bill]_NS]_S T:AF
7. [[John]_NNP] want [+Sg]PRES]_V [[Mary]_NNP] +C [be] AUX [love en]_V PREP [Bill]_NS]_S T:PD

Post Cycle:

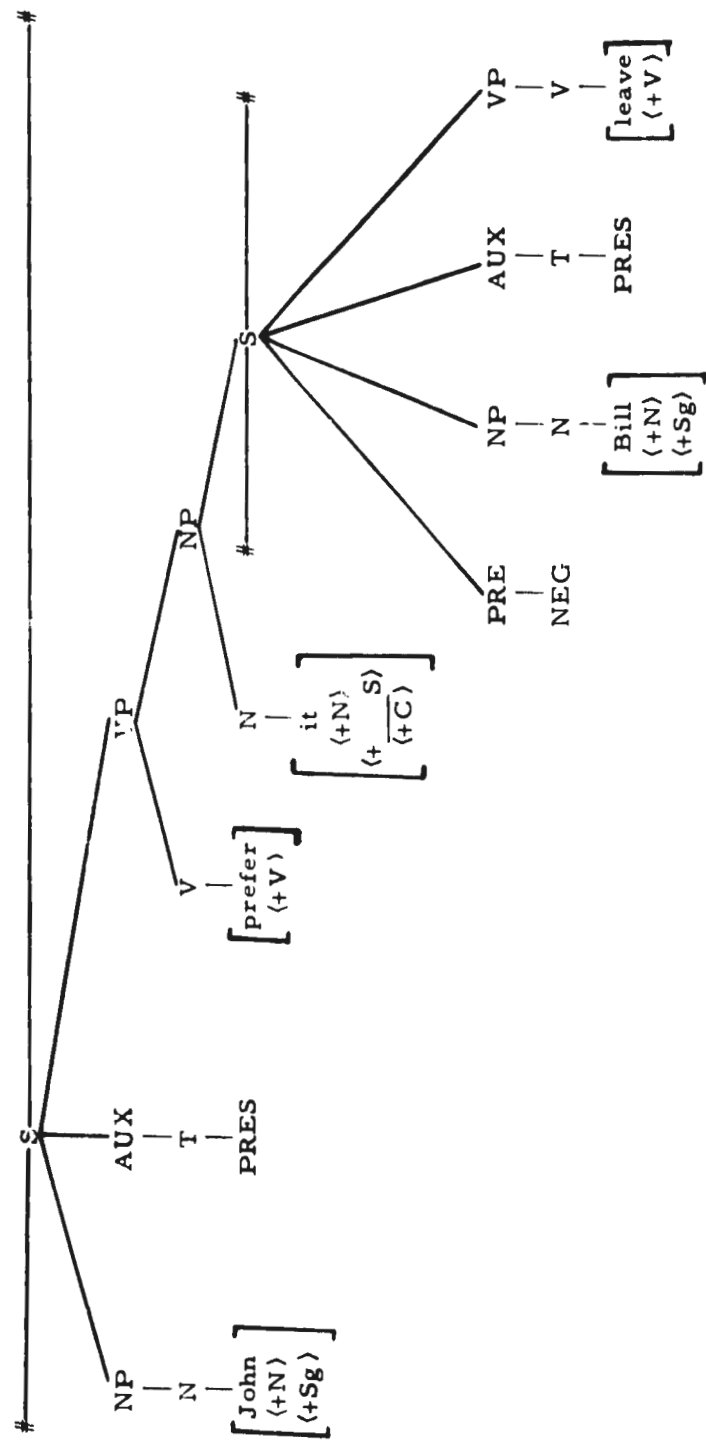
1. [[John]_NNP] want[s]_(+V)]_V [[[Mary]_NNP] +C [be] AUX [love en]_V PREP [Bill]_NS]_S T:NUAG
2. [[John]_NNP] want[s]_(+V)]_V [[Mary]_NNP] to [be] AUX [love en]_V [PREP [[Bill]_NNP]MAN]_SS T:C 2
3. [[John]_NNP] want[s]_(+V)]_V [[Mary]_NNP] to [be] AUX [love en]_V [by [[Bill]_NNP]MAN]_SS T:BY

Sentence Type No.	40
Actual string:	John prefers for Bill not to leave
Derived string:	John prefer s for Bill not to leave

Actual string: John prefers for Bill not to leave

Derived string: John prefers for Bill not to leave

Underlying structure:



Derivation:

First Cycle:

1. X [NEG [[Bill]_N]_{NP} [[+Sg]PRES]_TAUX [leave]_S]_V]_Y T:AG
2. X [[[Bill]_N]_{NP} [[+Sg]PRES]_TAUX NEG [leave]_V]_S]_Y T:NEGPLACE

Second Cycle:

1. [[[John]_N]_{NP} PRES [prefer]_V [it]_N]_Y [+C [[Bill]_N]_{NP} [+Sg]PRES NEG [leave]_V]_S]_Y T:CP 1
2. [[[John]_N]_{NP} [+Sg]PRES [prefer]_V [it]_N]_Y [+C [[Bill]_N]_{NP} [+Sg]PRES NEG [leave]_V]_S]_Y T:AG
3. [[[John]_N]_{NP} [+Sg]PRES [prefer]_V [it]_N]_Y [+C [[Bill]_N]_{NP} +C [[+Sg]PRES]_T NEG [leave]_V]_S]_Y T:CDUP
4. [[[John]_N]_{NP} [+Sg]PRES [prefer]_V [it]_N]_Y [+C [Bill]_N NEG +C [[+Sg]PRES]_T [leave]_V]_S]_Y T:CNEG
5. [[[John]_N]_{NP} [[+Sg]PRES]_T [prefer]_V [it]_N]_Y [+C [Bill]_N NEG +C [leave]_V]_S]_Y T:TS
6. [[[John]_N]_{NP} [prefer]_V [[+Sg]PRES]_T [it]_N]_Y [+C [Bill]_N NEG +C [leave]_V]_S]_Y T:AF
7. [[[John]_N]_{NP} [prefer [+Sg]PRES]_V]_Y [+C [Bill]_N NEG +C [leave]_V]_S]_Y T:PD

Post Cycle:

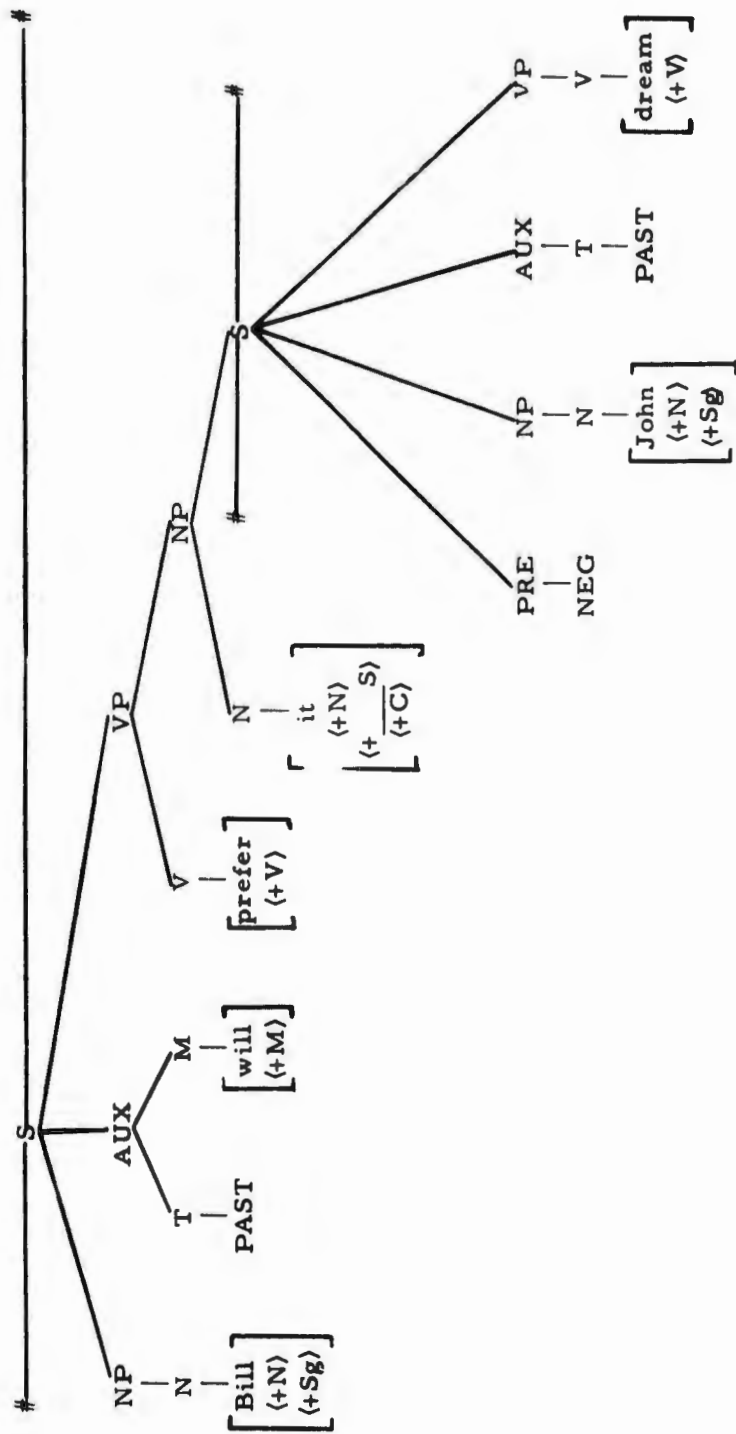
1. [[[John]_N]_{NP} [prefer[s]_T]_Y]_Y [+C [Bill]_N NEG +C [leave]_V]_S]_Y T:NUAG
2. [[[John]_N]_{NP} [prefer[s]_T]_Y]_Y [+C [[Bill]_N]_{NP} not +C [leave]_V]_S]_Y T:NEGSPELL
3. [[[John]_N]_{NP} [prefer[s]_T]_Y]_Y [for [Bill]_N not +C [leave]_V]_S]_Y T:C 1
4. [[[John]_N]_{NP} [prefer[s]_T]_Y]_Y [for [Bill]_N not to [leave]_V]_S]_Y T:C 2

Sentence Type No. 41

Actual string: Bill would prefer for John not to have dreamed

Derived string: Bill will ed prefer for John not to have dream en

Underlying structure:



Derivation:

First Cycle:

1. X [NEG [[John]_N]_{NP} [[[+Sg] PAST]_T]_{AUX} [dream]_V]_S ^{T:AG} Y
2. X [[[John]_N]_{NP} [[[+Sg] PAST]_T]_{AUX} NEG [dream]_V]_S ^{T:NEGPLACE} Y

Second Cycle:

1. [[[Bill]_N]_{NP} PAST [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} [+Sg] PAST NEG [dream]_V]_S]_S ^{T:CP 1}
2. [[[Bill]_N]_{NP} [+Sg] PAST [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} [+Sg] PAST NEG [dream]_V]_S]_S ^{T:AG}
3. [[[Bill]_N]_{NP} [+Sg] PAST [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} +C [[+Sg] PAST]_T NEG [dream]_V]_S]_S ^{T:CDUP}
4. [[[Bill]_N]_{NP} [+Sg] PAST [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} NEG +C [[+Sg] PAST]_T [dream]_V]_S]_S ^{T:CNEG}
5. [[[Bill]_N]_{NP} [+Sg] PAST [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} NEG +C have en [[+Sg] PAST]_T [dream]_V]_S]_S ^{T:CTENSE}
6. [[[Bill]_N]_{NP} [[+Sg] PAST]_T [will]_M [prefer]_V [it]_{<+C>}]_N [+C [[John]_N]_{NP} NEG +C have en [dream]_S]_S]_S ^{T:TS}
7. [[[Bill]_N]_{NP} [will]_M [[+Sg] PAST]_T]_M [prefer]_V [it]_{<+S>}]_N [+C [[John]_N]_{NP} NEG +C have [dream + en]_S]_S]_S ^{T:AF}
8. [[[Bill]_N]_{NP} [will]_M [[+Sg] PAST]_T]_M [prefer]_V [+C [[John]_N]_{NP} NEG +C have [dream + en]_S]_S]_S ^{T:PD}

Post Cycle:

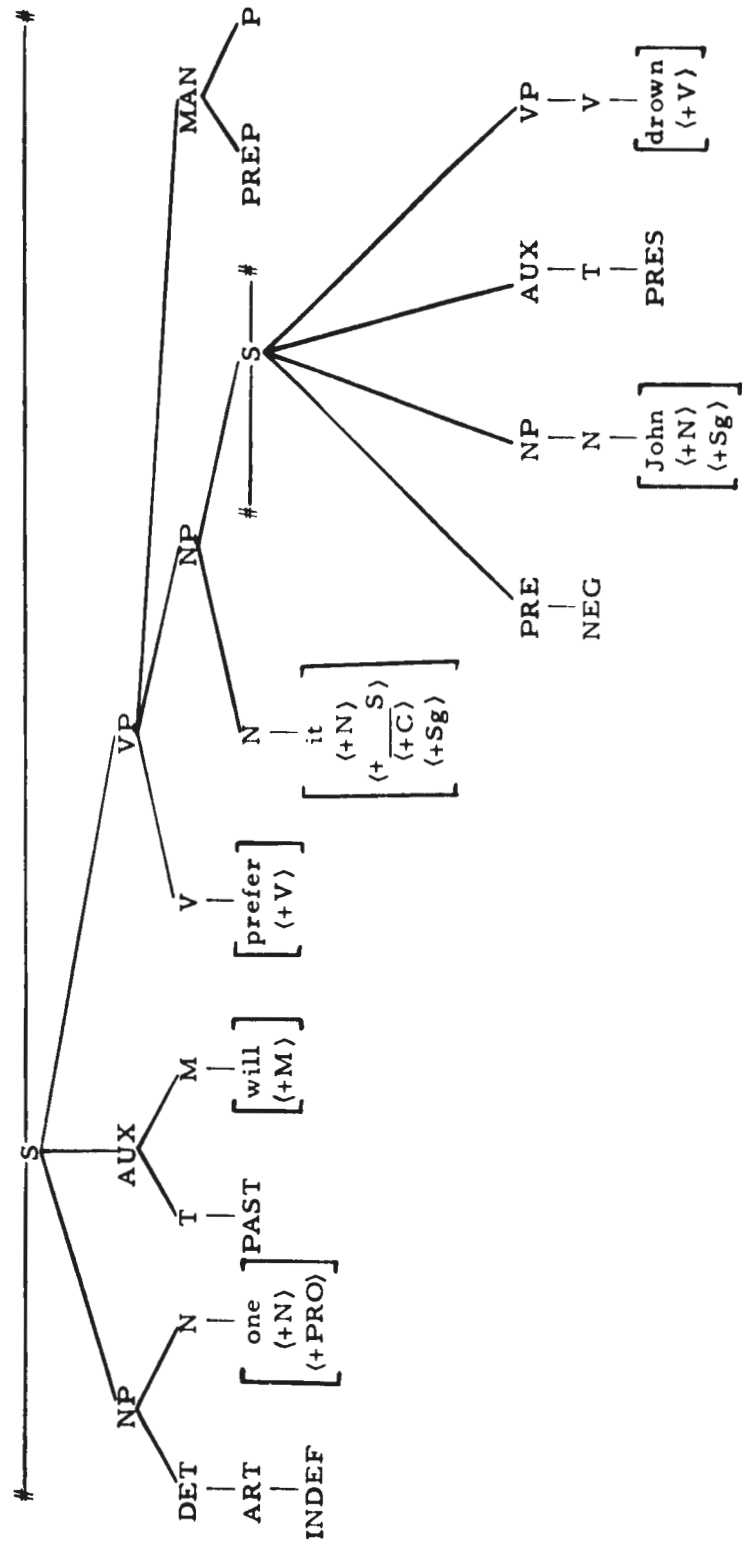
1. [[[Bill]_N]_{NP} [will]_M [ed]_T]_M [prefer]_V [+C [[John]_N]_{NP} NEG +C have [dream en]_V]_S]_S ^{T:PAST}
2. [[[Bill]_N]_{NP} [will]_M [ed]_T]_M [prefer]_V [+C [[John]_N]_{NP} not +C have [dream en]_V]_S]_S ^{T:NEGSPELL}
3. [[[Bill]_N]_{NP} [will]_M [ed]_T]_M [prefer]_V [for [[John]_N]_{NP} not +C have [dream en]_V]_S]_S ^{T:C 1}
4. [[[Bill]_N]_{NP} [will]_M [ed]_T]_M [prefer]_V [for [[John]_N]_{NP} not to have [dream en]_V]_S]_S ^{T:C 2}

Sentence Type No. 42

Actual string: for John not to drown would be preferred

Derived string: for John not to drown will ed be prefer en

Underlying structure:



Derivation:

First Cycle:

1. x [NEG [$\begin{bmatrix} \text{John} \\ \langle +Sg \rangle \end{bmatrix}_{NP}$]] $\begin{bmatrix} \text{PRES} \\ \langle +V \rangle \end{bmatrix}_{T}$ AUX $\begin{bmatrix} \text{drown} \\ \langle +V \rangle \end{bmatrix}_{V}$ y T:AG
2. x [[[John] $\begin{bmatrix} \text{PRES} \\ \langle +Sg \rangle \end{bmatrix}_{NP}$]] $\begin{bmatrix} \text{NEG} \\ \langle +V \rangle \end{bmatrix}_{T}$ AUX $\begin{bmatrix} \text{drown} \\ \langle +V \rangle \end{bmatrix}_{V}$ y T:NEGPLACE

Second Cycle:

1. $[[[INDEFone]_N]_{NP}[PAST[will]]_M]_{AUX}[prefer]_V \left[\begin{matrix} it \\ \langle +Sg \rangle \end{matrix} \right]_N [+C[[John]_N]_{NP}[[+Sg]_{PRES}]_T NEG[drown]_V]_{S[NP]_{PREPP}]_S$
T:CP 1
2. $[[\left[\begin{matrix} it \\ \langle +Sg \rangle \end{matrix} \right]_N [+C[John]_N]_{NP}[+Sg]_{PRES} NEG[drown]_V]_S]_{NP} PAST[will]_M be en [prefer]_V [PREP[INDEF[one]_N]_{NP}MAN]_S$
T:PASSIVE
3. $[[\left[\begin{matrix} it \\ \langle +Sg \rangle \end{matrix} \right]_N [+C[[John]_N]_{NP}[+Sg]_{PRES} NEG[drown]_V]_S]_{NP}[+Sg]_{PAST}[will]_M be en [prefer]_V [PREP INDEF one]_{MAN}]_S$
T:AG
4. $[[it]_N [+C[[John]_N]_{NP} + C[[+Sg]_{PRES}]_T NEG[drown]_V]_S [+Sg]_{PAST}[will]_M be en [prefer]_V [PREF INDEF one]_{MAN}]_S$
T:CDUP
5. $[[it]_N [+C[John]_N]_{NP} NEG + C[[+Sg]_{PRES}]_T [drown]_V]_S [+Sg]_{PAST}[will]_M be en [prefer]_V [PREP[INDEF[one]_N]_{NP}MAN]_S$
T:CNEG
6. $[[it]_N [+C[John]_N]_{NP} NEG + C[drown]_V]_S [[+Sg]_{PAST}]_T \left| \begin{matrix} will \\ \langle +M \rangle \end{matrix} \right|_M be en \left| \begin{matrix} prefer \\ \langle +V \rangle \end{matrix} \right|_V [PREP[INDEF[one]_N]_{NP}MAN]_S$
T:TS
7. $\left[\left[\begin{matrix} it \\ \langle +Sg \rangle \end{matrix} \right]_N [+C [John]_N]_{NP} NEG + C [drown]_V]_S \left[\begin{matrix} will + \\ \langle +M \rangle \end{matrix} \right]_S [+Sg]_{PAST}]_T be \left[\begin{matrix} prefer + en \\ \langle +V \rangle \end{matrix} \right]_V [PREP INDEF[one]_N]_{NP}MAN]_S$
T:AF
8. $[[+C [John]_N]_{NP} NEG + C [drown]_V]_S [will[+Sg]_{PAST}]_M be [prefer en]_V [PREP INDEF \left[\begin{matrix} one \\ \langle +PRO \rangle \end{matrix} \right]_{MAN}]_S$
T:PD
9. $[[+C [John]_N]_{NP} NEG + C [drown]_V]_S \left[\begin{matrix} will \\ \langle +M \rangle \end{matrix} \right]_S [+Sg]_{PAST}]_T be [prefer en]_V]_S$
T:AGDEL

Post Cycle:

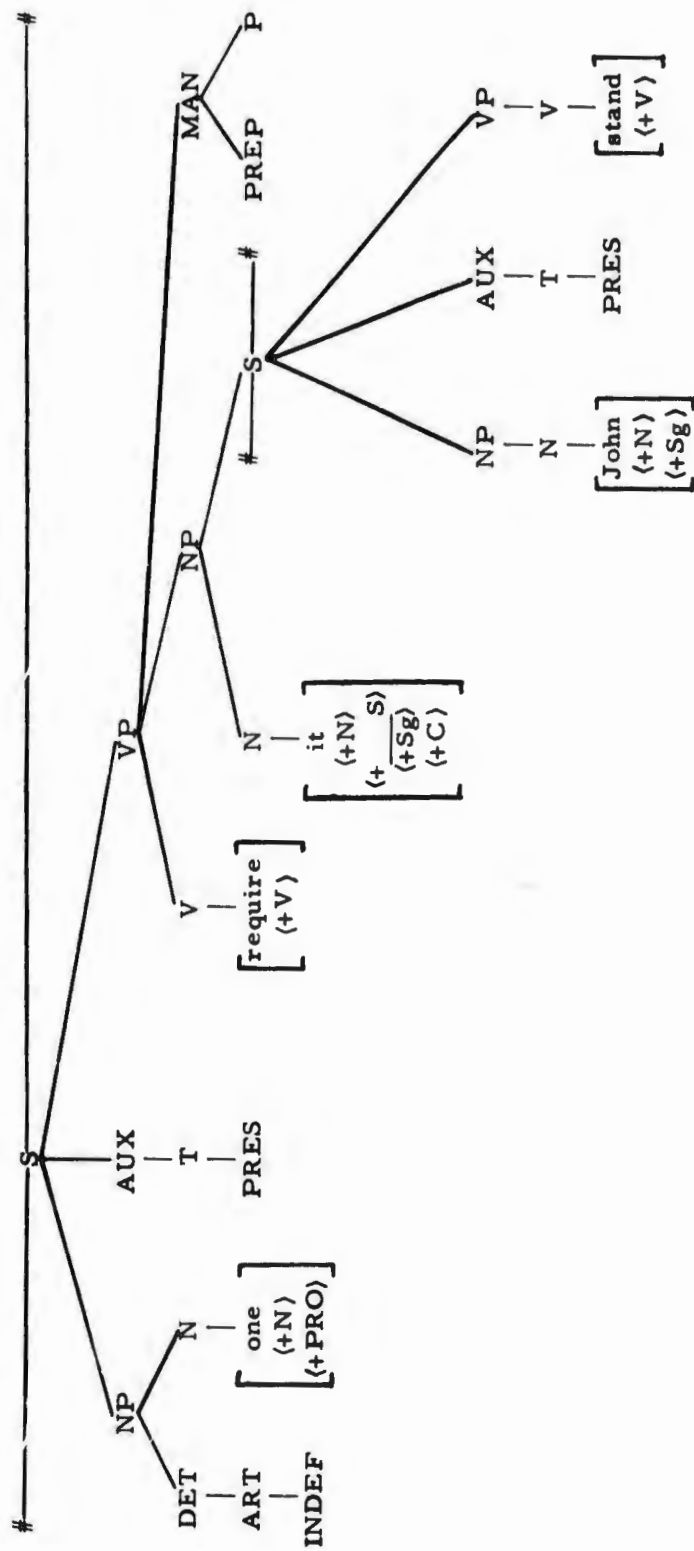
1.	$[[+C [John]_N \text{ NEG } +C [\text{drown}]_V]_S \left[\begin{smallmatrix} \text{will} [\text{ed}]_T \\ \text{ } \end{smallmatrix} \right]_{\langle +M \rangle}]_M$	T:PAST
2.	$[[+C [[John]_{N'} \text{ not } +C [\text{drown}]_V]_S [\text{will ed}]_M \text{ be } [\text{prefer en}]_V]_S$	T:NEGSPELL
3.	$[[\text{for } [[John]_{N'} \text{ not } +C [\text{drown}]_V]_S [\text{will ed}]_M \text{ be } [\text{prefer en}]_V]_S$	T:C 1
4.	$[[\text{for } [John]_N \text{ not to } [\text{drown}]_V]_S [\text{will ed}]_M \text{ be } [\text{prefer en}]_V]_S$	T:C 2

Sentence Type No. 43

Actual string: it is required for John to stand

Derived string: it is require en for John to stand

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\left[\text{John} \right]_{NP} \left[[+Sg] \text{PRES} \right]_T \left[\left[\text{stand} \right]_{(+V)} \right]_V \right]_S \right]_Y$ T:AG
2. $X \left[\left[\left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{(+V)} \right]_{NP} \left[[+Sg] \text{PRES} \right]_T \right]_S \right]_Y$ T:AF

Second Cycle:

1. $\left[\left[\text{INDEF} \left[\text{one} \right]_{NP} \left[\left[\text{PRES} \right]_T \right]_{AUX} \left[\text{require} \right]_V \left[\left[\text{it} \right]_{(+C)} \right]_N \left[+C \left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_{VP} \left[\text{PREP } P \right]_S \right]_{CP} \right]_1$ T:CP 1
2. $\left[\left[\left[\left[\text{it} \right]_{(+C)} \right]_{NP} \left[+C \left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_{VP} \left[\text{require} \right]_V \left[\text{PREP} \left[\text{INDEF} \left[\text{one} \right]_{NP} \left[\text{MAN} \right]_S \right]_{PASSIVE} \right]_{T:PASSIVE} \right]_{T:CP} \right]_N$ T:PASSIVE
3. $\left[\left[\left[\left[\text{it} \right]_{(+C)} \right]_{NP} \left[\left[\text{PRES} \right]_T \right]_{AUX} \text{be en} \left[\text{require} \right]_V \text{PREP INDEF} \left[\text{one} \right]_N \left[+C \left[\left[\text{John} \right]_N \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_{VP} \right]_S \right]_{T:EXTRA}$ T:EXTRA
4. $\left[\left[\left[\left[\text{it} \right]_{(+Sg)} \right]_{NP} \left[\left[\text{PRES} \right]_T \text{be} \right]_{AUX} \text{en} \left[\text{require} \right]_V \text{PREP INDEF} \left[\text{one} \right]_N \left[+C \left[\left[\text{John} \right]_N \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_{VP} \right]_S \right]_{T:AUXFILL}$ T:AUXFILL
5. $\left[\left[\left[\left[\text{it} \right]_{(+Sg)} \right]_{NP} \left[\left[[+Sg] \text{PRES} \right]_T \text{be} \right]_{AUX} \text{en} \left[\text{require} \right]_V \text{PREP INDEF} \left[\text{one} \right]_N \left[+C \left[\left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_{VP} \right]_S \right]_{T:AG}$ T:AG
6. $\left[\left[\left[\text{it} \right]_N \left[[+Sg] \text{PRES} \right]_{T:TS} \text{be en} \left[\text{require} \right]_V \text{PREP INDEF} \left[\text{one} \right]_N \left[+C \left[\left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{(+Sg)} \text{PRES} \right]_T \right]_S \right]_V \right]_{T:CDUP}$ T:CDUP
7. $\left[\left[\left[\text{it} \right]_N \left[\left[[+Sg] \text{PRES} \right]_T \text{be} \right]_{AUX} \text{en} \left[\left[\text{require} \right]_{(+V)} \right]_V \text{PREP INDEF} \left[\text{one} \right]_N \left[+C \left[\left[\text{John} \right]_N \left[\text{stand} \right]_{(+V)} \right]_V \right]_S \right]_{T:TS}$ T:TS
8. $\left[\left[\left[\text{it} \right]_N \left[\text{be} + \left[[+Sg] \text{PRES} \right]_T \right]_{AUX} \left[\left[\text{require} + \text{en} \right]_{(+V)} \right]_V \left[\text{PREP INDEF} \left[\left[\text{one} \right]_{MAN} \left[+C \left[\left[\text{John} \right]_N \left[\text{stand} \right]_{(+V)} \right]_V \right]_S \right]_{T:AF}$ T:AF
9. $\left[\left[\left[\text{it} \right]_N \left[\text{be} \right]_{(+Sg)} \text{PRES} \right]_{AUX} \left[\text{require en} \right]_V \left[+C \left[\left[\text{John} \right]_N \left[\text{stand} \right]_{VP} \right]_S \right]_{T:AGDEL}$ T:AGDEL

Post Cycle:

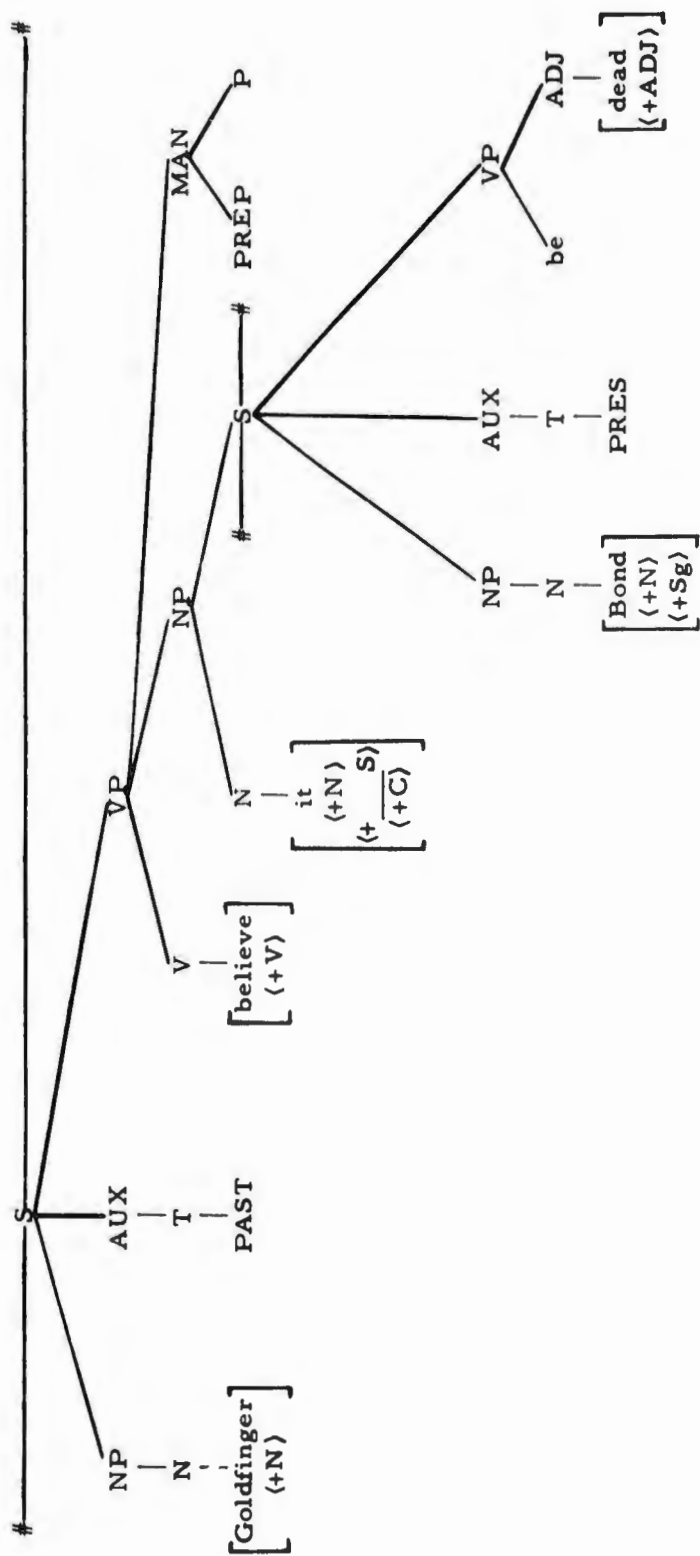
1. $\left[\left[\left[\text{it} \right]_N \left[\text{is} \right]_{AUX} \left[\text{require en} \right]_V \left[+C \left[\left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{VP} \right]_S \right]_{T:BE} \right]_1 \right]_{T:BE} \right]_1$ T:BE 1
2. $\left[\left[\left[\text{it} \right]_N \left[\text{is} \right]_{AUX} \left[\text{require en} \right]_V \left[\text{for} \left[\left[\left[\left[\text{John} \right]_{NP} \left[\text{stand} \right]_{VP} \right]_S \right]_{T:C} \right]_1 \right]_{T:C} \right]_1$ T:C 1
3. $\left[\left[\left[\text{it} \right]_N \left[\text{is} \right]_{AUX} \left[\text{require en} \right]_V \left[\text{for} \left[\left[\left[\text{John} \right]_N \left[\text{stand} \right]_{VP} \right]_S \right]_{T:C} \right]_2 \right]_{T:C} \right]_2$ T:C 2

Sentence Type No. 44

Actual string: Bond was believed to be dead by Goldfinger

Derived string: Bond was believe en to be dead by Goldfinger

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bond]_N]_{NP} [[PRES]_T be]_{AUX} [dead]_{ADJ}]_S ^Y
T:AUXFILL
2. X [[[Bond]_N]_{NP} [[[+Sg]PRES]_T be]_{AUX} [dead]_{ADJ}]_S ^Y
T:AG
3. X [[[Bond]_N]_{NP} [be + [[+ig]PRES]_T]_{AUX} [dead]_{ADJ}]_S ^Y
T:AF

Second Cycle:

1. [[[Goldfinger]_N]_{NP} [(PAST)_T]_{AUX} [believe]_V { [it]
T:CP 1
2. [[[[it
T:PASSIVE
3. [[[[it
T:EXTRA
4. [[[Bond]_N]_{NP} [(PAST)_T]_{AUX} be en [believe]_V PREP [Goldfinger]_N [+C [be [+Sg]PRES]_{AUX} [dead]_{ADJ}]_S]_S
T:PROREP
5. [[[[Bond]_N]_{NP} [[PAST]_T be]_{AUX} en [believe]_V PREP [Goldfinger]_N [+C [be [+Sg]PRES]_{AUX} [dead]_{ADJ}]_S]_S
T:AUXFILL
6. [[[[Bond]_N]_{NP} [[[+Sg]PAST]_T be]_{AUX} en [believe]_V PREP [Goldfinger]_N [+C [be [+Sg]PRES]_T]_{AUX} [dead]_{ADJ}]_S]_S
T:AG
7. [[[Bond]_N]_{NP} [[[+Sg]PAST]_T be]_{AUX} en [believe]_V PREP [Goldfinger]_N [+C [be]_{AUX} [dead]_{ADJ}]_S]_S
T:TS
8. [[[Bond]_N]_{NP} [be + [[+Sg]PAST]_T]_{AUX} [believe + en]_V PREP [Goldfinger]_N [+C [be]_{AUX} [dead]_{ADJ}]_{VP}]_S]_S
T:AF
9. [[[Bond]_N]_{NP} [be [+Sg]PAST]_{AUX} [believe en]_V [+C [be]_{AUX} [dead]_{ADJ}]_S [PREP [[Goldfinger]_N]_{NP}]_{MAN}]_S
T:VPCOMP

Post Cycle:

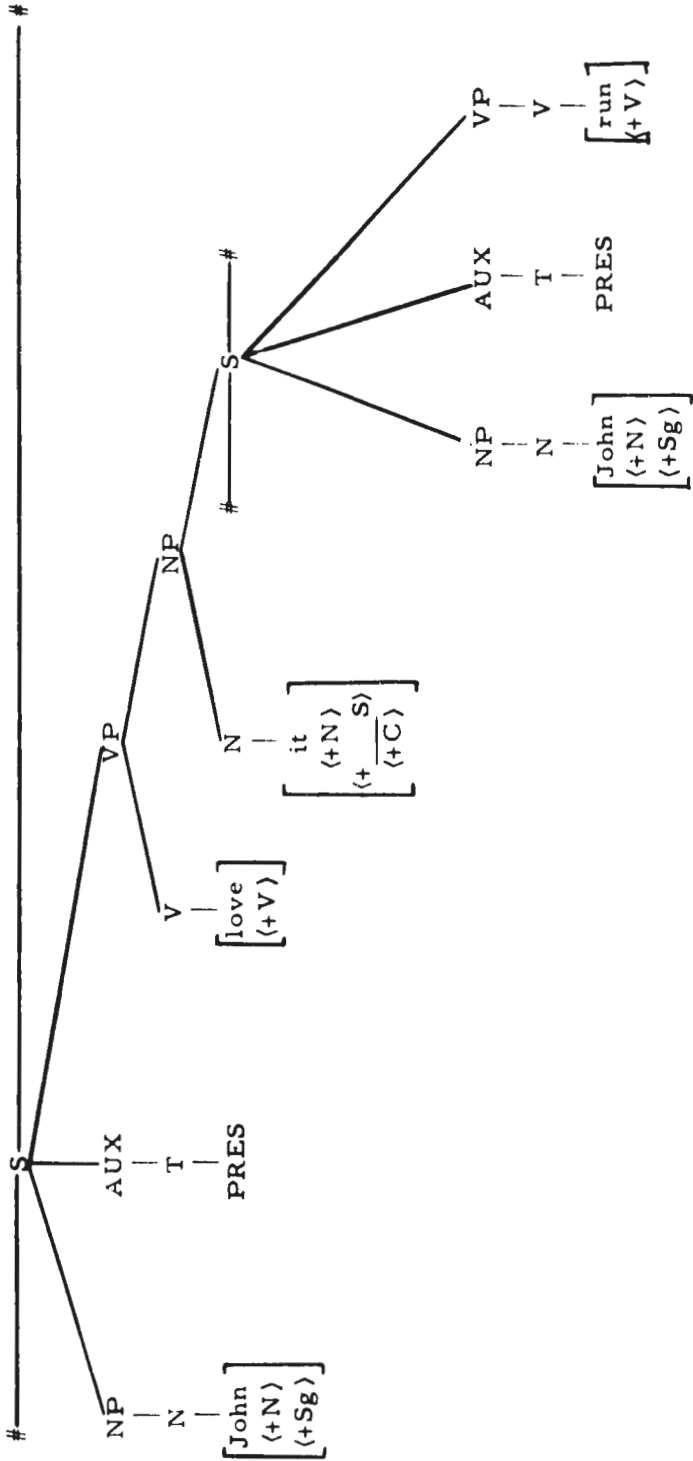
1. [[[Bond]_N]_{NP} [was]_{AUX} [believe en]_V [+C [be]_{AUX} [dead]_{ADJ}]_S [PREP [[Goldfinger]_N]_{NP}]_{MAN}]_S
T:BE 3
2. [[[Bond]_N]_{NP} [was]_{AUX} [believe en]_V [to [be]_{AUX} [dead]_{ADJ}]_S [PREP [[Goldfinger]_N]_{NP}]_{MAN}]_S
T:C 2
3. [[[Bond]_N]_{NP} [was]_{AUX} [believe en]_V [to [be]_{AUX} [dead]_{ADJ}]_S [by [[Goldfinger]_N]_{NP}]_{MAN}]_S
T:BY

Sentence Type No. 45

Actual string: John loves to run

Derived string: John love s to run

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\left[\text{John} \right]_{\text{NP}} \left[[+Sg]_{\text{PRES}} \right]_{\text{T}} \left[\left[\text{run} \right]_{\langle +V \rangle} \right]_{\text{S}} \right]_{\text{Y}} \right]_{\text{V}}$ T:AG
2. $X \left[\left[\left[\text{John} \right]_{\text{NP}} \left[\left[\text{run} \right]_{\langle +V \rangle} \right]_{\text{PRES}} \right]_{\text{T}} \right]_{\text{S}} \right]_{\text{Y}}$ T:AF

Second Cycle:

1. $\left[\left[\left[\text{John} \right]_{\text{NP}} \text{PRES} \left[\text{love} \right]_{\text{V}} \left[\left[\text{it} \right]_{\left[\left[\text{+C} \right]_{\text{NP}} \left[\text{run} \right]_{\text{+Sg}} \right]_{\text{PRES}} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{NP}} \right]_{\text{S}}$ T:CP 1
2. $\left[\left[\left[\text{John} \right]_{\text{NP}} \text{PRES} \left[\text{love} \right]_{\text{V}} \left[\text{it} \right]_{\text{N}} \left[\text{+C} \left[\text{run} \right]_{\text{+Sg}} \right]_{\text{PRES}} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}}$ T:IE
3. $\left[\left[\left[\text{John} \right]_{\text{NP}} \left[\text{+Sg} \right]_{\text{PRES}} \left[\text{love} \right]_{\text{V}} \left[\text{it} \right]_{\text{N}} \left[\text{+C} \left[\text{run} \right]_{\left[\left[\text{+Sg} \right]_{\text{PRES}} \right]_{\text{T}} \right]_{\text{S}} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}}$ T:AG
4. $\left[\left[\left[\text{John} \right]_{\text{NP}} \left[\left[\text{+Sg} \right]_{\text{PRES}} \right]_{\text{T}} \left[\left[\text{love} \right]_{\langle +V \rangle} \right]_{\text{V}} \left[\text{it} \right]_{\text{N}} \left[\text{+C} \left[\text{run} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}} \right]_{\text{S}}$ T:TS
5. $\left[\left[\left[\text{John} \right]_{\text{N}} \left[\left[\text{love} \right]_{\langle +V \rangle} \right]_{\text{PRES}} \right]_{\text{T}} \right]_{\text{V}} \left[\left[\text{it} \right]_{\left[\left[\text{+C} \right]_{\text{NP}} \left[\text{run} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}} \right]_{\text{S}}$ T:AF
6. $\left[\left[\left[\text{John} \right]_{\text{N}} \left[\left[\text{love} \right]_{\langle +V \rangle} \right]_{\text{PRES}} \right]_{\text{T}} \right]_{\text{V}} \left[\text{+C} \left[\text{run} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}}$ T:PD

Post Cycle:

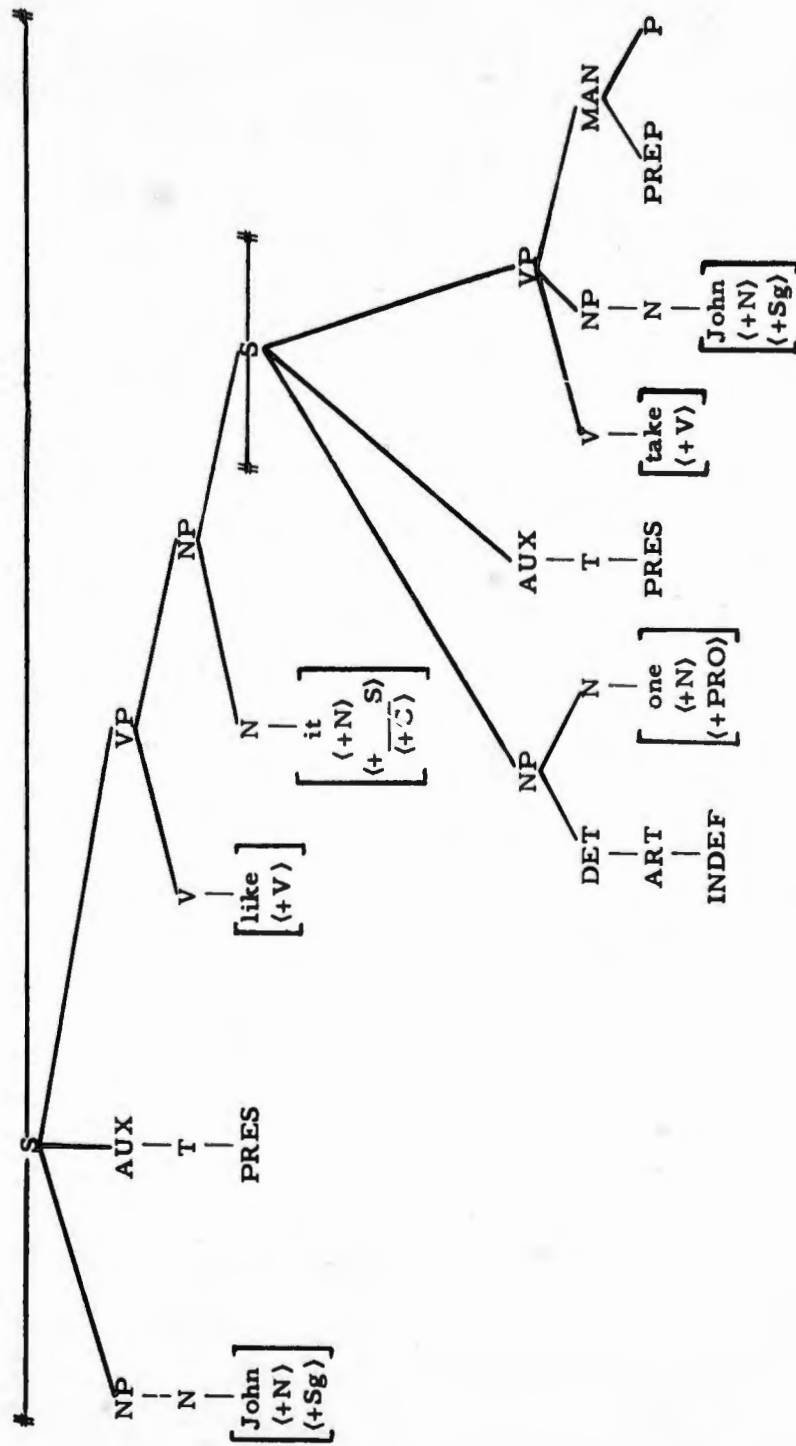
1. $\left[\left[\left[\text{John} \right]_{\text{N}} \left[\left[\text{love} \right]_{\langle +V \rangle} \right]_{\text{T}} \right]_{\text{V}} \left[\text{+C} \left[\text{run} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}}$ T:NUAG
2. $\left[\left[\left[\text{John} \right]_{\text{N}} \left[\text{love} \right]_{\text{V}} \left[\text{to} \left[\text{run} \right]_{\text{V}} \right]_{\text{S}} \right]_{\text{S}} \right]_{\text{S}}$ T:C 2

Sentence Type No. 46

Actual string: John likes to be taken

Derived string: John like s to be take en

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_N]_{NP}] PRES_T AUX be + en + [take]_V [PREP[INDEF[one]_N]_{NP}]_{MAN}'S_Y T: PASSIVE
2. X [[[John]_N]_{NP}] PRES_T be_{AUX} en [take]_V [PREP[INDEF[one]_N]_{NP}]_{MAN}'S_Y T: AUXFILL
3. X [[John]_N]_{NP}] PRES_T be_{AUX} en [take]_V [PREP[INDEF[one]_N]_{NP}]_{MAN}'S_Y T: AG
4. X [[[John]_N]_{NP}] PRES_T AUX [take + en]_V [PREP[INDEF[one]_N]_{NP}]_{MAN}'S_Y T: AF

Second Cycle:

1. [[[John]_N]_{NP}] PRES[like]_V [it]_N [+C [[John]_N]_{NP}] be [+Sg] PRES_{AUX} [take en]_V PREP INDEF[one]_N'S_{NP}'S_Y T: CP₁
2. [[[John]_N]_{NP}] PRES[like]_V [it]_N [+C [be [+Sg] PRES_{AUX} [take en]_V PREP INDEF [one]_N'S_{NP}'S_Y T: IE
3. [[John]_N]_{NP}] PRES [like]_V [it]_N [+C [be [[+Sg] PRES_T]_{AUX} [take en]_V PREP INDEF [one]_N'S_{NP}'S_Y T: AG
4. [[[John]_N]_{NP}] PRES_T [like]_V [it]_N [+C [be]_{AUX} [take en]_V PREP INDEF [one]_N'S_{NP}'S_Y T: TS
5. [[[John]_N]_{NP}] PRES_T [like]_V [it]_N [+C [be]_{AUX} [take en]_V PREP INDEF [one]_N'S_{NP}'S_Y T: AF
6. [[[John]_N]_{NP}] PRES_T [like]_V [it]_N [+C [be]_{AUX} [take en]_V [PREP INDEF [one]_N'S_{NP}'S_Y T: PD
7. [[[John]_N]_{NP}] PRES_T [like]_V [it]_N [+C [be]_{AUX} [take en]_V]_S'S_Y T: AGDEL

Post Cycle:

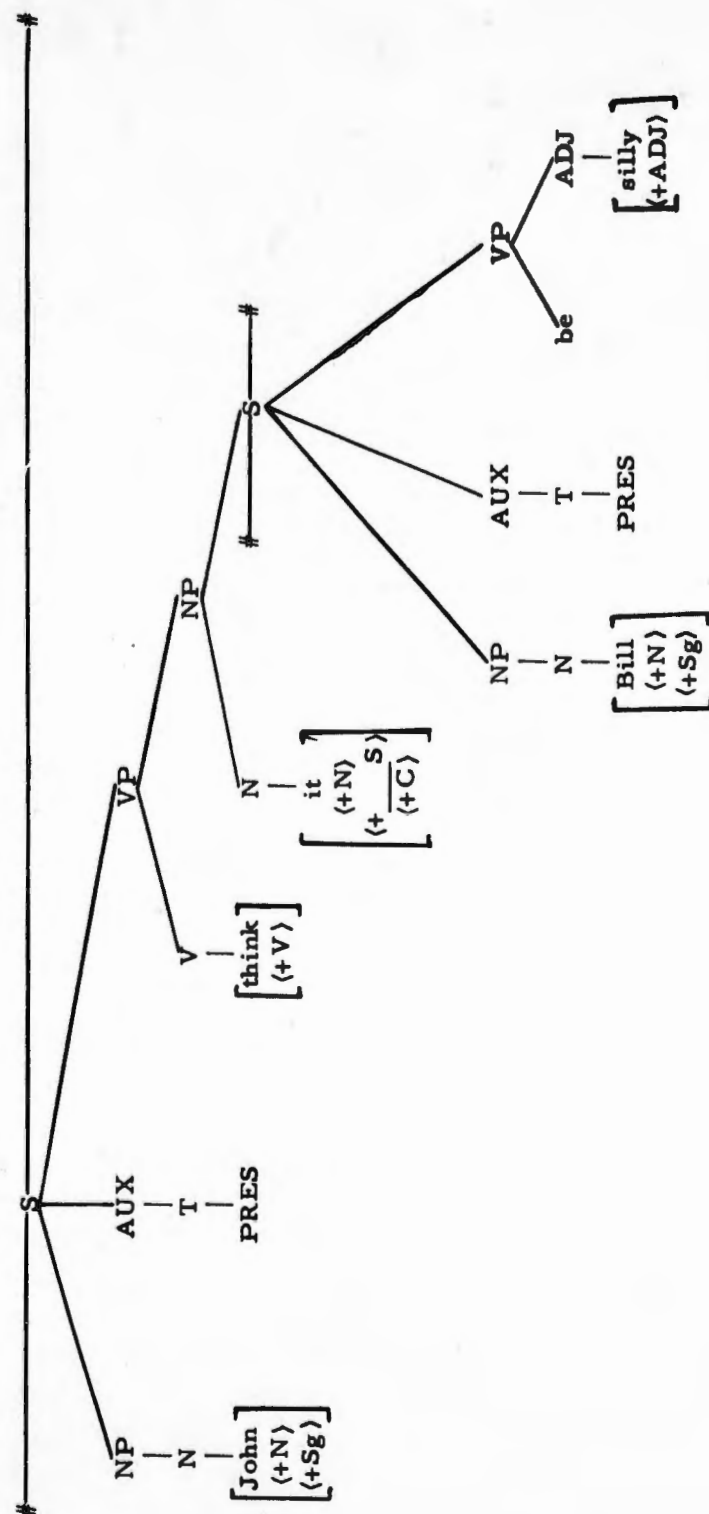
1. [[[John]_N]_{NP}] PRES_T [like]_V [it]_N [+C [be]_{AUX} [take en]_V]_S'S_Y T: NUAG
2. [[[John]_N]_{NP}] PRES_T [like]_V [to]_N [+C [be]_{AUX} [take en]_V]_S'S_Y T: C₂

Sentence Type No. 47

Actual string: John thinks Bill to be silly

Derived string: John think s Bill to be silly

Underlying structure:



First Cycle:

4. $X \left[\left[\text{Bill} \right]_{NP} \left[\left[\text{PRES}_T \text{ be} \right]_{AUX} \left[\text{silly} \right]_{ADJ} \right]_S \right]_N$
 (+Sg)

2. $X \left[\begin{array}{c} \text{Bill} \\ (+\text{Sg}) \end{array} \right]_{NP} \left[\begin{array}{c} [+Sg] \\ \text{PRES}_T \text{ be} \end{array} \right]_{AUX} \left[\begin{array}{c} \text{silly} \\ \text{ADJ}_S \end{array} \right]_Y$

3. $X \left[\left[\left[\text{Bill} \right]_N \right]_{NP} \left[\text{be} + \left[\left[+\text{Sg} \right]_{PRES-T} \right]_{AUX} \left[\text{silly} \right]_{ADJ} \right]_S Y$

Second Cycle:

1. [[John]_N PRES [think]_V [[it
(+C)
(+ S)]_N [+C [[Bill]_{NP} [be [+Sg] PRES_{AUX} [silly]_{ADJ}]_S NP_S]_{T:CP}]

2. $[[[\text{John}]_N \text{ PRES } [\text{think}]_V \left[\begin{array}{c} \text{it} \\ (+ \frac{S}{(+C)}) \end{array} \right]_N]_{NP} [+C \text{ } [[\text{Bill}]_N]_{NP} [\text{be } [+Sg] \text{ PRES }^{\text{AUX}} [\text{silly}]_{\text{ADJ}}]_{SS}]_{SS}]_{T:\text{EXTRA}}$

3. $[[[\text{John}]_{NP} \text{ PRES } [\text{think}]_V [[\text{Bill}]_{NP} [+C \text{ be } [+Sg]_{PRES} \text{ AUX } [\text{silly}]_{ADJ}]_{S'}]_{S'}]_N$ T:PROREP

4. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N \left[\begin{array}{c} [+Sg]_{PRES} \\ [+Sg]_{PRES} \end{array} \right]_V \left[\begin{array}{c} [think]_V \\ [[Bill]_{NP} \\ [+C \text{ be}]]_{NP} \end{array} \right]_{NP} \left[\begin{array}{c} [+Sg]_{PRES} \\ [+Sg]_{PRES} \end{array} \right]_T \left[\begin{array}{c} [silly]_{ADJ} \\ [silly]_{ADJ} \end{array} \right]_S$

5. $[[\text{John}]_{\text{N}}^{+Sg}]^N_T \text{ think } [\text{think}]_{(+V)}^V \text{ } [\text{Bill}]_{\text{N}}^{\text{NP}} \text{ } [+C \text{ be}]_{\text{AUX}}^{\text{ADJ}} \text{ } [\text{silly}]_{\text{ADJ}}^{\text{Sg}}$

6. $[[John]_N \text{ think } [+Sg]_{PRES_T} \text{ } [+V]_V \text{ } [[Bill]_{N_{NP}} [+C]_{AUX} \text{ } [silly]_{ADJ}]_{S_S}]_S$

Post Cycle:

1. $[[\text{John}]_N [\text{think}[s]_T]_V [\text{Bill}]_N [+C [\text{be}]_{\text{AUX}} [\text{silly}]_{\text{ADJ}}]_S]$

2. $[[[\text{John}]_N [\text{think s}]_V [\text{Bill}]_N [\text{to [be]}_{\text{AUX}} [\text{silly}]_{\text{ADJ}}]]]_S$

T:AUXFILL

T:AG

T:AF

T:CP 1

T:EXTRA

T:PROREP

T:AG

ST:I

T:AF

T:NUAG

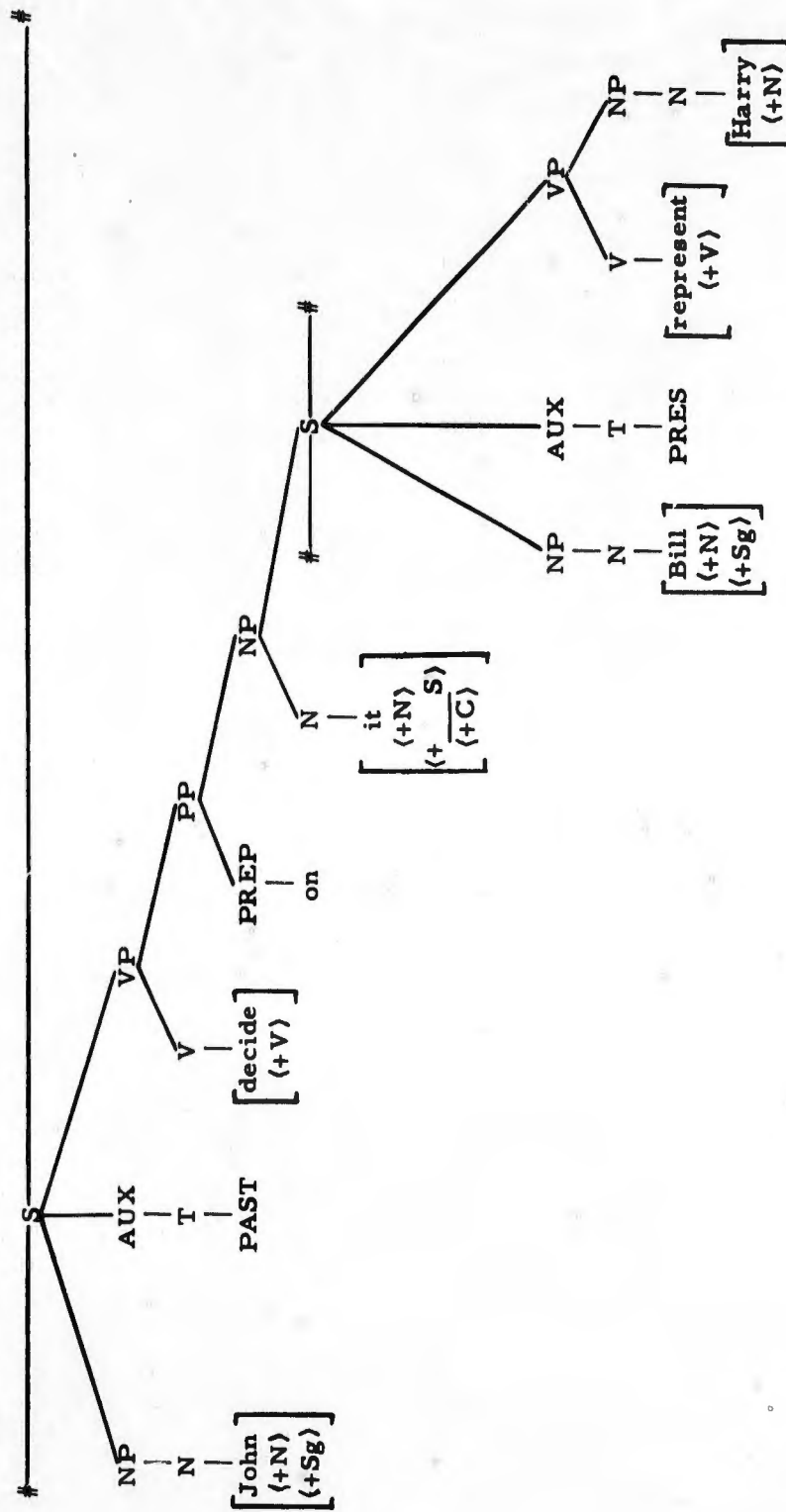
T:C 2

Sentence Type No. 48

Actual string: John decided for Bill to represent Harry

Derived string: John decide ed for Bill to represent Harry

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill] _{NP} [+Sg] PRES] _T [represent] _V] _N [Harry] _N _S _Y T:AG
2. X [[[Bill] _{NP} [represent] _V] _N] _T [Harry] _N _S _Y T:AF

Second Cycle:

1. [[[John] _{NP} PAST [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _{NP} [represent] _V] _N] _T [Harry] _N _S _{NP} _S T:CP 1
2. [[[John] _{NP} [+Sg] PAST [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _{NP} [represent] _V] _N] _T [Harry] _N _S _S T:AG
3. [[[John] _N [+Sg] PAST [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _{NP} +C [represent] _V] _N] _T [Harry] _N _S _S T:CDUP
4. [[[John] _N [+Sg] PAST] _T [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _N +C [represent] _V] _N] _T [Harry] _N _S _S T:TS
5. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _N +C [represent] _V] _N] _T [Harry] _N _S _S T:AF
6. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _N +C [represent] _V] _N] _T [Harry] _N _S _S T:PREPDEL
7. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _N +C [represent] _V] _N] _T [Harry] _N _S _S T:PD

Post Cycle:

1. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _{NP} +C [represent] _V] _N] _T [Harry] _N _S _S T:PAST
2. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _{NP} +C [represent] _V] _N] _T [Harry] _N _S _S T:C 1
3. [[[John] _N [decide] _V] _N] _T [[it] _N] _T [+C [[Bill] _N to [represent] _V] _N] _T [Harry] _N _S _S T:C 2

Actual string: John decided on Bill to represent Harry

Derived string: John decide ed on Bill to represent Harry

Diagram illustrating the hierarchical structure of the sentence "John decided on it that Bill represented Harry" using the Minimalist framework. The root node S branches into NP (John), VP (decided), and S (that Bill represented Harry). The NP "John" branches into NF (John) and N. The VP "decided" branches into V (decided) and PP (on it). The PP "on it" branches into PREP (on) and NP (it). The NP "it" branches into N (it) and a feature box containing (+N), (+S), and (+C). The S "that Bill represented Harry" branches into NP (Bill), AUX (T), and VP (represented). The NP "Bill" branches into N (Bill) and a feature box containing (+N) and (+Sg). The VP "represented" branches into V (represented) and NP (Harry). The NP "Harry" branches into N (Harry) and a feature box containing (+N).

Derivation:

First Cycle:

1. $X \left[\left[\left[\text{Bill} \right]_{N'} \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_Y$ T:AG
2. $X \left[\left[\left[\left[\text{Bill} \right]_{N'} \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_Y$ T:AF

Second Cycle:

1. $\left[\left[\left[\text{John} \right]_N \right]_{NP} \text{PAST} \left[\text{decide} \right]_V \left[\text{on} \right]_{PREP} \left[\left[\left[\text{it} \right]_{CP} \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:CP 1
2. $\left[\left[\left[\text{John} \right]_N \right]_{NP} \text{PAST} \left[\text{decide} \right]_V \left[\text{on} \right]_{PREP} \left[\left[\left[\text{it} \right]_{CP} \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:EXTRA
3. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \text{PAST} \left[\text{decide} \right]_V \left[\text{on} \right]_{PREP} \left[\left[\left[\text{it} \right]_{CP} \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S \right]_S$ T:PROREP
4. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:AG
5. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:TS
6. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:AF

Post Cycle:

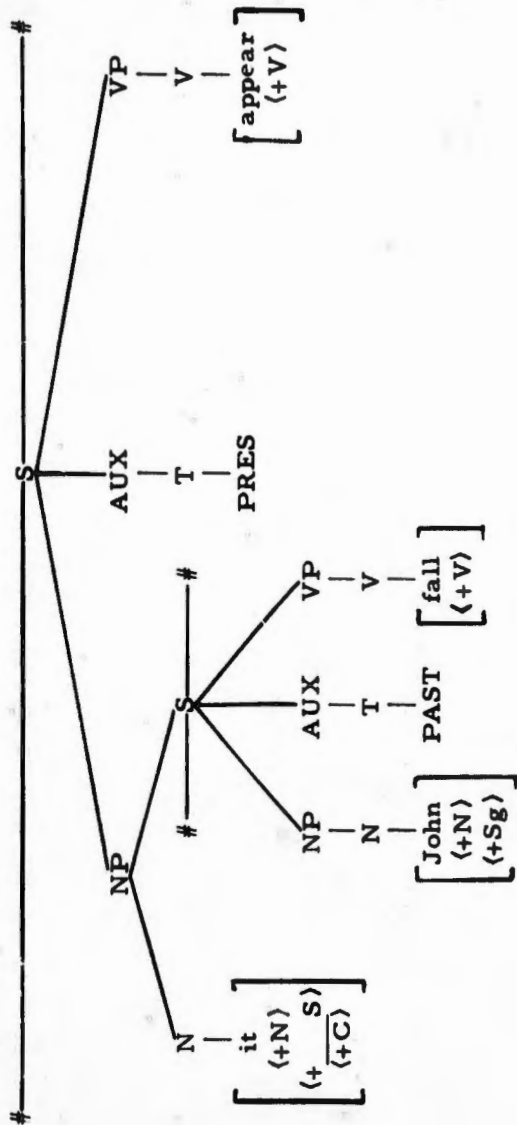
1. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:PAST
2. $\left[\left[\left[\left[\text{John} \right]_N \right]_{NP} \left[\left[\left[\text{represent} \right]_{T'} \right]_{VP} \right]_{VP} \left[\text{Harry} \right]_{N'} S \right]_S$ T:C 2

Sentence Type No. 50

Actual string: John appears to have fallen

Derived string: John appear s to have fall en

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\text{John} \right]_{\langle +Sg \rangle}^N \left[[+Sg]_{PAST-T} \right]_{S^Y} \right]_{S^Y}$
2. $X \left[\left[\left[\left[\text{John} \right]_{N^{NP}} \left[[+Sg]_{PAST-T} \right]_{S^Y} \right]_{\langle +V \rangle}^V \right]_{S^Y} \right]_{S^Y}$

T:AG

T:AF

Second Cycle:

1. $\left[\begin{array}{c} \text{it} \\ \langle +C \rangle \\ \langle +S \rangle \end{array} \right]_N \left[+C \left[[\text{John}]_N \text{NP} \left[\text{fall} [+Sg]_{\text{PAST}} \right]_V \right]_S \text{PRES} \left[\text{appear} \right]_V \right]_S$ T:CP 1
2. $\left[\left[\begin{array}{c} \text{it} \\ \langle +S \rangle \\ \langle +C \rangle \end{array} \right]_N \text{NP} \text{PRES} \left[\text{appear} \right]_V \left[+C \left[[\text{John}]_N \text{NP} \left[\text{fall} [+Sg]_{\text{PAST}} \right]_V \right]_S \right]_S \right]_S$ T:EXTRA
3. $\left[\left[\begin{array}{c} \text{John} \\ \langle +Sg \rangle \end{array} \right]_N \text{NP} \text{PRES} \left[\text{appear} \right]_V \left[+C \left[\text{fall} [+Sg]_{\text{PAST}} \right]_V \right]_S \right]_S$ T:PROREP
4. $\left[\left[\begin{array}{c} \text{John} \\ \langle +Sg \rangle \end{array} \right]_N \left[+Sg \right]_{\text{PRES}} \left[\text{appear} \right]_V \left[+C \left[\text{fall} [+Sg]_{\text{PAST}} \right]_V \right]_S \right]_S$ T:AG
5. $\left[[\text{John}]_N \left[+Sg \right]_{\text{PRES}} \left[\text{appear} \right]_V \left[+C \text{have en} \left[\begin{array}{c} \text{fall} [+Sg]_{\text{PAST}} \\ \langle +V \rangle \end{array} \right]_V \right]_S \right]_S$ T:CTENSE
6. $\left[[\text{John}]_N \left[[+Sg]_{\text{PRES}} \right]_T \left[\text{appear} \right]_V \left[+C \text{have en} \left[\begin{array}{c} \text{fall} \\ \langle +V \rangle \end{array} \right]_V \right]_S \right]_S$ T:TS
7. $\left[[\text{John}]_N \left[\text{appear} \left[\begin{array}{c} [+Sg]_{\text{PRES}} \\ \langle +V \rangle \end{array} \right]_T \right]_V \left[+C \text{have} \left[\begin{array}{c} \text{fall} + \text{en} \\ \langle +V \rangle \end{array} \right]_V \right]_S \right]_S$ T:AF

Post Cycle:

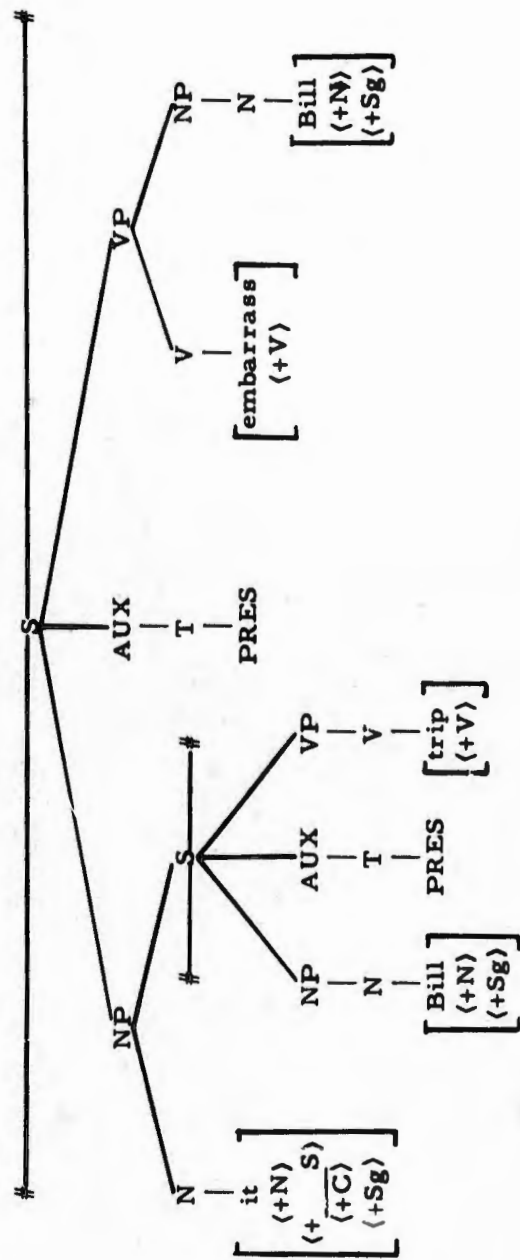
1. $\left[[\text{John}]_N \left[\text{appear} [s]_T \right]_V \left[+C \text{have} \left[\begin{array}{c} \text{fall en} \\ \langle +V \rangle \end{array} \right]_V \right]_S \right]_S$ T:NUAG
2. $\left[[\text{John}]_N \left[\text{appear } s \right]_V \left[\text{to have} \left[\text{fall en} \right]_V \right]_S \right]_S$ T:C 2

Sentence Type No. 51

Actual string: it embarrasses Bill to trip

Derived string: it embarrass s Bill to trip

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\text{Bill} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \right]_{S^Y}$

2. $X \left[\left[\left[\text{Bill} \right]_{NP} \left[\text{trip} \left[[+Sg]_{PRES} \right]_T \right]_{S^Y} \right]_{VP} \right]_{V^Y}$

T:AG

T:AF

Second Cycle:

1.	$\left[\begin{array}{c} \left[\begin{array}{c} \text{it} \\ \langle +C \rangle \end{array} \right] \\ \left[\begin{array}{c} +C \\ \langle +C \rangle \end{array} \right] \end{array} \right] \left[+C \left[[\text{Bill}]_N \right]_{NP} \left[\text{trip} \left[+Sg \right]_{PRES} \right]_V \right]_S \text{PRES} \left[\text{embarrass} \right]_V \left[[\text{Bill}]_N \right]_{NP} \right]_S$	T:CP 1
2.	$\left[\begin{array}{c} \left[\begin{array}{c} \text{it} \\ \langle + \frac{\text{S}}{\langle +C \rangle} \rangle \end{array} \right] \\ \left[\begin{array}{c} +C \\ \langle +C \rangle \end{array} \right] \end{array} \right] \left[+C \left[\text{trip} \left[+Sg \right]_{PRES} \right]_V \right]_S \text{PRES} \left[\text{embarrass} \right]_V \left[[\text{Bill}]_N \right]_{NP} \right]_S$	T:IE
3.	$\left[\left[\begin{array}{c} \left[\begin{array}{c} \text{it} \\ \langle + \frac{\text{S}}{\langle +C \rangle} \rangle \end{array} \right] \\ \left[\begin{array}{c} +C \\ \langle +Sg \rangle \end{array} \right] \end{array} \right] \right]_{NP} \text{PRES} \left[\text{embarrass} \right]_V \left[\text{Bill} \right]_N \left[+C \left[\text{trip} \left[+Sg \right]_{PRES} \right]_V \right]_S \right]_S$	T:EXTRA
4.	$\left[\left[\begin{array}{c} \left[\begin{array}{c} \text{it} \\ \langle +Sg \rangle \end{array} \right] \right]_{NP} \left[+Sg \right]_{PRES} \left[\text{embarrass} \right]_V \left[\text{Bill} \right]_N \left[+C \left[\text{trip} \left[+Sg \right]_{PRES} \right]_T \right] \right]_S \right]_S$	T:AG
5.	$\left[[\text{it}]_N \left[+Sg \right]_{PRES} \right]_T \left[\begin{array}{c} \left[\text{embarrass} \right] \\ \langle +V \rangle \end{array} \right]_V \left[\text{Bill} \right]_N \left[+C \left[\text{trip} \right]_V \right]_S \right]_S$	T:TS
6.	$\left[[\text{it}]_N \left[\begin{array}{c} \left[\text{embarrass} \left[+Sg \right]_{PRES} \right]_T \\ \langle +V \rangle \end{array} \right] \right]_V \left[\text{Bill} \right]_N \left[+C \left[\text{trip} \right]_V \right]_S \right]_S$	T:AF

Post Cycle:

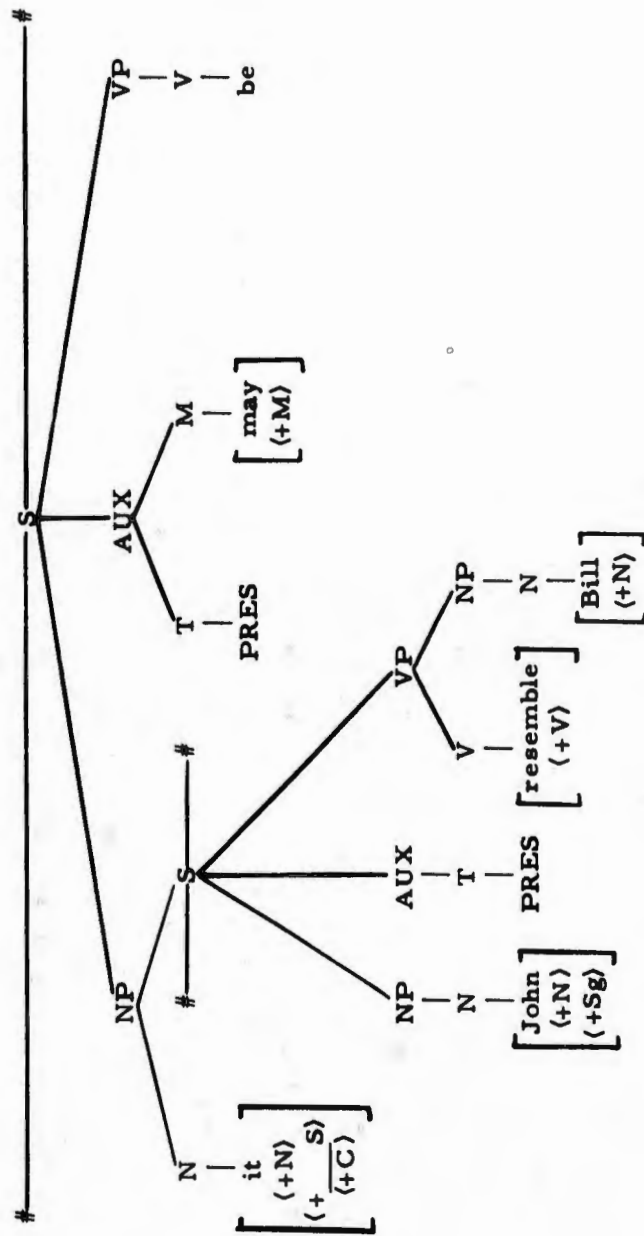
1. $[[[it]_N [\text{embarrass}[s]_T]_V [\text{Bill}]_N [+C [\text{trip}]_V]_S]$
2. $[[[it]_N [\text{embarrass } s]_V [\text{Bill}]_N [\text{to } [\text{trip}]_V]_S]$

Sentence Type No. 52

Actual string: John may resemble Bill

Derived string: John may resemble Bill

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_{NP}]_{NP}]_{NP} [[[+Sg] PRES]_T]_V [resemble]_V [Bill]_N]_S]_Y
2. X [[[John]_N]_{NP}]_{NP} [resemble_V]_V [[+Sg] PRES]_T]_V [Bill]_N]_S]_Y

T:AG

T:AF

Second Cycle:

1. $\left[\begin{array}{c} \text{it} \\ (+C) \\ (+S) \end{array} \right]_N$ [+C [[John]_NNP [resemble [+Sg]PRES]_V [Bill]_NS PRES [may]_M be]_S T:CP 1
2. $\left[\begin{array}{c} \text{it} \\ (+S) \\ (+C) \end{array} \right]_N$]_{NP} PRES [may]_MAUX be [+C [[John]_NNP [resemble [+Sg]PRES]_V [Bill]_NS]_S T:EXTRA
3. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} PRES [may]_M be [+C [resemble [+Sg]PRES]_V [Bill]_NS]_S T:PROREP
4. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES [may]_M be [+C [resemble [[+Sg]PRES]_T]_V [Bill]_NS]_S T:AG
5. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES]_T [may]_M be [+C [resemble]_V [Bill]_NS]_S T:TS
6. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES]_T [may]_M be [+C [resemble]_V [Bill]_NS]_S T:AF
7. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES]_T [may]_M be [+C [resemble]_V [Bill]_NS]_S T:BEDEL
8. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES]_T [may]_M be [+C [resemble]_V [Bill]_NS]_S T:MCDEL

Post Cycle:

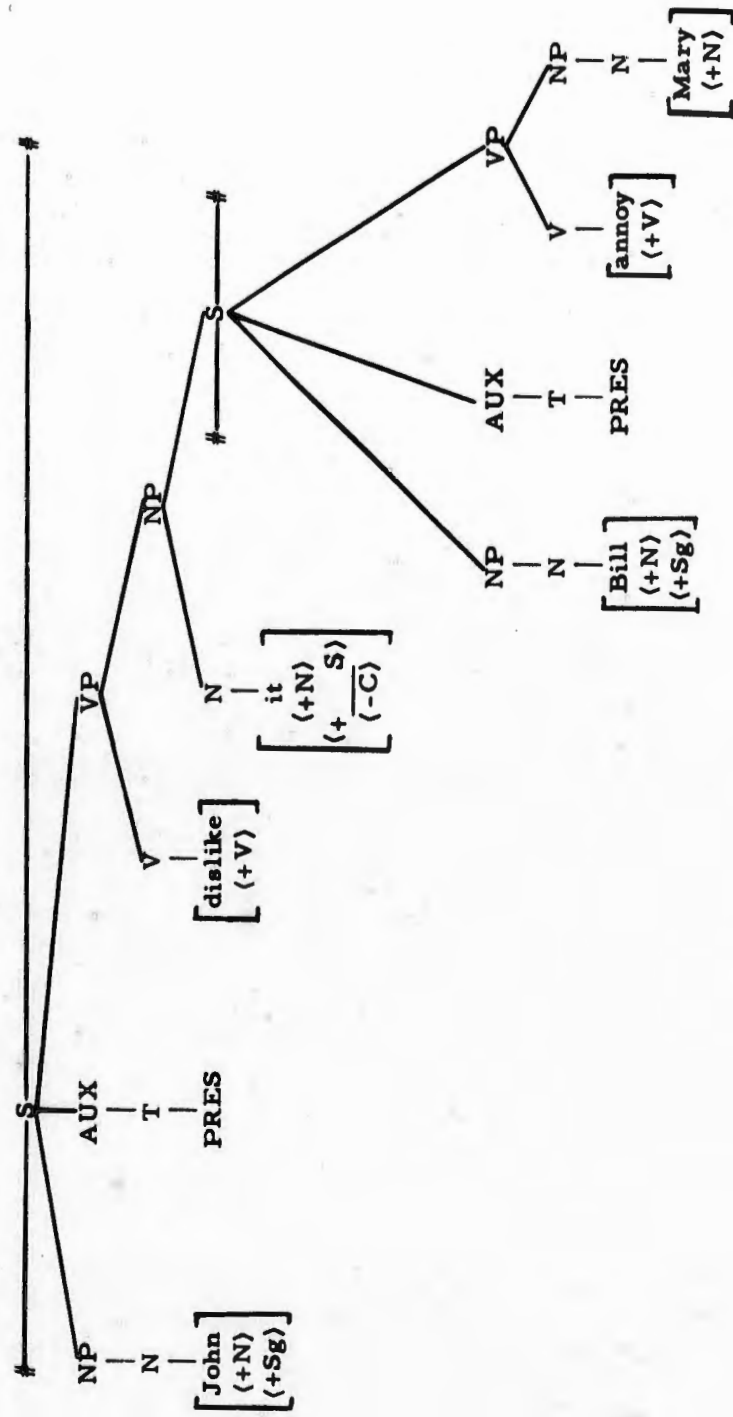
1. $\left[\begin{array}{c} \text{John} \\ (+Sg) \end{array} \right]_N$]_{NP} [+Sg]PRES]_T [may]_M be [+C [resemble]_V [Bill]_NS]_S T:MTDEL

Sentence Type No. 53

Actual string: John dislikes Bill's annoying Mary

Derived string: John dislike s Bill 's annoy ing Mary

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill]_N]_{NP} [[+Sg] PRES]_T [annoy]_N]_V [Mary]_N]_S Y T:AG
2. X [[[[Bill]_N]_{NP} [annoy]_N]_{NP} [[+Sg] PRES]_T]_V [Mary]_N]_S Y T:AF

Second Cycle:

1. [[[[John]_N]_{NP} PRES [dislike]_V [it]_N]_{NP} [-C [[Bill]_N]_{NP} [annoy [+Sg] PRES]_V [Mary]_N]_S]_S T:CP 1
2. [[[[John]_N]_{NP} [[+Sg] PRES [dislike]_V [it]_N]_{NP} [-C [[Bill]_N]_{NP} [annoy [+Sg] PRES]_V [Mary]_N]_S]_S T:AG
3. [[[John]_N]_{NP} [[+Sg] PRES [dislike]_V [it]_N]_{NP} [-C [[Bill]_N]_{NP} -C [annoy [[+Sg] PRES]_T]_V [Mary]_N]_S]_S T:CDUP
4. [[[John]_N]_{NP} [[+Sg] PRES]_T [dislike]_V [it]_N]_{NP} [-C [[Bill]_N]_{NP} -C [annoy [[+Sg] PRES]_T]_V [Mary]_N]_S]_S T:TS
5. [[[John]_N]_{NP} [dislike [[+Sg] PRES]_T]_V [it]_N]_{NP} [-C [[Bill]_N]_{NP} -C [annoy + -C]_V [Mary]_N]_S]_S T:AF
6. [[[John]_N]_{NP} [dislike [[+Sg] PRES]_T]_V [[[[Bill]_N]_{NP} -C [annoy -C]_V]_N]_{NP} [Mary]_N]_S]_S T:PD

Post Cycle:

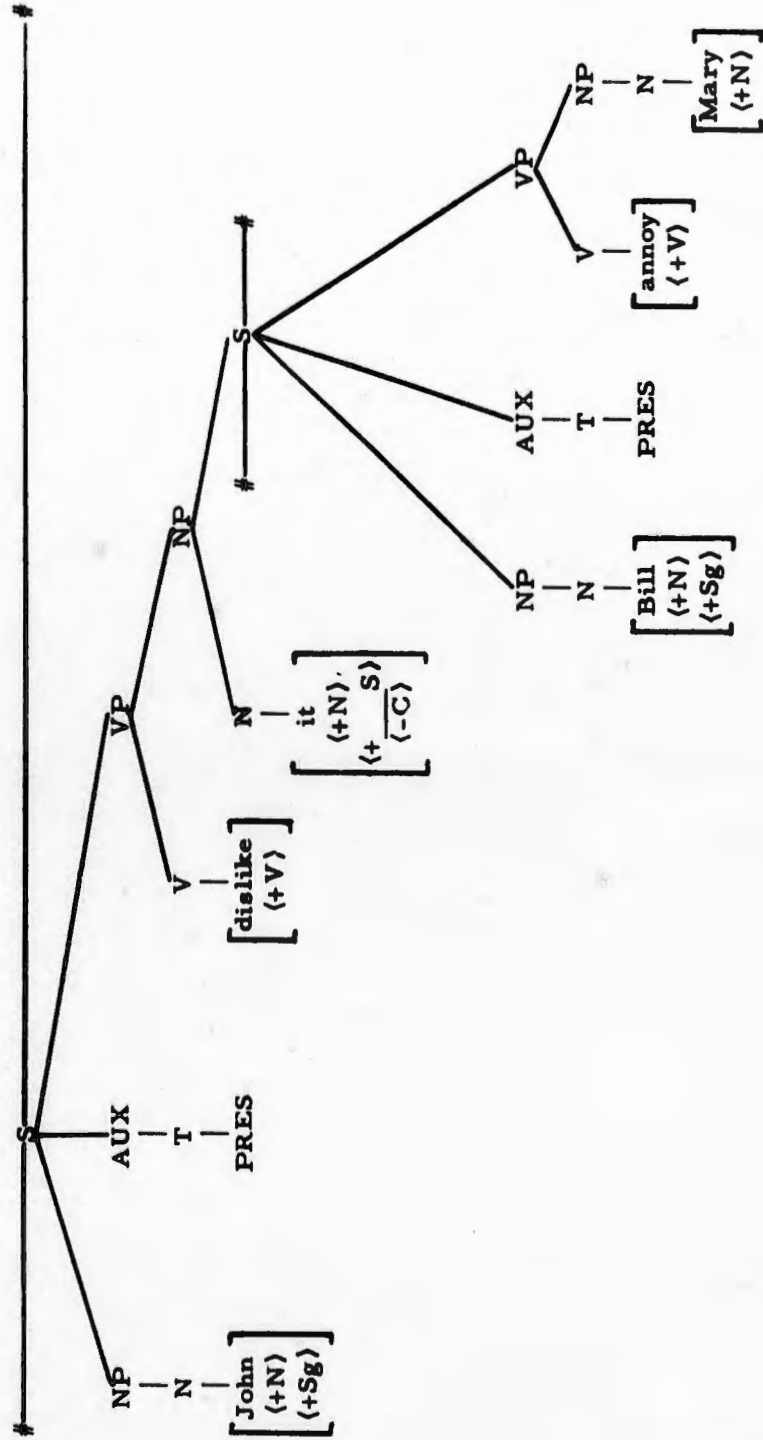
1. [[[John]_N]_{NP} [dislike [s]_T]_V [[[[Bill]_N]_{NP} -C [annoy -C]_V]_N]_{NP} [Mary]_N]_S]_S T:NUAG
2. [[[John]_N]_{NP} [dislike s]_V [[[[Bill]_N]_{NP} 's [annoy -C]_V]_N]_{NP} [Mary]_N]_S]_S T:C 3
3. [[[John]_N]_{NP} [dislike s]_V [[[[Bill]_N]_{NP} 's [annoy ing]_V]_N]_{NP} [Mary]_N]_S]_S T:C 4

Sentence Type No. 54

Actual string: John dislikes Bill annoying Mary

Derived string: John dislike s Bill annoy ing Mary

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill]_N]_{NP} [[+Sg]PRES]_T [annoy]_V [Mary]_N]_S Y T:AG
2. X [[[Bill]_N]_{NP} [annoy [[+Sg]PRES]_T [Mary]_N]_S Y T:AF

Second Cycle:

1. [[[John]_N]_{NP} PRES [dislike]_V [it]_N [-C [[Bill]_N]_{NP} [annoy [+Sg]PRES]_V [Mary]_N]_S]_S T:CP 1
2. [[[John]_N]_{NP} [+Sg]PRES [dislike]_V [it]_N [-C [[Bill]_N]_{NP} [annoy [+Sg]PRES]_V [Mary]_N]_S]_S T:AG
3. [[John]_N [+Sg]PRES [dislike]_V [it]_N [-C [[Bill]_N]_{NP} -C [annoy [[+Sg]PRES]_T]_V [Mary]_N]_S]_S T:CDUP
4. [[John]_N [+Sg]PRES [dislike]_V [it]_N [-C [[Bill]_N]_{NP} -C [annoy]_V [Mary]_N]_S]_S T:TS
5. [[John]_N [[+Sg]PRES]_T [dislike]_V [it]_N [it]_N [[[Bill]_N]_{NP} -C [annoy]_V [Mary]_N]_S]_S T:CD
6. [[John]_N [dislike [[+Sg]PRES]_T]_V [it]_N [it]_N [[[Bill]_N]_{NP} [annoy + -C]]_V [Mary]_N]_S]_S T:AF
7. [[John]_N [dislike [[+Sg]PRES]_T]_V [[[Bill]_N]_{NP} [annoy -C]]_V [Mary]_N]_S]_S T:PD

Post Cycle:

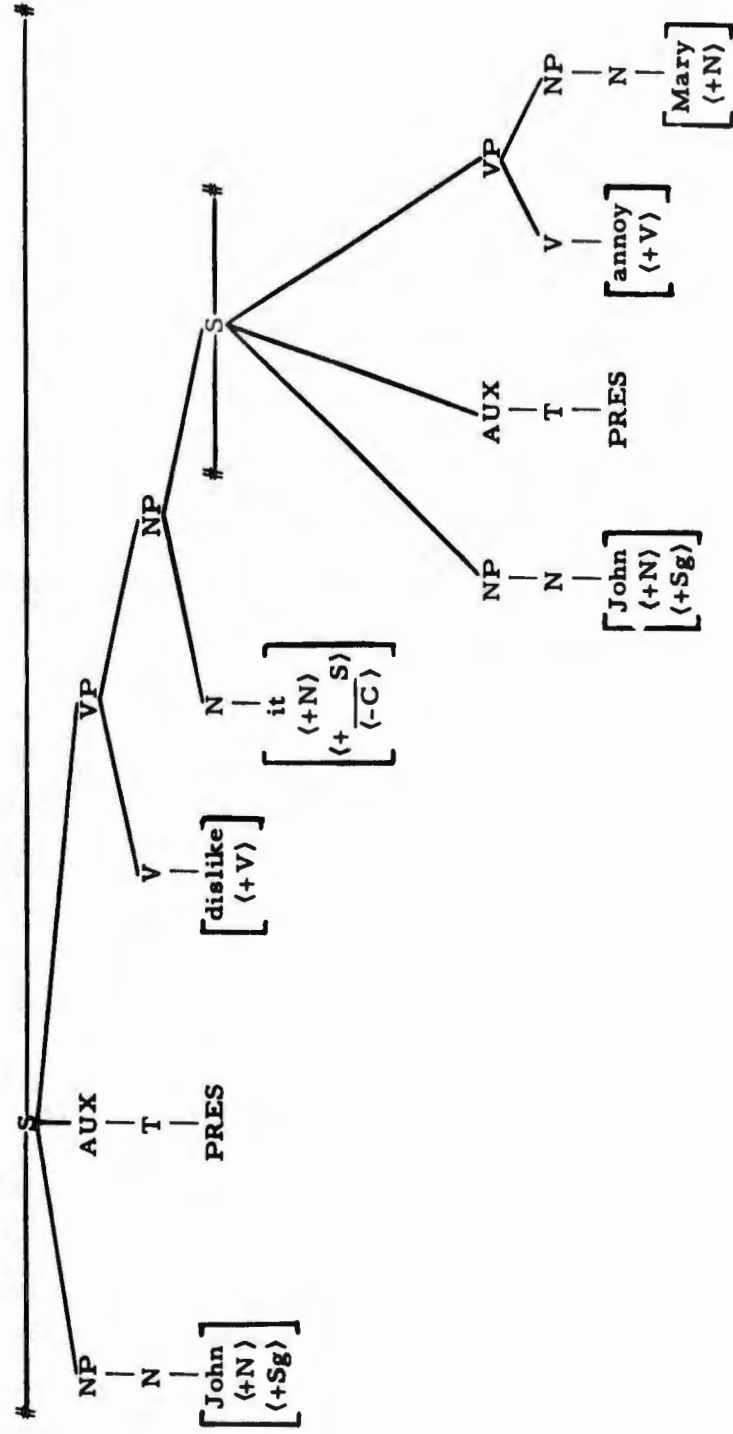
1. [[John]_N [dislike[s]_T]_V [Bill]_N [annoy -C]_V [Mary]_N]_S]_S T:NUAG
2. [[John]_N [dislike s]_V [Bill]_N [annoy ing]_V [Mary]_N]_S]_S T:C 4

Sentence Type No. 55

Actual string: John dislikes annoying Mary

Derived string: John dislike s annoy ing Mary

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\left[\text{John} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \left[\text{annoy} \right]_{(V)} \right]_V \left[\text{Mary} \right]_{N'S} Y$ T:AG
2. $X \left[\left[\left[\text{John} \right]_{NP} \left[\text{annoy} \right]_{(V)} \right]_V \left[[+Sg]_{PRES} \right]_T \left[\text{Mary} \right]_{N'S} Y$ T:AF

Second Cycle:

1. $\left[\left[\left[\text{John} \right]_{NP} \text{PRES} \left[\text{dislike} \right]_V \left[\text{it} \right]_{(-C)} \right]_N \left[-C \left[\left[\text{John} \right]_{NP} \left[\text{annoy} [+Sg]_{PRES} \right]_V \left[\text{Mary} \right]_{N'S} \right]_S \right]_{CP1}$ T:CP 1
2. $\left[\left[\left[\text{John} \right]_{NP} \text{PRES} \left[\text{dislike} \right]_V \left[\text{it} \right]_{(-C)} \right]_N \left[-C \left[\text{annoy} [+Sg]_{PRES} \right]_V \left[\text{Mary} \right]_{N'S} \right]_S$ T:IE
3. $\left[\left[\left[\text{John} \right]_{NP} [+Sg]_{PRES} \left[\text{dislike} \right]_V \left[\text{it} \right]_{(-C)} \right]_N \left[-C \left[\text{annoy} \left[[+Sg]_{PRES} \right]_T \left[\text{Mary} \right]_{N'S} \right]_S \right]_V$ T:AG
4. $\left[\left[\left[\text{John} \right]_{NP} [+Sg]_{PRES} \right]_T \left[\text{dislike} \right]_{(V)} \left[\text{it} \right]_{(-C)} \right]_N \left[-C \left[\text{annoy} \right]_{(V)} \left[\text{Mary} \right]_{N'S} \right]_S$ T:TS
5. $\left[\left[\left[\text{John} \right]_{NP} \left[\text{dislike} \left[[+Sg]_{PRES} \right]_T \right]_V \left[\text{it} \right]_{(+S)} \right]_N \left[\left[\text{annoy} + -C \right]_{(V)} \left[\text{Mary} \right]_{N'S} \right]_S$ T:AF
6. $\left[\left[\left[\text{John} \right]_{NP} \left[\text{dislike} \left[[+Sg]_{PRES} \right]_T \right]_V \left[\text{annoy} -C \right]_{(V)} \right]_V \left[\text{Mary} \right]_{N'S} \right]_S$ T:PD

Post Cycle:

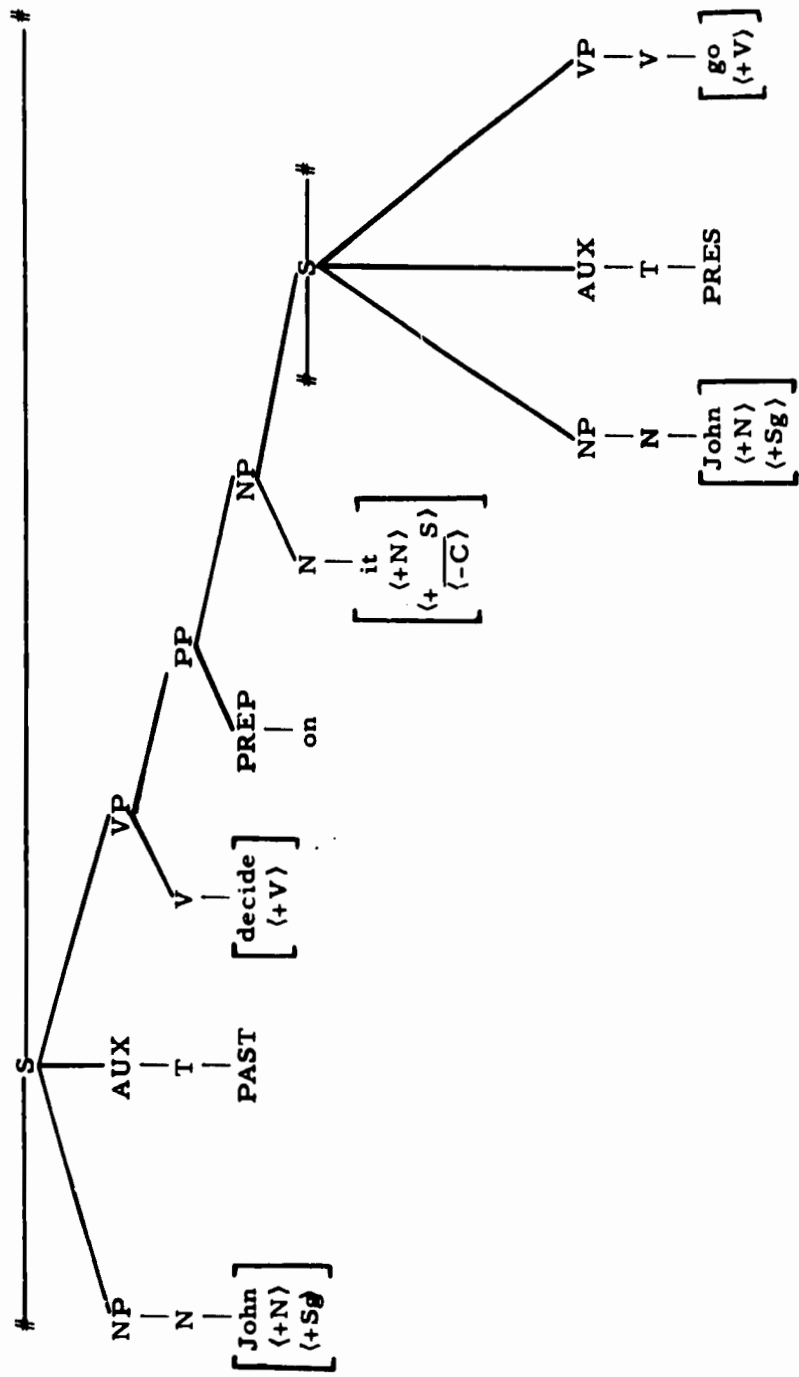
1. $\left[\left[\left[\text{John} \right]_{NP} \left[\text{dislike} [s] \right]_T \right]_V \left[\text{annoy} -C \right]_V \left[\text{Mary} \right]_{N'S} \right]_S$ T:NUAG
2. $\left[\left[\left[\text{John} \right]_{NP} \left[\text{dislike} s \right]_V \left[\text{annoy ing} \right]_V \left[\text{Mary} \right]_{N'S} \right]_S$ T:C 4

Sentence Type No. 56

Actual string: John decided on going

Derived string: John decide ed on go ing

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_N]_{NP} [[+Sg]PRES]_T [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S Y T:AG
2. X [[[[John]_N]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S]_{NP} [[+Sg]PRES]_T]_S Y T:AF

Second Cycle:

1. [[[[John]_N]_{NP} PAST [decide]_V [on]PREP [$\begin{smallmatrix} it \\ (-C) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S]_{NP} [[+Sg]PRES]_T]_S S T:CP 1
2. [[[[John]_N]_{NP} PAST [decide]_V [on]PREP [$\begin{smallmatrix} it \\ (-C) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S]_{NP} [[+Sg]PRES]_T]_S S T:IE
3. [[[[John]_N]_{NP} [+Sg]PAST [decide]_V [on]PREP [$\begin{smallmatrix} it \\ (-C) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S]_{NP} [[+Sg]PRES]_T]_S S T:AG
4. [[[John]_N]_{NP} [[+Sg]PAST]_T [decide]_V [on]PREP [$\begin{smallmatrix} it \\ (-C) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S S T:TS
5. [[John]_N]_{NP} [decide [[+Sg]PAST]_T]_V [on]PREP [$\begin{smallmatrix} it \\ (+-S) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S S T:AF
6. [[John]_N]_{NP} [decide [[+Sg]PAST]_T]_V [on]PREP [$\begin{smallmatrix} go \\ (-C) \end{smallmatrix}$]_N]_S]_{NP} [$\begin{smallmatrix} go \\ (+V) \end{smallmatrix}$]_V]_S S T:PD

Post Cycle:

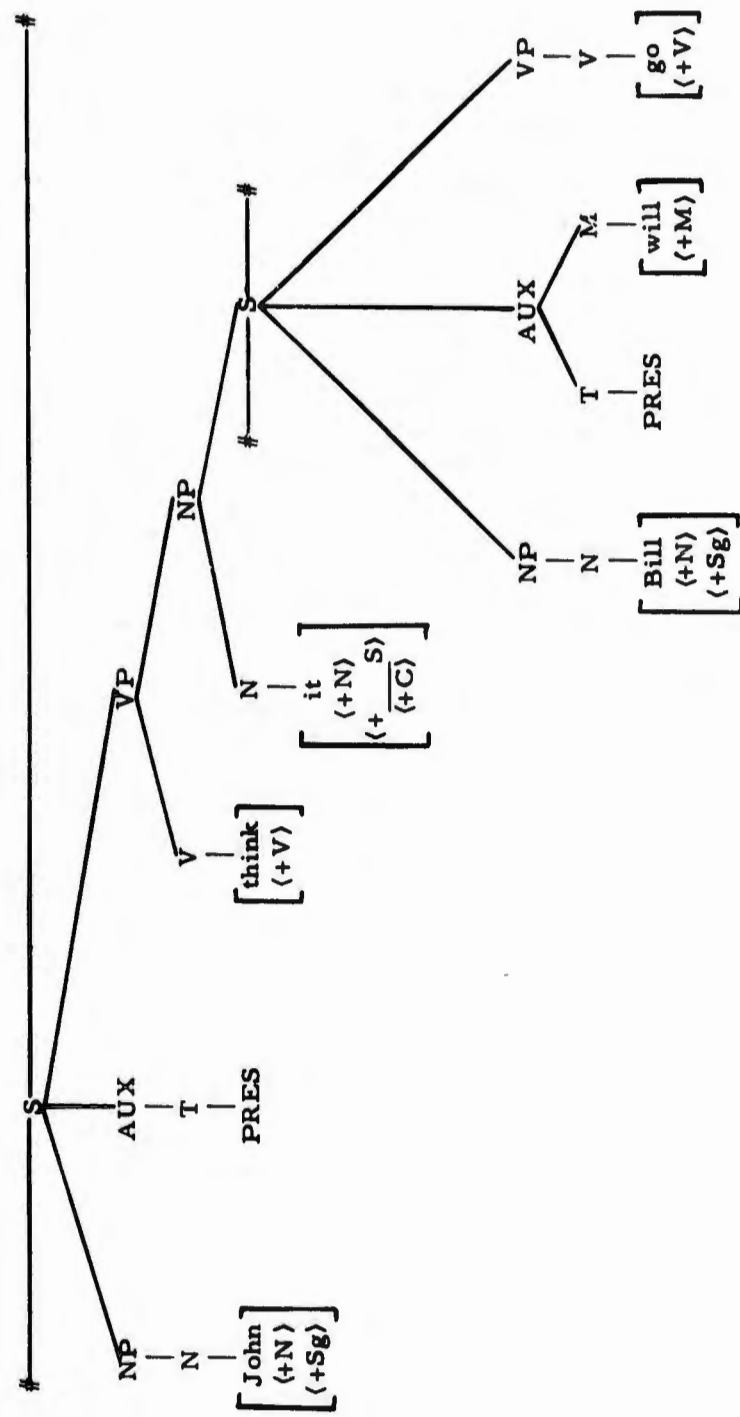
1. [[John]_N]_{NP} [decide [ed]_T]_V [on]PREP [$\begin{smallmatrix} go \\ (-C) \end{smallmatrix}$]_N]_S S T:PAST
2. [[John]_N]_{NP} [decide ed]_T]_V [on]PREP [$\begin{smallmatrix} go \\ ing \end{smallmatrix}$]_N]_S S T:C 4

Sentence Type No. 57

Actual string: John thinks that Bill will go

Derived string: John think s that Bill will go

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill]_N]_{NP} [[+Sg] PRES]_T [will_(+M)]_M]_N [go]_V]_S Y T:AG
2. X [[[[Bill]_N]_{NP} [will_(+M) + [[+Sg] PRES]_T]_M]_M]_N [go]_V]_S Y T:AF

Second Cycle:

1. [[[[John]_N]_{NP} PRES [think]_V [it]_N [that [[Bill]_N]_{NP} [will [+Sg] PRES]_M]_M]_N]_N]_{CP} 3
2. I [[[John]_N]_{NP} [[+Sg] PRES]_T [think]_V [it]_N [that [[Bill]_N]_{NP} [will [+Sg] PRES]_M]_M]_N]_N]_{CP} 3
3. [[[John]_N]_{NP} [think [[+Sg] PRES]_T]_V [it]_N [that [[[Bill]_N]_{NP} [will [+Sg] PRES]_M]_M]_N]_N]_{CP} 3
4. [[[John]_N]_{NP} [think [[+Sg] PRES]_T]_V [that [[[Bill]_N]_{NP} [will [+Sg] PRES]_M]_M]_N]_N]_{CP} 3

Post Cycle:

1. [[[John]_N]_{NP} [think [+Sg] PRES]_V [that [[Bill]_N]_{NP} [will]_M]_M]_N]_N]_{CP} 3
2. [[[John]_N]_{NP} [think [s]_T]_V [that [[Bill]_N]_{NP} [will]_M]_M]_N]_N]_{CP} 3

T:MTDEL

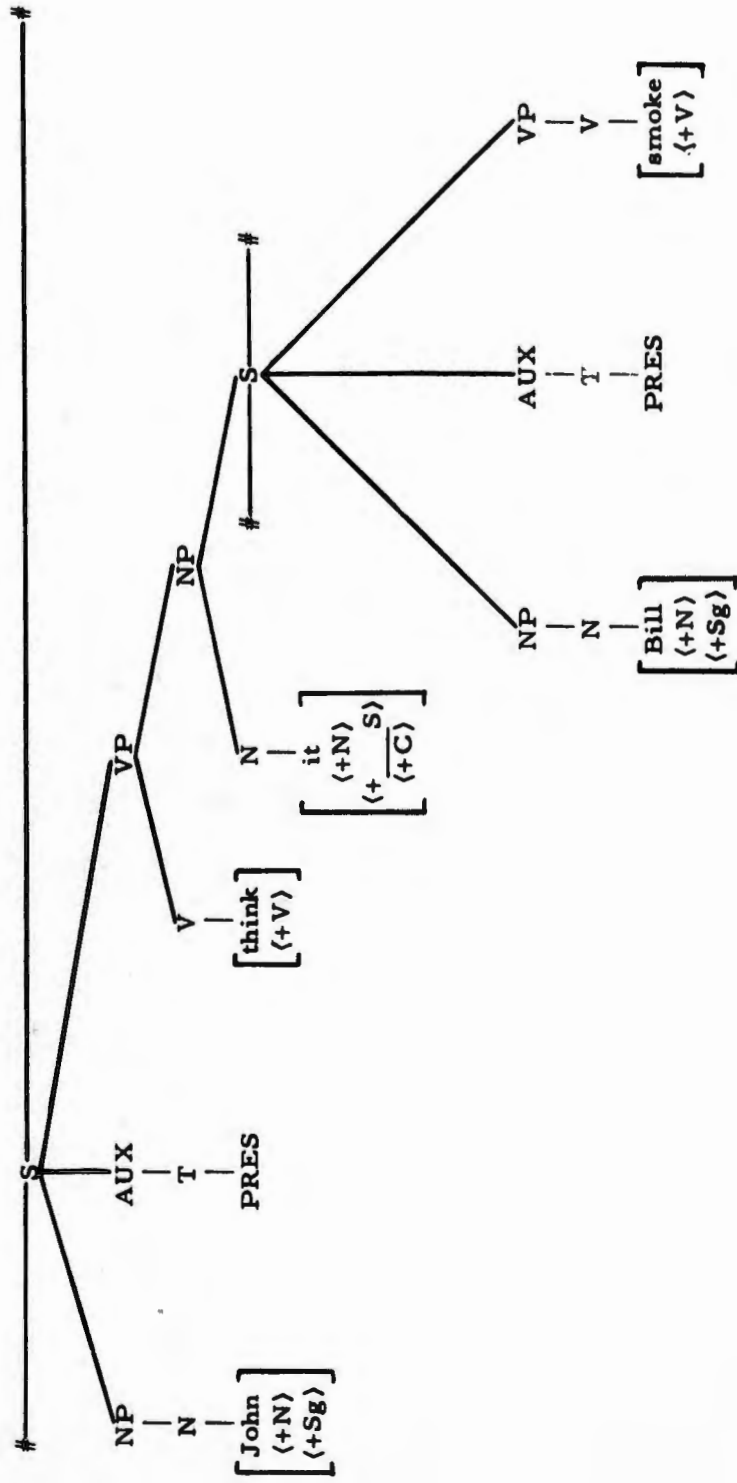
T:NUAG

Sentence Type No. 58

Actual string: John thinks Bill smokes

Derived string: John think s Bill smoke s

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill]_N]_{NP} [[+Sg]PRES]_T [smoke]_S]_V]_S Y
T:AG
2. X [[[Bill]_N]_{NP} [smoke [[+Sg]PRES]_T]_S]_V]_S Y
T:AF

Second Cycle:

1. [[[John]_N]_{NP} PRES [think]_V [it]_N [that [[Bill]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S]_{CP} 3
T:CP 3
2. [[[John]_N]_{NP} [[+Sg]PRES]_T [think]_V [it]_N [that [[Bill]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S]_{CP} 3
T:AG
3. [[[John]_N]_{NP} [think [[+Sg]PRES]_T]_V [it]_N [that [[Bill]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S]_{CP} 3
T:AF
4. [[[John]_N]_{NP} [think [[+Sg]PRES]_T]_V [that [[Bill]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S]_{CP} 3
T:PD
5. [[[John]_N]_{NP} [think [+Sg]PRES]_V [[Bill]_N [smoke [+Sg]PRES]_S]_S]_{CP} 3
T:THAT

Post Cycle:

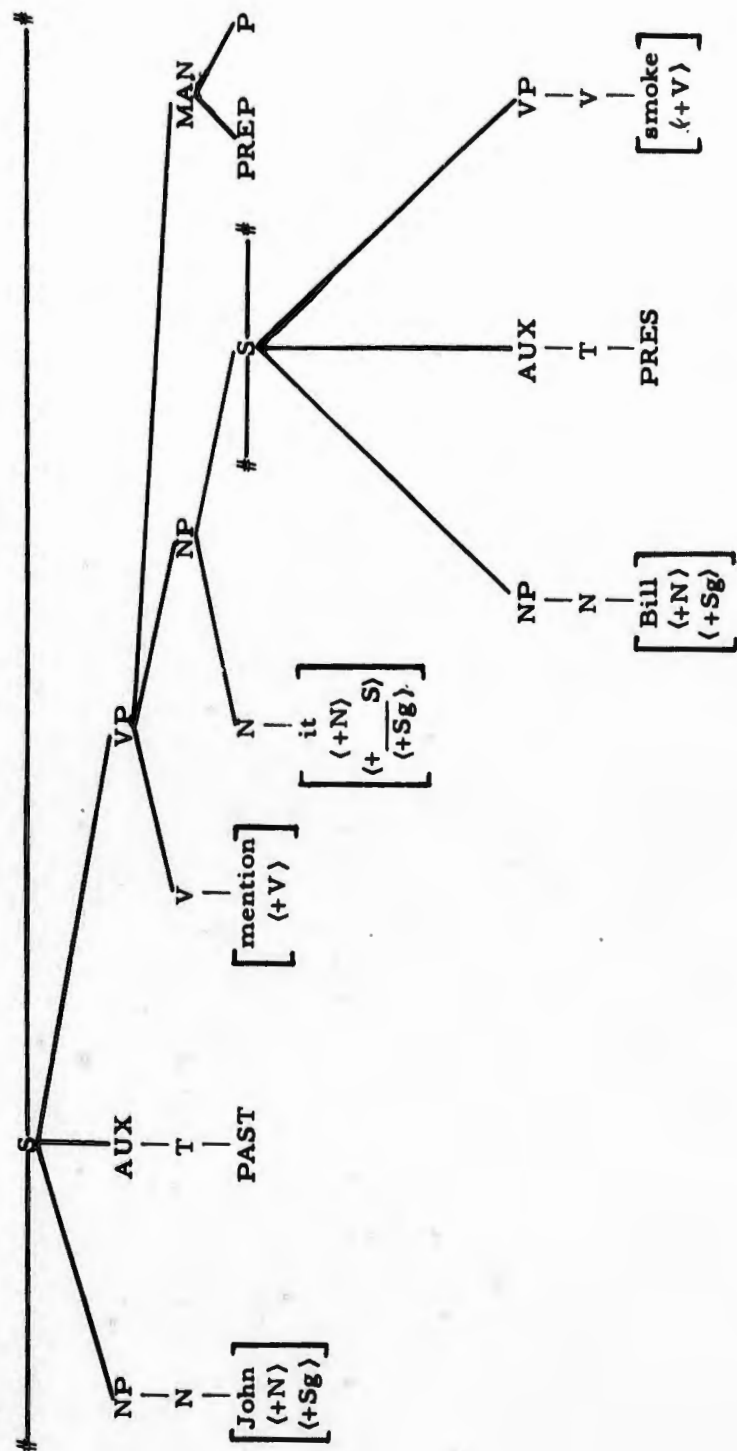
1. [[[John]_N]_{NP} [think[s]_T]_V [[Bill]_N [smoke[s]_T]_V]_S]_S]_{CP} 3
T:NUAG

Sentence Type No. 59

Actual string: that Bill smokes was mentioned by John

Derived string: that Bill smoke s was mention en by John

Underlying structure:



Derivation:

First Cycle:

1. X [[[Bill]_N]_{NP} [[+Sg] PRES]_T]_S Y
_(+V)_N _(+V)_V T:AG
2. X [[[Bill]_N NP]_{NP} [[+Sg] PRES]_T]_S Y
_(+V)_V T:AF

Second Cycle:

1. [[[John]_N NP]_{NP} [[PAST]_T]_T AUX [mention]_V [[it]_N [that[[Bill]_N NP]_{NP} [smoke[+Sg] PRES]_V S NP]_S PREP P]_S]_S
T:CP 3
2. [[[it]_N [that[Bill]_N [smoke[+Sg] PRES]_V S NP]_{NP} [[PAST]_T]_T AUX be + en + [mention]_V [PREP[[John]_N NP]_{NP} MAN]_S]_S
T:PASSIVE
3. [[[[it]
_(+Sg)_N] [that[Bill]_N [smoke[+Sg] PRES]_V S NP]_{NP} [[PAST]_T be]_{AUX} en [mention]_V [PREP[[John]_N NP]_{NP} MAN]_S]_S
T:AUXFILL
4. [[[[it]
_(+Sg)_N] [that[Bill]_N [smoke[+Sg] PRES]_V S NP]_{NP} [[+Sg] PAST]_T be]_{AUX} en [mention]_V [PREP[[John]_N NP]_{NP} MAN]_S]_S
T:AG
5. [[[[it]
_(+Sg)_N] [that[Bill]_N [smoke[+Sg] PRES]_V S NP]_{NP} [be + [[+Sg] PAST]_T]_T AUX [mention + en]_V [PREP[[John]_N NP]_{NP} MAN]_S]_S
_(+V)_N _(+V)_V T:AF
6. [[[that[Bill]_N [smoke[+Sg] PRES]_V]_S NP]_{NP} [be [[+Sg] PAST]_T]_T AUX [mention en]_V [PREP[[John]_N NP]_{NP} MAN]_S]_S
_(+V)_V T:PD

Post Cycle:

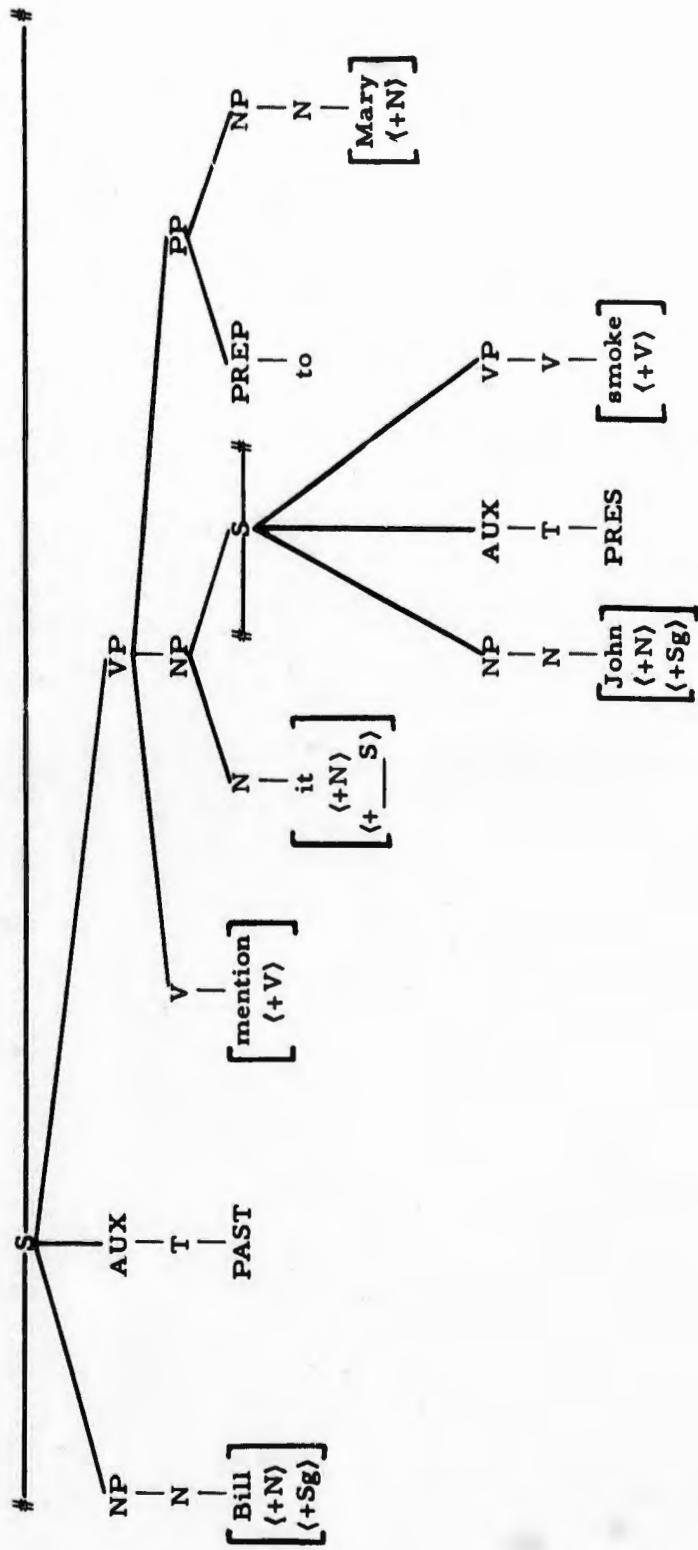
1. [[[that [Bill]_N [smoke [s]_T]_T]_S NP]_{NP} [be [+Sg] PAST]_T AUX [mention en]_V [PREP [John]_N NP]_{NP} MAN]_S]_S
T:NUAG
2. [[[that [Bill]_N [smoke s]_V]_S NP]_{NP} [was]_{AUX} [mention en]_V [PREP [[John]_N NP]_{NP} MAN]_S]_S
T:BE 3
3. [[[that [Bill]_N [smoke s]_V]_S NP]_{NP} [was]_{AUX} [mention en]_V [by [[John]_N NP]_{NP} MAN]_S]_S
T:BY

Sentence Type No. 60

Actual string: Bill mentioned to Mary that John smokes

Derived string: Bill mention ed to Mary that John smoke s

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_N]_{NP} [[+Sg]PRES]_T [smoke]_V]_S ^Y T:AG
2. X [[[John]_N]_{NP} [smoke [[+Sg]PRES]_T]_V]_S ^Y T:AF

Second Cycle:

1. [[Bill]_N PAST [mention]_V [[it]_N]_{CP} 3 [that [[John]_N]_{NP} [smoke[+Sg]PRES]_V]_S]_{NP} [to]PREP [Mary]_N]_S T:CP 3
2. [[[Bill]_N]_{NP} PAST [mention]_V [[it]_N]_{CP} 3 [that [John]_N]_{NP} [smoke[+Sg]PRES]_V]_S]_S T:EXTRA
3. [[[Bill]_N]_{NP} [[+Sg]PAST]_T [mention]_V [[it]_N]_{CP} 3 [that [John]_N]_{NP} [smoke[+Sg]PRES]_V]_S]_S T:AG
4. [[Bill]_N [mention[[+Sg]PAST]_T]_V [[it]_N]_{CP} 3 [that [John]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S T:AF
5. [[Bill]_N [mention [[+Sg]PAST]_T]_V [to]PREP [Mary]_N [that [John]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S T:PD

Post Cycle:

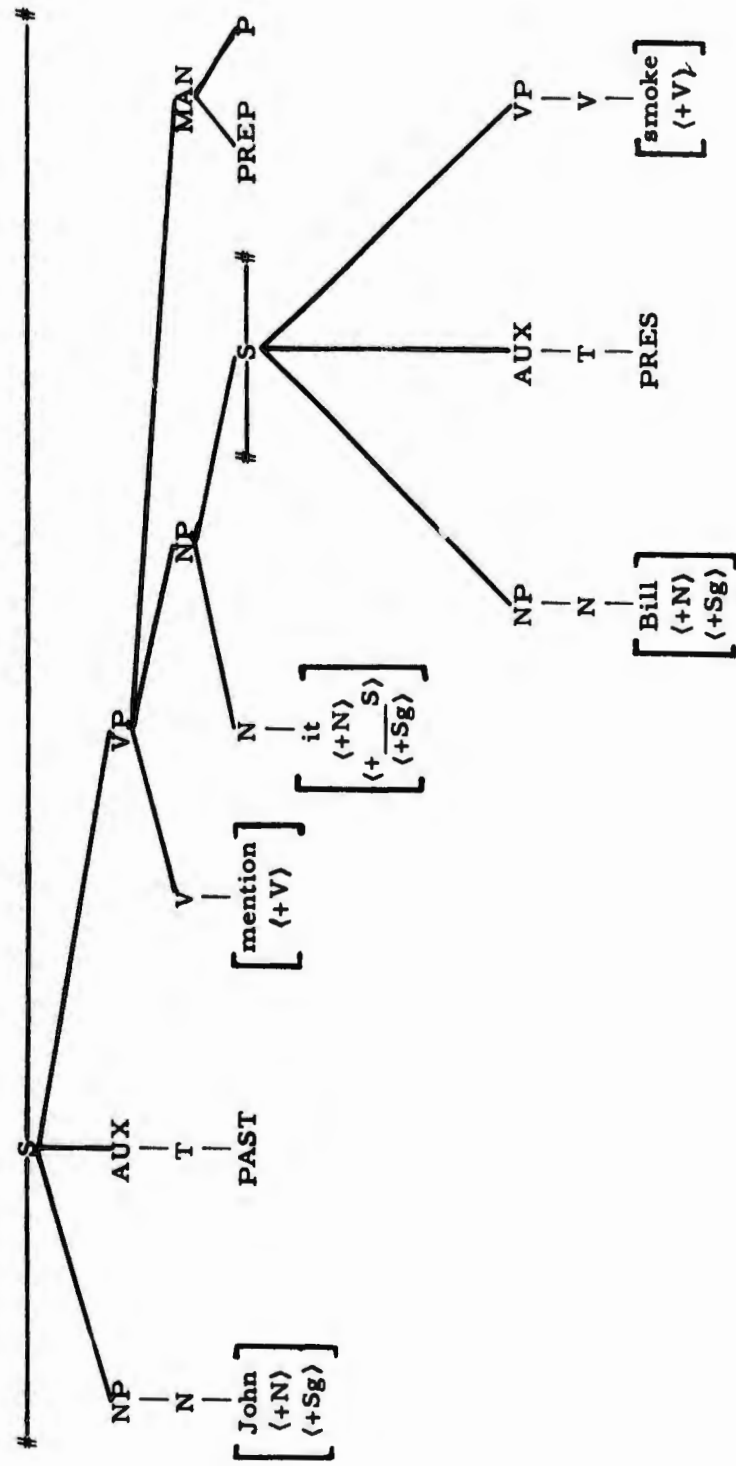
1. [[Bill]_N [mention [ed]_T]_V [to]PREP [Mary]_N [that [John]_N]_{NP} [smoke [+Sg]PRES]_V]_S]_S T:PAST
2. [[Bill]_N [mention ed]_V [to]PREP [Mary]_N [that [John]_N]_{NP} [smoke [s]_T]_V]_S]_S T:NUAG

Sentence Type No. 61

Actual string: it was mentioned by John that Bill smokes

Derived string: it was mention en by John that Bill smoke s

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\begin{array}{c} \text{Bill} \\ [+Sg] \end{array} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \left[\begin{array}{c} \text{smoke} \\ \langle +V \rangle \end{array} \right]_S \right]_Y$ T:AG
2. $X \left[\left[\left[\left[\begin{array}{c} \text{Bill} \end{array} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \right]_S \right]_{\langle +V \rangle} \right]_Y$ T:AF

Second Cycle:

1. $[[[John]_N]_{NP} [[PAST]_T]_{AUX} [mention]_V [[it]_N [that [Bill]_{NP} [smoke [+Sg] PRES]_V]_{S'}]_{NP} PREP P]_S]$
T:CP 3
2. $[[[[it]_{(+S)}]_N [that [Bill]_N [smoke [+Sg] PRES]_V]_{S'}]_{NP} [[PAST]_T]_{AUX} be + en + [mention]_V [PREP [John]_{NP} [NP]_{MAN}'S]_{S'}]$
T:PASSIVE
3. $[[[[it]_{(+S)}]_N]_{NP} [[PAST]_T]_{AUX} be en [mention]_V PREP [John]_N [that [Bill]_N [smoke [+Sg] PRES]_V]_{S'}]$
T:EXTRA
4. $[[[[it]_{(+Sg)}]_N]_{NP} [[PAST]_T]_{AUX} be [mention]_V PREP [John]_N [that [Bill]_N [smoke [+Sg] PRES]_V]_{S'}]$
T:AUXFILL
5. $[[[[it]_{(+Sg)}]_N]_{NP} [[[+Sg] PAST]_T]_{AUX} en [mention]_V PREP [John]_N [that [Bill]_N [smoke [+Sg] PRES]_V]_{S'}]$
T:AG
6. $[[[it]_N [be + [+Sg] PAST]_T]_{AUX} [mention + en]_{(+V)}]_{S'} PREP [John]_N [that [Bill]_N [smoke [+Sg] PRES]_V]_{S'}]$
T:AF

Post Cycle:

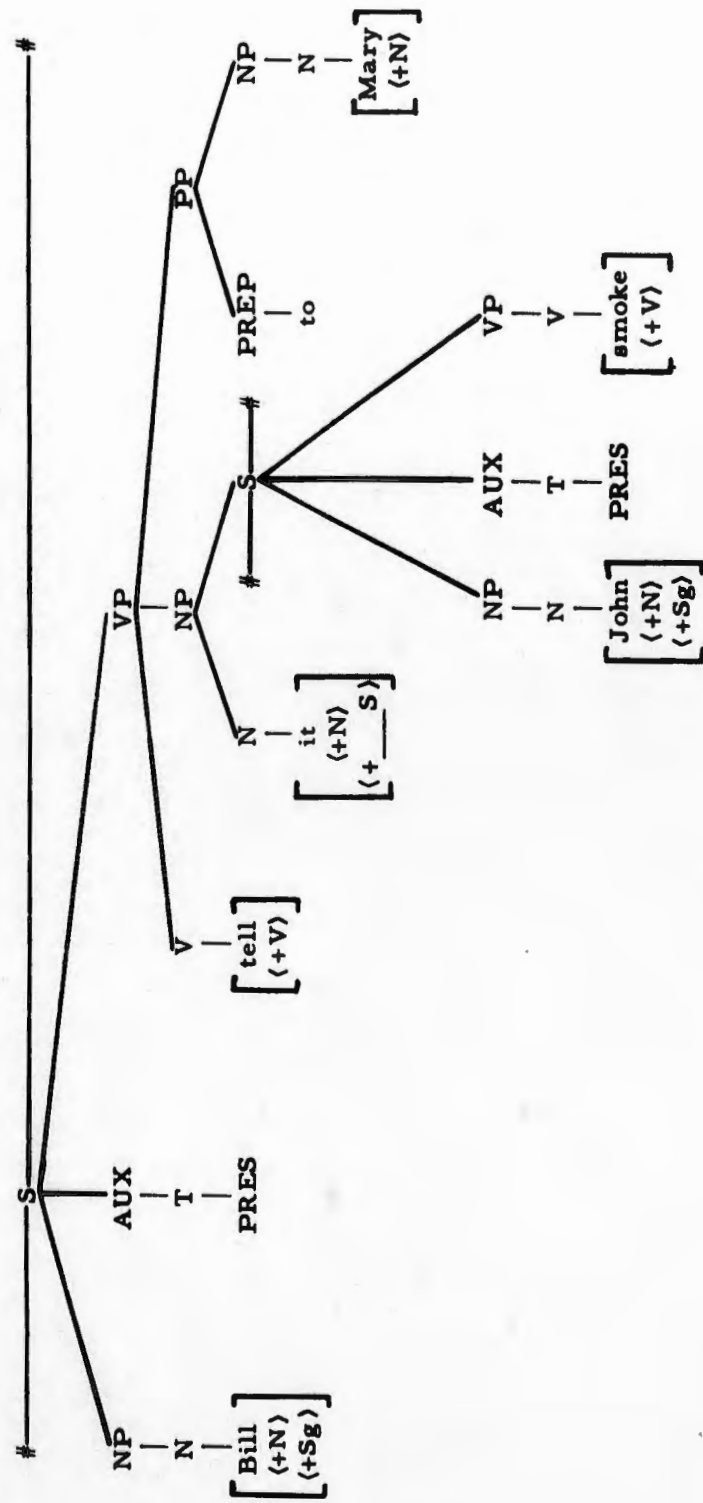
1. $[[it]_N [be [+Sg/PAST]_T]_{AUX} [[mention]_{(+V)}]_V [PREP [John]_N [that [Bill]_N [smoke [s]_T]_V]_S]_S$ T:NUAG
2. $[[it]_N [was]_{AUX} [[mention]_{(+V)}]_V [PREP [[John]_N]_{NP} MAN [that [Bill]_N [smoke s]_V]_S]_S$ T:BE 3
3. $[[it]_N [was]_{AUX} [[mention]_{(+V)}]_V [[by [[John]_N]_{NP} MAN [that [Bill]_N [smoke s]_V]_S]_S$ T:BY

Sentence Type No. 62

Actual string: Bill tells Mary John smokes

Derived string: Bill tell s Mary John smoke s

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_N]_{NP} [[+Sg]PRES]_T [smoke]_S]_S]_V T:AG
2. X [[[John]_N]_{NP} [smoke [[+Sg]PRES]_T]_S]_V T:AF

Second Cycle:

1. [[[Bill]_N PRES [tell]_V [[it]_N [that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_{PP}]_S]_{CP} T:CP 3
2. [[[Bill]_N PRES [tell]_V [[to]PREP [[Mary]_N]_{NP}]_{PP} [[it]_N [that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{IOI} T:IOI
3. [[[Bill]_N]_{NP} PRES [tell]_V [[[Mary]_N]_{NP}]_{PP} [[it]_N [that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{TO} T:TO
4. [[Bill]_N]_{NP} [[+Sg]PRES]_T [tell]_V [[[Mary]_N]_{NP}]_{PP} [[it]_N [that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{AG} T:AG
5. [[[Bill]_N [tell [[+Sg]PRES]_T]_V]_{NP}]_{PP} [[it]_N [that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{AF} T:AF
6. [[[Bill]_N [tell [[+Sg]PRES]_T]_V]_{NP}]_{PP} [[that[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{PD} T:PD
7. [[[Bill]_N [tell [[+Sg]PRES]_T]_V]_{NP}]_{PP} [[[John]_N [smoke[+Sg]PRES]_V]_S]_{NP}]_S]_{THAT} T:THAT

Post Cycle:

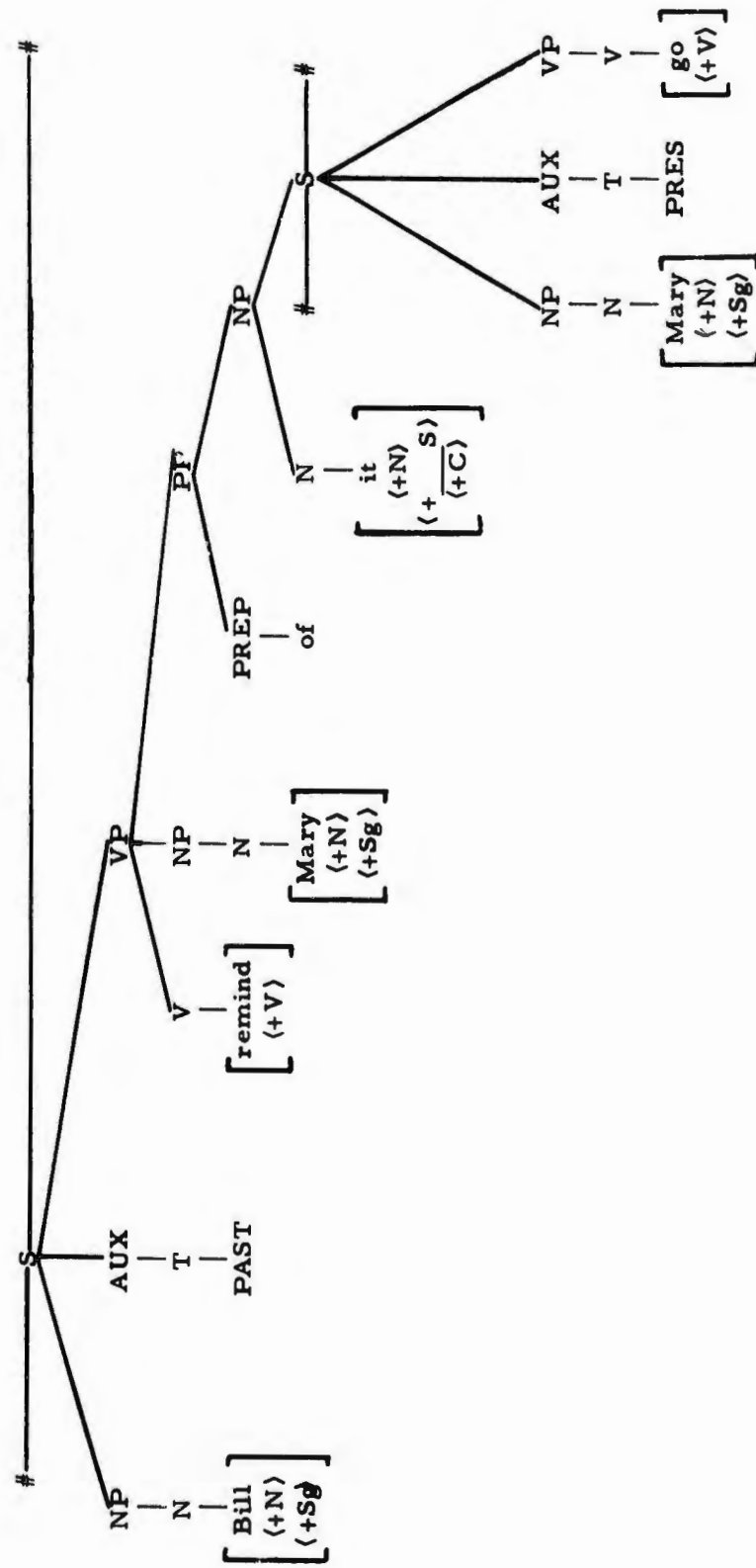
1. [[[Bill]_N [tell [s]_T]_V]_{NP}]_{PP} [[[Mary]_N]_{NP}]_{PP} [[[John]_N [smoke [s]_T]_V]_S]_{NP}]_S]_{NUAG} T:NUAG

Sentence Type No. 63

Actual string: Bill reminded Mary to go

Derived string: Bill remind ed Mary to go

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{+Sg} \right]_{PRES} \right]_T \left[\left[\text{go} \right]_{\langle +V \rangle} \right]_V \right]_S \right]_Y \right]$ T:AG
2. $X \left[\left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +V \rangle} \right]_{PRES} \right]_T \right]_S \right]_Y$ T:AF

Second Cycle:

1. $\left[\left[\text{Bill} \right]_N \text{PAST} \left[\text{remind} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +Sg \rangle} \right]_{PRES} \right]_V \right]_S \right]_S \right]_{CP1}$ T:IE
2. $\left[\left[\left[\text{Bill} \right]_{\langle +Sg \rangle} \right]_N \text{PAST} \left[\text{remind} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +Sg \rangle} \right]_{PRES} \right]_V \right]_S \right]_S \right]_{TS}$ T:IE
3. $\left[\left[\left[\text{Bill} \right]_{\langle +Sg \rangle} \right]_N \left[\left[\text{+Sg} \right]_{PAST} \left[\text{remind} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +Sg \rangle} \right]_{PRES} \right]_T \right]_S \right]_S \right]_{TS}$ T:AG
4. $\left[\left[\text{Bill} \right]_N \left[\left[\left[\text{+Sg} \right]_{PAST} \right]_T \left[\left[\text{remind} \right]_{\langle +V \rangle} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +V \rangle} \right]_S \right]_S \right]_S \right]_{TS}$ T:TS
5. $\left[\left[\text{Bill} \right]_N \left[\left[\left[\text{remind} \right]_{\langle +V \rangle} \right]_{\left[\left[\text{+Sg} \right]_{PAST} \right]_T} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +V \rangle} \right]_S \right]_S \right]_S \right]_{AF}$ T:AF
6. $\left[\left[\text{Bill} \right]_N \left[\left[\left[\text{remind} \right]_{\langle +V \rangle} \right]_{\left[\left[\text{+Sg} \right]_{PAST} \right]_T} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\left[\text{of} \right]_{PREP} \left[\left[\left[\text{it} \right]_{\langle +C \rangle} \right]_N \right] \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{go} \right]_{\langle +Sg \rangle} \right]_{PRES} \right]_V \right]_S \right]_S \right]_{PREPDEL}$ T:PREPDEL
7. $\left[\left[\text{Bill} \right]_N \left[\left[\left[\text{remind} \right]_{\langle +V \rangle} \right]_{\left[\left[\text{+Sg} \right]_{PAST} \right]_T} \right]_V \left[\left[\text{Mary} \right]_{NP} \left[\left[\left[\text{+C} \right]_{\left[\text{go} \right]_{\langle +V \rangle} \right]_S} \right]_S \right]_{PD}$ T:PD

Post Cycle:

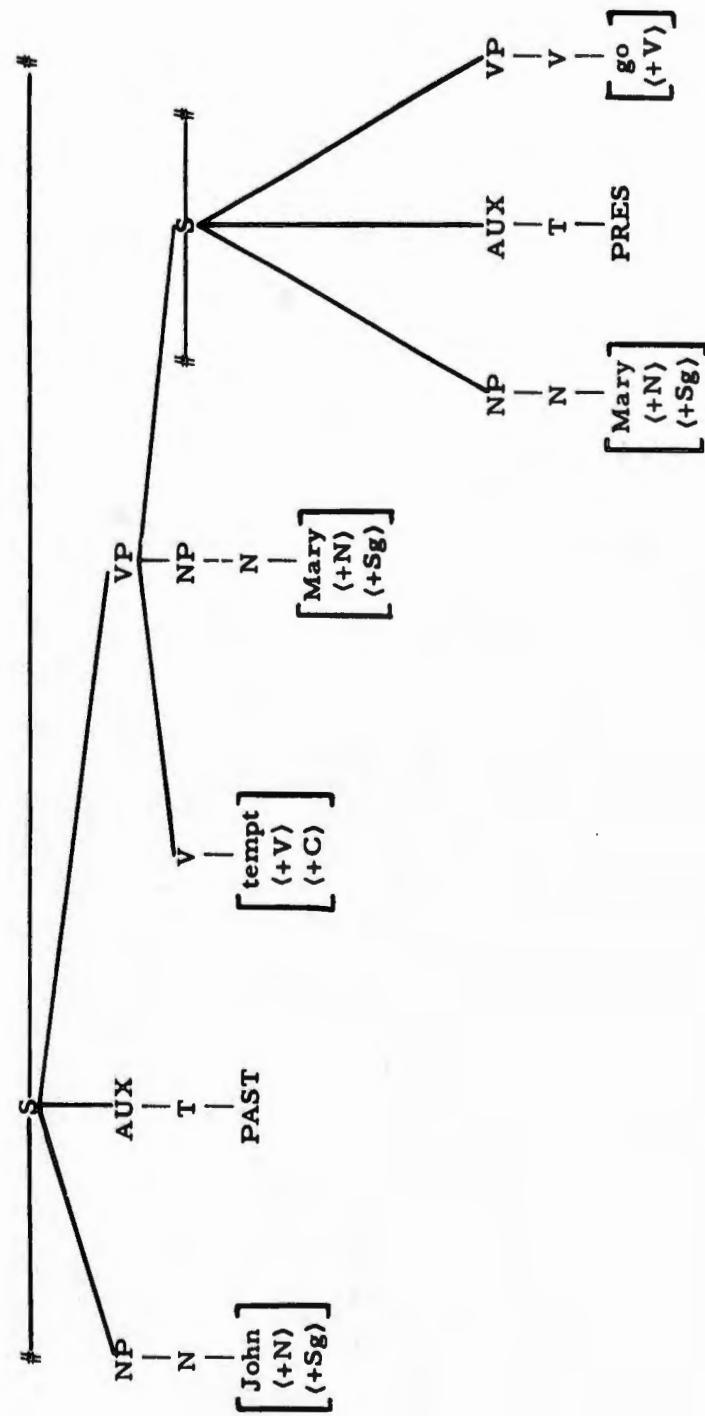
1. $\left[\left[\text{Bill} \right]_N \left[\text{remind} \right]_{\left[\left[\text{ed} \right]_T \right]_V} \left[\text{Mary} \right]_N \left[\left[\text{+C} \right]_{\left[\text{go} \right]_{\langle +V \rangle} \right]_S \right]_S$ T:PAST
2. $\left[\left[\text{Bill} \right]_N \left[\text{remind ed} \right]_V \left[\text{Mary} \right]_N \left[\left[\text{to} \right]_{\left[\text{go} \right]_{\langle +V \rangle} \right]_S \right]_S$ T:C2

Sentence Type No. 64

Actual string: John tempted Mary to go

Derived string: John tempt ed Mary to go

Underlying string:



Derivation:

First Cycle:

1. X [[[Mary]_N]_{NP} [[+Sg]PRES]_T [go_(+V)]_V]_S Y T:AG
2. X [[[Mary]_N]_{NP} [go_(+V)]_V]_S Y T:AF

Second Cycle:

1. [[[John]_N]_{NP} PAST [tempt_(+C)]_V]_S Y T:CP 2
2. [[[[John]_N]_{NP} PAST [tempt_(+C)]_V]_S Y T:IE
3. [[[John]_N]_{NP} [+Sg]PAST [tempt_(+C)]_V]_S Y T:AG
4. [[[John]_N]_{NP} [[+Sg]PAST]_T [tempt_(+C)]_V]_S Y T:TS
5. [[[John]_N]_{NP} [tempt_(+V)]_V]_S Y T:AF

Post Cycle:

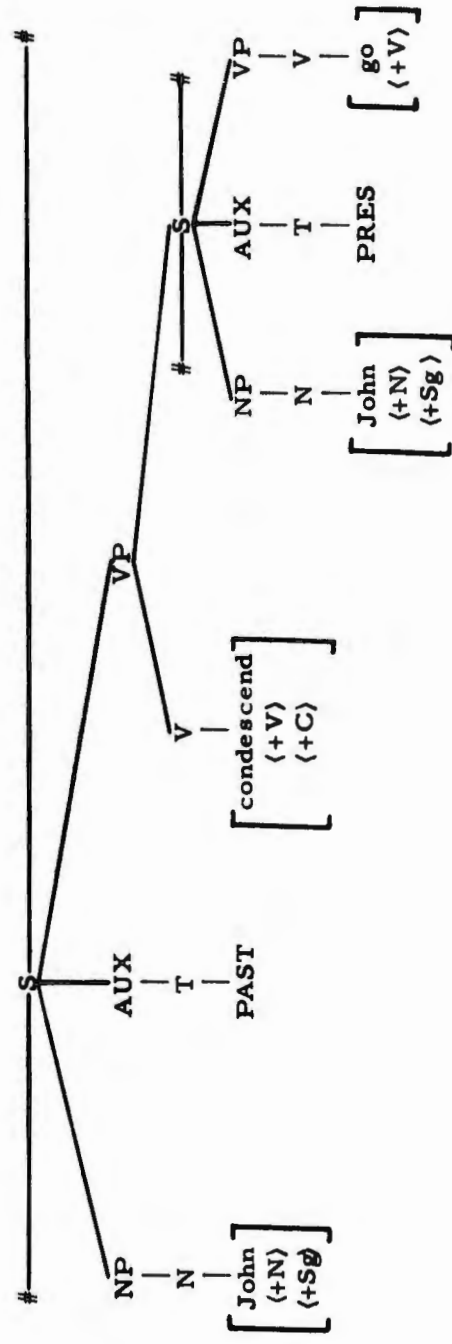
1. [[[John]_N]_{NP} [tempt ed]_T]_S Y T:PAST
2. [[[John]_N]_{NP} [tempt ed]_V]_S Y T:C 2

Sentence Type No. 65

Actual string: John condescended to go

Derived string: John condescend ed to go

Underlying structure:



Derivation:

First Cycle:

1. X [[[John]_{NP} [[[+Sg] PRES]_T]_S]_Y]_V
2. X [[[John]_N]_{NP} [go [[+Sg] PRES]_T]_S]_V

T:AG

T:AF

Second Cycle:

1. $[[[John]_N]_{NP} PAST \left[\begin{array}{c} \text{condescend} \\ \langle +C \rangle \end{array} \right]_V [+C [[John]_N]_{NP} [[go [+Sg] PRES]_V]_{VP} S]_S$ T:CP 2
2. $[[[John]_{\langle +Sg \rangle}]_N]_{NP} PAST \left[\begin{array}{c} \text{condescend} \\ \langle +Sg \rangle \end{array} \right]_V [+C [go [+Sg] PRES]_V]_S$ T:IE
3. $[[[John]_{\langle +Sg \rangle}]_N]_{NP} [+Sg] PAST \left[\begin{array}{c} \text{condescend} \\ \langle +Sg \rangle \end{array} \right]_V [+C [go [[+Sg] PRES]_T]_S]_S$ T:AG
4. $[[John]_N]_{NP} [+Sg] PAST \left[\begin{array}{c} \text{condescend} \\ \langle +V \rangle \end{array} \right]_V [+C [go]_V]_S$ T:TS
5. $[[John]_N]_{NP} \left[\begin{array}{c} \text{condescend} \\ \langle +V \rangle \end{array} \right]_V [+C [go]_V]_S$ T:AF

Post Cycle:

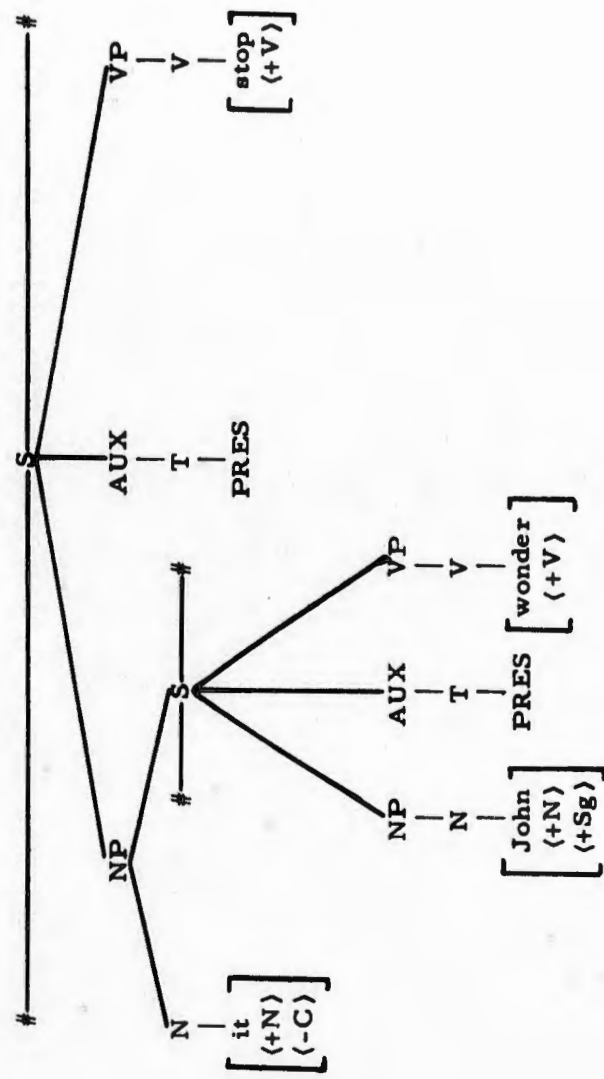
1. $[[John]_N]_{NP} \left[\begin{array}{c} \text{condescend} \\ \langle +V \rangle \end{array} \right]_V [+C [go]_V]_S$ T:PAST
2. $[[John]_N]_{NP} \left[\begin{array}{c} \text{condescend} \\ \langle +V \rangle \end{array} \right]_V [+C [go]_V]_S$ T:C 2

Sentence Type No. 66

Actual string: John stops wondering

Derived string: John stop s wonder ing

Underlying structure:



Derivation:

First Cycle:

1. $X \left[\left[\left[\text{John} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \left[\text{wonder} \right]_{VP} \right]_{S} \right]_Y$
2. $X \left[\left[\left[\left[\text{John} \right]_{NP} \left[\text{wonder} \right]_{NP} \left[[+Sg]_{PRES} \right]_T \right]_{S} \right]_{VP} \right]_Y$

T:AG

T:AF

T:CP 1

1. [[[it
(-C)
(+S)] N
[-C [[John]_NNP [wonder [[+Sg]PRES_TV_SNP PRES [stop]_V]S
2. [[[it
(+S)] N
[_NP [[PRES]_TAUX [stop]_V [-C [[John]_NNP [wonder [[+Sg]PRES_TV_S]S
3. [[[John
(+Sg)] N
PRES [stop]_V [-C [wonder [[+Sg]PRES_TV_S]S
4. [[[John
(+Sg)] N
[_NP [[+Sg]PRES_T [stop
(+V)] V
[-C wonder [[+Sg]PRES_T l_S]S
V
5. [[[John
(+Sg)] N
[_NP [[+Sg]PRES_T [stop
(+V)] V
[-C [wonder
(+V)] V
l_S]S
6. [[[John]_NNP [stop [[+Sg]PRES_T] { [wonder -C] l_S]S
(+V) } V (+V)] V

Post Cycle:

1. $[[[\text{John}]]_{N'}]_{NP} [\text{stop}[s]]_{TV} [[\text{wonder} - \text{Cl}]_V]_{SS}$
2. $[[[\text{John}]]_{N'}]_{NP} [\text{stop } s]_V [[\text{wonder ing}]_V]_{SS}$

T:NUAG

T:C 4

5.0 Notes

¹In particular, see Noam Chomsky, Aspects of the Theory of Syntax, (Cambridge, 1965).

²Cf. G. Lakoff, On the Nature of Syntactic Irregularity, Indiana University Doctoral Dissertation, (1966).

³Cf. Chomsky, Aspects . . . , Ch. 2.

⁴Cf. G. Lakoff, On the Nature

⁵Cf. P. Postal, 'A Note on 'Understood Transitively', " IJAL, January, 1966, Vol. 32, No. 1, for discussion of the identity condition in the transformational component.

⁶The rules for relative clause formation employed in the CG are adapted from the more general formulation of S. Y. Kuroda.

⁷The obligatoriness of the rule CP 1 is lexically determined.

⁸For discussion of the condition imposed upon the Identity Erasure Transformation, Cf. P. S. Rosenbaum, 'A Principle Governing Deletion in English Sentential Complementation, " IBM Research Report, RC 1519, (Yorktown Heights, 1965). For a general discussion of sentential complementation in English, Cf. P. S. Rosenbaum, Grammar of English Predicate Complement Constructions, MIT Doctoral Dissertation, 1965, and "Phrase Structure Principles of Complex Sentence Formation in English, " (in preparation), and "Transformational Principles of Complex Sentence Formation in English, " (in preparation).

⁹For a transformation which must apply if the P-marker meets the structural conditions imposed by the structure index of the rule, e. g., the Identity Erasure Transformation for Verb Phrase Complement constructions, the failure of this P-marker to meet the identity condition imposed on this transformation must indicate an ungrammatical derivation. Such a "blocking" facility is not currently provided in the CG and the grammar, consequently, will generate ungrammatical strings. This facility is omitted at present primarily because it is not clear whether derivations should actually "block," (which is a simple enough matter to accomplish, Cf. English preprocessor manual, Information System Language Studies Number Seven, Mitre Corp., May, 1965,) or whether derivations should more appropriately proceed to termination becoming marked in the process either for legality or for illegality. The latter mechanism currently appears promising and will, in all probability, be incorporated into a later version of the IBM English Grammar.

BLANK PAGE

PART II

"DESIGN OF A GRAMMAR TESTER"

D. Lieberman

**This Part has been submitted for presentation at the Fall
Joint Computer Conference, November 8-10, 1966,
San Francisco, California**

Design of a Grammar Tester

Contents

	<u>Page</u>
<u>Design Considerations</u>	1
1. Automatic expansion of a grammar	4
2. Checking the legality of a derivation	4
3. Traces	5
4. Automatic comparison with expected result	6
5. Partially automatic update of a set of test sentences	6
6. Testing a sequence of variations of a grammar in one run	9
7. Grammar and test sentence maintenance	10
<u>Input and Output Formats and Control Cards</u>	13
1. Form of the grammar	13
2. Conversion of compact form to expanded form	21
3. Grammar update	24
4. Production of specified derivation and comparison with expected result	25
<u>Present Status</u>	31

The work described herein is the first step in a long-range program to develop computer aids to linguistic research and machine techniques to exploit linguistic theory in practical applications.

Our immediate goal is a computer aid in the development of a comprehensive and detailed transformational grammar of English. This computer aid is intended to provide material assistance to the grammarian and, at the same time, to be a vehicle for the accumulation of initial experience in linguist-computer interaction under conditions of half-day turn-around time. Also, some initial information regarding the programmer and machine resources (including running times) required for this type of work will be obtained and will be used in planning future work.

Design Considerations

The two main design considerations are: 1) what facilities are necessary and/or useful to the linguist, and 2) in what form will these facilities be provided? In other words, what will the machine do for the linguist, and what must the linguist do for the machine?

There are many things, ranging from simple book-keeping to random generation of sentences, that can be done by a computer operating on a grammar in machinable form. In fact, there have been suggestions in the literature that computers could perform what are usually considered to be creative aspects of grammatical research. For example, it has been suggested that a computer could be programmed to accept some corpus as input and produce a grammar of the

corpus as output, or could accept as input some corpus and its translation into some other language, and produce a translation algorithm as output. Whatever the merit of such suggestions may be (especially without the support of a theory of linguistic universals), our approach is quite different and far less spectacular as regards its immediate objectives. Rather than attempt to revolutionize the manner in which linguistic research is carried out, we accepted current methods as being basically sound and asked whether there were aspects of the linguist's work which could be carried out more effectively (economically) or more extensively by using a computer. Although we did not rule out the possibility that certain activities previously considered creative might turn out to be routine and machinable, we looked first for obviously routine and tedious tasks which could be done as well or better by a computer than by a linguist.

The task we chose for immediate implementation was the testing of a grammar. Specifically, given a grammar, will it in fact generate the variety and complexity of sentences and accompanying structural descriptions expected by the grammarian? Moreover, will it correctly exclude all that it is supposed to exclude? Exhaustive testing is, of course, out of the question even with a computer. What we are interested in is establishing the extent and type of testing required to "check out" a grammar, and developing computer techniques to facilitate the work.

The problem of grammar testing, or, more generally, the characterization of the coverage of a grammar, has received very little attention in the literature in the past, since

it has not been a pressing problem. Transformational theory has only recently reached a stage of development at which it is feasible to use it as a basis for a comprehensive and detailed grammar of English. Until recently, applications of transformational theory to descriptions of English have had as their main objective the testing of the adequacy of the general theory. The grammars actually produced were either detailed descriptions of fragments of English, or were skeletal descriptions of the main features of English syntax. Moreover, these grammars served their purpose without requiring an unreasonable amount of testing effort (either they were small enough to be tested exhaustively manually, or they were inadequate in some way at a gross level and there was no point in examining them in greater detail).

The situation is now quite different. Using current transformational theory, we are able to write grammars which are too big for exhaustive manual testing and at the same time appear to be adequate at a very detailed level. The grammar of English presented in Part I of this report is just about at the limit of manual tractability, but it is far from limitations imposed by the current state of the theory. Thus, the use of machine testing procedures is not just highly desirable, but appears to be a necessity for further expansion of the grammar.

The basic test facility is the automatic generation of a sentence. The input is a grammar and a specified derivation (sequence of rule numbers), and the output is a sentence and its structural description or a failure indication if the derivation could not be carried out. But if this were all that was

provided for the linguist, it would probably be more of a hindrance than a help. Eventually, experience will indicate what sort of computer aids are useful and feasible, but even at the present design stage it is clear that a number of additional facilities are required:

1. Automatic expansion of a grammar. In specifying a derivation, it is not enough to specify rule numbers--specific subrule numbers must be specified. Thus, a facility is needed to take as input a grammar in the usual notation (using braces and parentheses) and produce as output a listing of the grammar in which each subrule is explicitly displayed and uniquely identified in some way. If this task were left to the linguist and had to be repeated each time the grammar was modified, it would detract significantly from the over-all usefulness of the computer aid.

2. Checking the legality of a derivation. The fact that some specified sequence of rules led to a successful derivation (an acceptable sentence and structural description) does not guarantee that the grammar can actually generate the sentence which resulted from the application of the specified sequence of rules. The specified sequence may have violated some traffic rule conventions in the grammar. For example, a particular context-free subrule may have been specified, but a context-sensitive subrule in the same main rule may have been applicable at the time the specified rule was applied. This would be a violation of the convention that, within each main rule, the subrules form two main groups--one context-sensitive and one context-free--and the context-sensitive group is considered to precede the context-free group in order.

Similar violations can occur in the application of transformational rules. It is therefore necessary to include a facility for checking the legality of a specified derivation as it is carried out. Moreover, there should be various options available to the linguist regarding the action to be taken if a specified derivation is illegal at some point. As a minimum, there should be two options--one to halt and one to proceed--with appropriate comments regarding the illegality printed out in either case. The reason for including a proceed option is that it may, at times, be of experimental value to be able to determine the result of applying a particular sequence of rules even though the grammar in which they occur contains other rules which would take precedence in a rigorous application of the grammar.

3. Traces. Thus far, the only computer output we have mentioned (aside from printout of comments regarding illegalities in specified derivations) is the final derived sentence and its surface structure. All of the intermediate steps (the current string and tree after each rule application) are readily available, but, with the exception of the step at which the deep structure is established, the status after each rule application would be of interest only in special cases. It therefore seems best to leave the extent of tracing as an option to be selected for each derivation by the linguist. The choices are:

- a. no trace,
- b. complete trace--the current string and current tree are printed out after each rule application,
- c. as called for--the current string or the current tree or both are printed out after the application of appropriately marked items in the specified

derivation.

4. Automatic comparison with expected result. It can be expected that during the early stages of testing of a grammar most sentences will reveal some deficiency in the grammar, and relatively few sentences will be generated correctly. The trace facilities would be used heavily, and the proportion of time spent in checking the final result for correctly generated sentences would be small. However, as the grammar is corrected and refined, a larger and larger number of sentences will be correctly generated. But every time the grammar is modified, even in a minor way, it will be necessary to re-check a large number of sentences supposedly unaffected by the modification. At this point, the task of manually checking just the final results of the derivations could become a major bottleneck in the use of the tester as a tool in the development of a grammar. Thus, a facility for automatically checking the final results (string and surface tree, and perhaps deep structure as well) appears to be highly desirable. This facility should, of course, be optional. The input to the tester when using the automatic check option would then be a grammar, a specified derivation and a final string, a final tree, or both (and appropriate control cards, as described below, to indicate the selected option). The output would be simply a YES or NO for each of the specified structures. Additional output could be obtained simultaneously by selecting an appropriate TRACE option.

5. Partially automatic update of a set of test sentences. As the testing of a grammar proceeds, a collection of sentences will be accumulated which could form a test set in the

sense that derivations of the entire set would involve a very large portion of the grammar. Such a set would probably be of the order of about 100 sentences. A final string and tree would be associated with each of the test sentences so that the automatic checking option could be used. The grammarian could simply scan the output for a NO result in order to find the remaining few errors in the grammar. This would be a very effective procedure when the grammar is developed to a point at which 90% or more of the test sentences are correctly generated (or correctly not generated--the test set would probably contain not only legal derivations, but also specified sequences of rules to verify that certain non-grammatical strings are in fact suppressed by the grammar). However, there is one drawback to the above procedure. Whenever the grammar is modified, even slightly, in such a way as to directly affect only a few test sentences, a much larger number of test sentences may be indirectly affected. By affected, I mean that the specified sequence of subrule numbers which represents the derivation of a particular sentence will require modification. For example, suppose a new symbol were added to the right hand side of constituent structure rule 1 as an optional term, and that the selection of that item was required in only a few of the test sentences. Moreover, suppose that the new symbol were rewritten as a string of old symbols in the next rule (of the new grammar) and that no further changes in the content of the rules were necessary. Then the new grammar would be related to the old as follows:

Rule 1 would have one or more new subrules.

Rule 2 would be entirely new.

Rule 3 would have the same content as Rule 2 in the old grammar, and in general rule i in the new grammar would have the same content as Rule $i-1$ in the old grammar for $i \geq 3$.

This means that just about every single rule number in every specified derivation representing the set of test sentences would have to be changed. If this had to be done manually, the advantages of having a set of test sentences would be largely nullified. Fortunately, most of the modification required in this example can be done automatically by keeping track of added (and, in general, also deleted) rules during grammar update and establishing a mapping from old rule number to new rule number. In this example, the mapping would be:

$(\text{Rule } i)_{\text{old}} \rightarrow (\text{Rule } i+1)_{\text{new}}$ for $i \geq 2$. The mapping could then be applied to the set of specified derivations representing the set of test sentences to modify all occurrences of relevant rule numbers ($i \geq 2$) appropriately.

In the above example, we considered the automatic establishment of a mapping involving only subrules within main rules whose content was not affected by the grammar modification. The mapping can be carried further to include those subrules within a modified main rule whose contents are not affected by the modification. This can be done by expanding the modified main rule, comparing the resulting collection of subrules with the original collection of subrules, and establishing a mapping between subrules with identical content before and after the modification. The mapping applied to the set of test sentences would then be a resultant mapping obtained by combining the main rule mapping and the subrule mapping.

It should be mentioned again that the updating of the set of test sentences cannot be fully automatic. However, a large amount of updating can be done automatically and it is essential that provision for such facilities be included in the design of the grammar tester.

6. Testing a sequence of variations of a grammar in one run. Thus far, we have discussed facilities for testing one grammar at a time. The procedure would be somewhat as follows:

- carry out a testing run;
- examine the results and prepare modifications;
- carry out an update run on the grammar and test sentences;
- complete the updating of the test sentence, specified derivations, and expected results manually where necessary;
- carry out a new testing run;
- etc.

This procedure will be effective only if, after each testing run, it is of interest to apply only one set of modifications to the grammar. But if it is desirable to try several alternatives, the above procedure would be too cumbersome since it would require either a long series of runs or else the maintenance of several complete copies of the program, grammar, set of test sentences, and any other sentences being studied.

A limited amount of simultaneous testing of several variants of a grammar can be achieved by using the update facility described in (5) above, and making provision for a sequence of tests and updates in one run. It should be possible to apply the various updates to either the original grammar

(the input grammar at the start of the run) or to any of its updated versions (formed earlier in the run). This can be done by labeling each version of the grammar as it is formed and including with each update the name of the grammar to which it is to be applied. From the programming point of view, this requires the construction of a primitive supervisory facility.

The extent to which this facility for experimenting with several variants of a grammar simultaneously can be used is limited because, as noted above, updating of test derivations cannot, in general, be fully automatic. However, it will be possible to carry out some updates completely automatically, and even when the automatic part of the update is not complete, it may be sufficient to permit useful experimentation to be carried out. In particular, specified derivations which are supposedly not affected (except for renumbering) by a given modification can be tested even if the updating of other derivations must be completed later manually. Since it will often be easier to foresee the effect of a modification on a small problem area than on the remainder of the grammar, the testing of supposedly unaffected portions of the grammar will provide a very useful preliminary evaluation of the correctness of a modification. Thus, it appears to be quite worthwhile to include planning for this type of facility in the initial design consideration.

7. Grammar and test sentence maintenance. The grammar tester will permit almost continuous experimentation with and modification of the grammar. In fact, it may be possible to develop a grammar along several different lines, or, in effect, to develop several grammars simultaneously. A

vast amount of material will be produced as the grammar development proceeds, most of which will be of only transitory interest. The relatively small portion selected for temporary or more or less permanent retention will still be voluminous enough to require machine aids in its management. The number of simultaneously active grammars or variants should be limited only by the grammarian's ability to handle them effectively from a linguistic point of view, and should not be limited by storage and retrieval problems associated with the use of the computer.

Every grammar can originate as an update. Even if an entirely new grammar is being loaded from cards, the system can be initialized with a null grammar and the grammar being loaded can be treated as an update of the null grammar. The same holds for the set of test sentences corresponding to the grammar. The control parameters of the update run (or update portion of a run) will be used to assign a label to the grammar and to indicate whether the grammar is to be saved. The set of test sentences will be handled similarly. The control parameters of the test portion of the run will be used to indicate whether the results of the test are to be saved on tape or just printed out. The reason for saving the results is that it may turn out, in the future, to be useful to scan the tapes by machine to find grammars with particular properties with respect to certain test sentences. After the automatic update, that portion of the updating of the set of test sentences which could not be done automatically will be completed manually and the tape will be corrected by a special program. Provision should also be made for allowing notes and comments to be

saved along with the grammar. Thus, maintenance programs are required to:

- select a specified grammar from the output of an update and/or test run;
- select the associated set of test sentences;
- add manual corrections to the updated test sentences;
- select the results of a test of the grammar;
- form a unit consisting of the above items and any comments provided by the linguist.

The above discussion was concerned with the first of the main design considerations mentioned at the beginning of this section of the report, namely, what facilities are necessary and/or useful to the linguist. Now let us consider the second--in what form will the facilities be provided to the linguist? The alternatives range from a high-level linguistics-oriented programming language which would be used by the linguist to program the facilities he desires to a single program package with all facilities built in and not modifiable by the user. We chose the latter extreme alternative, because when the various factors affecting the choice, such as programmer's time, linguist's time, ease of modification of the program, economy of machine utilization, etc. were considered, there turned out to be one overriding fact, namely, the extreme scarcity of transformational grammarians at present. Thus, the basic design principle was to make as little demand as possible on the linguist for non-linguistic work. This meant not only that the linguist should not be required to program, but also that he should be able to write grammars in the format which has been developed by the transformational

school. Of course, some special format conventions will be required to achieve machinability, but even here convenience to the linguist should be given more weight than convenience to the programmer or the machine.

Input and Output Formats and Control Cards

1. Form of the grammar. Each line of the grammar is punched on a separate card. If a line is too long for one card, a continuation card is used. The card preceding a continuation card will have a C in column 77.

The type and number of each main constituent structure and transformational rule will occupy a line of its own, and will start in column 1. Some typical rule labels are CS 15 or UT 7 or BT 3, where CS, UT and BT represent constituent structure rules, unary transformations and binary transformations, respectively. The rule label may be followed after one or more spaces by a mnemonic label. No restriction is placed on the number of spaces between the rule label and the mnemonic, but the mnemonic must be on the same card.

All symbols which can appear on the left side of a CS rule will consist of from 1 to 5 contiguous alphabetic or numeric characters, i. e., no special characters will be used within such symbols. The first character will be alphabetic, but the remaining may be mixed alphabetic and numeric.

There will be a special set of symbols called sentence boundary symbols which will start with a special character, preferably # if it is available on the printer.

The manuscript notation $\rightarrow$ will be replaced by = and the manuscript brace will be replaced by two asterisks, one

marking the upper end of the brace and one marking the lower end. Thus,

$$\left. \begin{array}{c} A \\ B \\ C \\ D \end{array} \right\} \quad \text{will be replaced by} \quad \begin{array}{c} *A* \\ B \\ C \\ *D* \end{array}$$

In the manuscript format, a line of symbols outside a brace is positioned at the center of the brace. In the input format, such lines will be "justified" upward to correspond with the top of the brace. Thus,

$$AB \left\{ \begin{array}{c} C \\ D \\ E \\ F \end{array} \right\} \quad \text{will be replaced by} \quad \begin{array}{c} AB*C* \\ D \\ E \\ *F* \end{array}$$

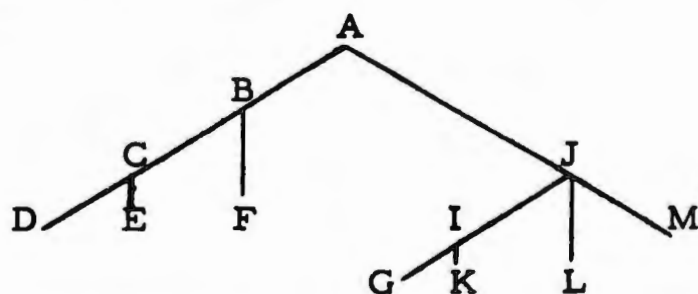
Also, in the manuscript format, braces on the same "line" are positioned so that their midpoints are at the same level. In the input format, braces on the same "line" will be justified upward so that their upper ends will be on the same line. Thus,

$$AB \left(\left\{ \begin{array}{c} C \\ D \\ E \\ F \end{array} \right\} \right) GH \left\{ \begin{array}{c} (I) \\ J \end{array} \right\} \quad \text{will be replaced by} \quad \begin{array}{c} AB(*C*)GH*(I)* \\ D \quad *J* \\ E \\ *F* \end{array}$$

Note that parentheses in the manuscript format will remain as parentheses in the input format.

In the manuscript format, trees are drawn with the root at the top and with descendants below their ancestors and usually positioned more or less symmetrically with respect to their ancestor. In the input format, the root will be at the left, descendants will be to the right of their ancestors, and

the descendents of a node will be positioned such that the top-most descendent (rightmost in the manuscript format) will be on the same line as its ancestor. Thus,



will be replaced by

```

A      J      M
      L
      I      K
      G
      B      F
      C      E
      D
  
```

In the manuscript format for context conditions, the position of the symbol on the left-hand side of the rule is usually indicated by a dash. In the input form, three contiguous asterisks will be used.

The initial strings will be represented in the same format as CS rules. The rule label will be STRT and the rule type will be INST. The grammar may start as follows:

```

STRT          INITIAL STRINGS
INST 1 = A * B   * F
             C D
             * E   *
  
```

The CS rules will follow. A typical form is illustrated below, and the symbols IF/n and TIF/n are discussed following the

illustrations.

CS 15

A = B * C (*DS1 E*) IF/1 F * (G) IF/2

H TIF/1

I JS2

K S3 TIF/2

* L M IF/3 (N) S4 *

IF/1+ ST ***

IF/2- T *** U

IF/3+ *** U

TIF/1- P Q R

D ***

E F

TIF/2+ X Y ***

Context-sensitive rules are represented by placing an IF statement immediately following a symbol, a right parenthesis or a right brace. If the IF statement immediately follows a symbol, it is associated with all subrules involving selection of the occurrence of that symbol. If the IF statement immediately follows a right parenthesis or right brace, it is associated with all subrules involving the selection of any symbol which occurs between the right parenthesis or brace and its corresponding left parenthesis or brace. Since the inclusion of complete IF statements at several points in a main rule would be unwieldy and would considerably reduce the transparency of the main rule, only a reference to an IF statement is included directly within the rule, and the complete statement is listed below the main rule. The reference consists of a slash followed by a number followed by a + or - sign,

for example, IF/3+. The + sign means that the context must be present, and the - sign means that it must not be present.

In addition to the usual context-sensitive condition represented by IF/n, we have included a special device represented by TIF/n. This is a context condition which permits reference to the current tree and not just the current string. This special device is a fundamental violation of the theory since it effectively does away with the constituent structure part of the grammar by making it essentially transformational. Thus, the TIF would not be used in a grammar which is intended entirely to be an expression of or a test of the theory. It is included in the design because it may turn out to be a useful ad hoc device in extending the grammar for practical applications.

The trees in the TIF statements are not necessarily complete subtrees of the current tree (where a complete subtree consists of a node and all that it dominates, but may specify only relevant parts of a tree. For example, TIF/2+ in the above illustration says only that A must have Y as an immediate ancestor and Y must have X as an immediate ancestor, but says nothing about possible siblings of Y.

The format for complex symbols is more flexible than required for the particular form of lexicon we are developing as part of the grammar. The feature part of the complex symbol can be treated either as an ordered vector or as an unordered collection of symbols, i. e., each position in the vector can be named and referred to by name. Each complex category type will have a feature vector of specified length and an auxiliary table containing names for some or all of the vector

positions. If a position is not given a name, it automatically takes its number as its name. Named positions can also be referred to by their position number. For example, if N is a complex symbol, its auxiliary table might have the form

<u>Complex Symbol</u>	<u>Vector Position Number</u>	<u>Position Name</u>
N	1	ANIM
	2	HUM
	3	CONC
	4	
	5	PROP
	6	COL

Each vector position can be filled by a string of characters, although our present type of lexicon requires only a plus or minus in each position. In the examples below, a single character will be used.

A complex symbol can have one of the following forms:

i. pure ordered vector

N [H, K, +, -, T, -]

ii. pure table reference

N//CONC = +, ANIM = H, 4 = -, COL = -, PROP = T,
HUM = K//

Note that the entries are unordered.

iii. mixed

N [, K, , -, ,]//PROP = T, COL = -, ANIM = H,
CONC = +//

where again, the entries within the double slashes are unordered.

The input format for transformational rules uses three lines: the first line is the structural index, the second line is a consecutive numbering of the items in the structural index, and the third line indicates the structural change. Each item in the first line defines a column running through all three lines, and the corresponding entries in lines 2 and 3 are left-justified with respect to the column. Thus, the manuscript format

```

ABC DEF IJ KL M ==> 3 0 P + 3 + 1 + Q 5 1
1      2      3 4 5

```

will be replaced by the input format

```

ABC DEF IJ          KL M
1      2      3          4 5
3      0      P + 3 + 1 + Q 5 1

```

Conditions imposed on transformational rules are called WHERE statements herein. The first line of each WHERE statement will have WHR beginning in column 4, followed by any number of spaces (but not going to a continuation card) followed by a "+" or a "-" followed by a code (CXT or FTR) followed by a space followed by either one number or by two numbers separated by a comma, where the numbers refer to the structural index. The + and - indicate that the condition must or must not obtain, respectively. One number is used when we wish to specify the context (CXT) or feature (FTR) content of the item indicated by the number. Two numbers are used when we wish to compare the contexts or feature contents of the two items indicated by the two numbers. In this latter case, only the + sign may be used.

The actual statement will start in column 7 of the next line. The following examples illustrate the various forms of WHERE statements (where BAK, BIK, BOK represent complex symbol position names).

```
WHR  +CXT  3
      A    B    C
                        D [, , E, , +] //26 = F, BIK = G, 19 = -//
      ***    H
      I [, J, -] // BIK = K, 1 = L//
```

The asterisks indicate the position of the item labeled 3 in the rule.

```
WHR  -FTR  3
      [, , A, B] // 14 = C, BAK = D//
```

```
WHR  +CXT  3,5
      ***    A
      B// 1, 5, BIK, 37//
```

```
WHR  +FTR  3,5
      // 2, 8, 9, BOK, BIK, 17//
```

```
WHR  +FTR  3,5
      // 2, 5// LX
```

In the last example, LX stands for any lexical item and indicates that features 2 and 5 of lexical items 3 and 5 and the lexical items themselves must be the same. If all the features must agree, we will write

WHR +FTR 3,5

// ALL //

Note that when features are to be compared, we indicate only the positions to be compared, not the content of the various positions.

The grouping of the unary transformations will be indicated by using the same rule number for all transformations in a given group. If a group is optional, the rule number will be followed by a - sign. The ordering and identifications of the transformations within a group will be indicated by a number separated from the rule number (and "-" sign, if present) by a slash. Thus, we may have the following sequence of unaries:

UT1

UT2/1

UT2/2

UT2/3

UT3

UT4-/1

UT4-/2

UT5

.

.

.

2. Conversion of compact form to expanded form. The input is a grammar in the form described above and control cards (described below). The control cards indicate that the desired operation is Conversion, and also indicate whether both main rules and subrules or just subrules are to be printed

out, and whether the conversion is to be applied only to CS-rules or to transformational rules as well. If printout of main rules is selected, the output should give each main rule exactly as in the input, followed by its subrules.

The CS-subrules should be grouped according to the IF (but not the TIF) statements associated with them, and the groups should be ordered in order of increasing IF number. In general, there may be several IF's associated with a given rule. A set of rules forms a group only if they have exactly the same set of associated IF statements. Groups characterized by multiple IF's are ordered as follows: groups with an IF/1 will precede groups without an IF/1. Among all groups with an IF/1, those with an IF/2 will precede those without an IF/2, etc. To illustrate, suppose some main rule has four IF's in it, and suppose that all combinations of IF's occur, then the groups would be ordered as follows:

IF/1	IF/2	IF/3	IF/4
IF/1	IF/2	IF/3	
IF/1	IF/2	IF/4	
IF/1	IF/2		
IF/1	IF/3	IF/4	
IF/1	IF/3		
IF/1	IF/4		
IF/1			
IF/2	IF/3	IF/4	
IF/2	IF/3		
IF/2	IF/4		
IF/2			
IF/3	IF/4		

character of the last symbol and the TIF reference, but the TIF reference should start no earlier than column 103.

The full IF and TIF statements should follow the last subrule in the same format as in the input. If the main rule as well as the subrule is printed out, the IF and TIF statements need appear only after the last subrule and not also after the main rule.

If a "line" does not fit in one line of printout (using 120 columns) it should be continued on the next line starting in column 16 if it is part of a subrule, and in column 106 if it is part of a TIF reference.

3. Grammar update. The input is a grammar in the compact form and one or more of the following ordered types of control cards and associated data. The first control card indicates that the desired operation is grammar update.

Type 1. The purpose of this type of update is to replace some symbol everywhere it occurs by some other symbol. The control symbol and the data are on the same card. The control symbol REPSYM starts in column 1 followed by any number of spaces followed by a symbol followed by any number of spaces followed by a symbol, but all three items must be on one card. The interpretation is that the first symbol is to be replaced everywhere by the second. If more than one replacement is desired, a separate card is used for each one.

Type 2. This is to replace an entire rule. The control symbol (starting in column 1) is REPRUL followed by any number of spaces followed by a main rule name and number. The replacement rule starts on the next card and is expressed

in the input format described above. If more than one replacement is desired, each is preceded by an appropriate control card.

Type 3. This is to delete an entire rule. The control symbol (starting in column 1) is DELRUL followed by any number of spaces followed by a main rule name and number.

Type 4. This is to add an entire rule. The control card is like that for REPRUL except that the control symbol will be ADDRUL. The rule to be added starts on the next card and is in the input format.

When rules are added or deleted, the rules following the point of insertion or deletion will have to have their rule numbers updated appropriately. This updating should not be done until all additions and deletions have been carried out, since the point of insertion or deletion is defined in terms of the pre-updated grammar.

4. Production of specified derivation and comparison with expected result. The following options are possible:

Input	{	compact format, or
Grammar		expanded format, or
Format		result of updating in the same run
Traffic	{	blocking, or
Rules		non-blocking
Deletion	{	immediate, or
		delayed
Special	{	use TIF's
Devices		do not use TIF's

Illegality	{ stop, or continue
Trace	{ none, or all strings, or all trees, or all strings and all trees, or as called for
Comparison	{ none, or final string, or final string and final tree

The first control card will have PROD followed by one or more blanks followed by the identification title of the grammar to be used. The next card will have MODE in columns 1- 4. Selection of options will be indicated on the MODE card as follows:

Col. 7- 9	for grammar format:
CMP	for compact format
EXP	for expanded format
UPD	for result of updating in the same run
Col. 13-15	for traffic rules:
BLK	for blocking grammar
NØN	for non-blocking grammar
Col. 19-22	for deletion:
IMMD	for immediate deletion
DLYD	for delayed deletion
Col. 25-28	for special devices:
TIF	for use TIF's
IGN	for ignore TIF's

Col. 31-34	for illegality:
STOP	for stop
CONT	for continue
Col. 37-40	for trace:
NONE	for none
STRG	for all strings
TREE	for all trees
STTR	for all strings and all trees
CALL	for as called for
Col. 43-46	for comparison:
NONE	for none
STRG	for string
STTR	for string and tree

The MODE card is followed by a sentence identification card. The identification starts with an S- in columns 1-2 followed by any number of alphanumeric characters, including blanks, but not going beyond column 42.

If a comparison string and tree is included, the next card will have EXP STRG starting in column 7, and the following card will contain the string in the input format. Following the string will be a card with EXP TREE starting in column 7, and the tree will be given on successive cards in the input format.

After the above cards (if present) comes the specified derivation. It is introduced by a card with DERIV starting in column 4. The sequence of rules (actually subrules) starts on the next card, and is given one rule to a card, with a sequence number starting in column 13 and the rule name and number starting in column 25. Each rule may be followed (after any

number of spaces) by either STRG, TREE, or STTR. If the trace option is CALL, the current string, tree, or both will be printed out after application of a rule followed by STRG, TREE, or STTR, respectively. If in any other trace mode, these symbols will be ignored.

The next card is an output control card permitting optional selection of one or more of the following for printout:

- Input control card
- Output control card
- Record of illegalities
- EXP STRG and associated string
- EXP TREE and associated tree
- DERIV and associated sequence of rules.

Printout of traces is not optional. If a trace is obtained, it is automatically printed out. Also, if a comparison is made, the results (YES or NO) are automatically printed out.

The output control card has OUTPUT in columns 1-6.

The print out selections will be indicated as follows:

Col. 13-16	for input control card
MODE	print out MODE control card (in input format)
blank	no printout
Col. 19-22	for output control card
OUTP	print out output control card (in input format)
blank	no printout
Col. 25-28	for illegalities
ILLG	print out record of illegalities (in format described below)
blank	no printout

Col. 31-34 for expected string

EXST print out expected string (in input format,
 starting with the EXP STRG card)

blank no printout

Col. 37-40 for expected tree

EXTR print out expected tree (in input format,
 starting with the EXP TREE card)

blank no printout

Col. 43-47 for specified derivation

DERIV print out specified derivation (in input format,
 starting with the DERIV card)

blank no printout

The various items printed out are ordered as follows:

Grammar identification

Sentence identification

Results of comparison. These are printed out on
the same line as the Sentence identification, as follows:

Starting in column 55: STRG followed by two blanks
followed by YES or NO (for match or not match, re-
spectively);

Starting in column 73: TREE followed by two blanks
followed by YES or NO.

MODE Control Card

Output Control Card

Record of illegalities - RECORD OF ILLEGALITIES
is printed out, starting in column 1, and then separate
lines are used for each illegality, with each line con-
taining the appropriate rule sequence number and rule
name and number as in the input format, followed,

starting in column 37, by a statement describing the illegality (for example, 9R23 WAS APPLICABLE, or 9R23 WAS NOT APPLICABLE).

EXP STRG and associated string (in input format)

EXP TREE and associated tree (in input format)

DERIV and associated sequence of rules (in input format)

Traces - in the order obtained. For each trace, TRACE is printed, starting in column 1 followed on the same line by the rule sequence number and rule name and number (in input format) which "triggered" the trace. On the next line, STRING is printed, starting in column 4, and the current string is printed on the next line in input format. Then, on the next line, TREE is printed, starting in column 4, and the tree is printed out in input format, starting on the next line.

The control card for the Conversion operation has EXPAND in columns 1-6 followed by one or more blanks followed by the identification of the grammar to be expanded.

The next card indicates what is to be printed out and what is to be saved on tape. This card has DISP in columns 1-4 and:

Col. 13-18	PRCOM	for print only compact grammar
	PRXPND	for print only expanded grammar
	PRBOTH	for print both (in format described above)
	blank	for print out neither
Col. 25-30	SVCOM	for save only compact grammar on tape at end of run

SVXPND	for save only expanded grammar
SVBOTH	for save both
blank	for save neither

The control card for the Update operation has UPDATE in columns 1-6 followed by one or more blanks followed by the identification of the grammar to be updated. The next card assigns an identification title to the updated grammar. It has IDENT in columns 1-5 followed by one or more blanks followed by an identification title. The following card indicates what is to be printed out and what is to be saved on tape. This card has DISP in columns 1-4 and:

Col. 13	PRUPD	print out update cards
	blank	do not print out update cards
Col. 25-30	PRPRE	for print out pre-updated grammar
	PRPOST	for print out updated grammar
	PRBOTH	for print out both
	blank	for print out neither
Col. 37-41	SVPRE	save pre-updated grammar on tape
	SVPOST	save updated grammar on tape
	blank	save neither

Present Status

The programming done thus far is described in detail in Part III of this report. The basic routines for automatic expansion of a grammar, for carrying out a derivation, for checking the legality of a derivation, and for providing traces are running. However, the present program does not as yet include provision for context restrictions (WHERE statements) on transformational rules. Also, the input and output formats

being used at present do not yet conform to the specifications described above, which were designed with convenience to the linguist in mind. The present formats are still in a form convenient to the programmer for debugging purposes.

PART III

PROGRAMMING OF THE GRAMMAR TESTER

F. Blair

Programming of the Grammar Tester

Contents

	<u>Page</u>
1.0 INTRODUCTION	1
1.1 Compiler	1
1.2 Derivation Tester	2
2.0 FUNCTION DESCRIPTIONS FOR THE GRAMMAR COMPILER	2
3.0 FUNCTION DESCRIPTIONS FOR THE DERIVATION TESTER	14
4.0 ON MATCHING ARBITRARY PROPER STRINGS IN TREES	27
4.1 Introduction	27
4.2 Pattern Specification	30
4.3 The Function Properanalysis	32
4.3.1 Literal match	32
4.3.2 Back-referenced match	32
4.3.3 Indefinite pattern or "Dollars" match	33
4.3.4 The pattern element NIL	33
4.3.5 LISP Functions for pattern matching	33
4.4 Some More General Comments on Matching Proper Strings in Trees	34
4.5 The Question of Efficiency and the Combinatorial Problems Involved	35
4.6 Conclusion	36
REFERENCES	37
FIGURES	follow 37

APPENDICES

- A, I PRINTOUT OF AN INPUT GRAMMAR
 (From Gutsin [])
- A, II PICASO PRINT OF S-EXPRESSION FOR
 AN EXPANDED GRAMMAR
- A, III LISTING OF THE PUNCHED CARDS FOR
 THE S-EXPRESSION FOR THE EXPANDED
 GRAMMAR
- B SAMPLE DERIVATION TEST
- C SELECTED EXAMPLES

BLANK PAGE

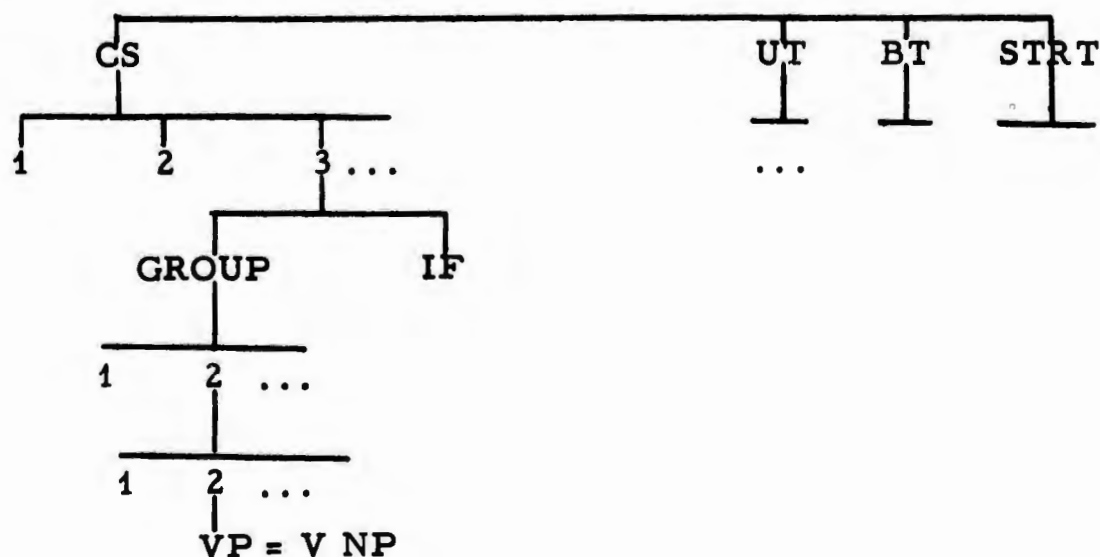
1.0 INTRODUCTION

This part of the report describes the implementation of a process for compiling and updating transformational grammars and a process for testing such grammars. Both processes were written in LISP 1.5^[3] and some knowledge of that language will be presumed in these notes.

1.1 Compiler

The Compiler facilities provide a notation and a translation-expansion procedure for converting an input grammar on punched cards into an expanded grammar, which is in the form required by the Derivation Tester. The notation of the input grammar is described elsewhere in this report. Figure 1 provides a description of the conceptual organization of the translation-expansion procedure. Figure 2 gives a more detailed outline of the flow of control of this process. The reader who wishes to follow the flow of control further is directed to the individual function descriptions (Section 2).

The output of this process, the expanded, translated, and updated grammar, is a LISP s-expression^[3]. The conceptual form of the s-expression is that of a tree structure, e. g.,



Appendix A contains a sample input grammar, a printout of its expanded form, and the s-expression for the expanded form.

1.2 Derivation Tester

Within this system, testing a grammar is a conceptually simple process of checking whether it will generate the structures that it is intended to produce. In order to accomplish this objective, the user must specify for each structure the sequence of rules which he hypothesizes should be applied in order to produce it. A deck of cards, known as the Derivation Test Packet, with one rule number per card, is used for this purpose. Figure 3 illustrates the process of testing a given expanded grammar. Figures 4 and 5 illustrate, in much greater detail, the flow of control during testing. Still greater detail is provided in the function descriptions (Section 3). As part of the program for derivation testing, some functions were created to accomplish the matching of proper analyses in structure trees. These functions, which are of rather general interest, are discussed in Section 4 below. Appendix B includes printouts of several sample derivations.

2.0 FUNCTION DESCRIPTIONS FOR THE GRAMMAR COMPILER

2.1 Picaso [x; n]

Arguments: x is any s-expression^[3] to be printed.

n is the number to be assigned to the
current level. (initially, usually 1)

Description: Prints x in a form where substructures are indented according to their depth. The depth number is also

printed. Used to print the s-expression for the expanded grammar.

Requires subfunctions: Dopnt, Leonardo, Prtchar 1.

Example: See Figure 2

Remarks: For best results, the verbosity of the garbage collector^[3] should be suppressed.

2.2 Dave []

Description: Sets the free variable expgrm to the value of the function Gutsin []; that is, the s-expression for the expanded grammar.

Remark: See Figure 2

2.3 Gutsin [] (Grammar Update and Test System INput translator)

Description: Reads input cards representing the rules of a grammar until an ENDOFALL card is encountered, at which time it returns as value the cumulative expansions of these rules. This accumulation of rules is referred to as the s-expression for the expanded grammar. The new rules are accumulated on the s-expression expgrm which may be set to an existing grammar for updating purposes or may be initially set to ((CS)(UT)(BT)(STRT)) - the headings for Constituent Structure rules, Unary Transformations, Binary Transformations, and the set of initial strings, respectively. The schematic in Figure 2 illustrates the reading of these rule types by Read6 [] and their subsequent processing within Gutsin [].

Requires: Read6, Buread, Transf, Stots, Treescan

Remark: See Figure 2

2.4 Read6 []

Description: Reads one card record from the current input device (72 columns). Returns as value a list of symbols on the card. Any string of alphabetics or any string of numerals separated by commas or blanks constitutes a symbol to this function. Used for reading IF rules*, rule names, control cards, and derivation cards.

2.5 Buread [x]

Argument: x is vacuous; that is, it is an artifact of a previous version of the function which survived to this time.

Description: Causes one Bracket-expression form to be read from cards, translated, and expanded. Returns as value the grouped, sorted, and expanded corresponding B-expression. This is the main function for processing CS rules. Has the effect of filling the character matrix A1 and passing control to Buread1.

Requires: Carrchararray, Buread1.

Remarks: See Figure 2; see also Appendix C.

2.6 Buread1 [x]

Argument: x is the greatest depth, in cards, of the current Bracket-expression form.

Description: The character matrix A1 (x; 72) contains the Bracket-expression form. This array is printed by Printarray. The Scan function creates the corresponding B-expression for the form. Thus, a B-expression is the representation of a Bracket expression as an s-expression. The B-expression is expanded, then grouped and returned in a form for attachment to the current grammar.

*IF rules express context restrictions on the current string cs.

Requires: Printarray, Mkpairsfrmgrp, Group, Expandr,
Scan.

2.7 Scan [i; j; ipu; ipl; jpl; jpr]

Arguments: i = the current row index
j = the current column index
ipu = the present row upper limit
ipl = the present row lower limit
jpr = the present column right-hand limit
jpl = the present column left-hand limit

Description: A function which is used to translate that part of the matrix (A1) of character elements A1 (i; j) from ipu = (i present upper) to ipl = (i present lower) and jpl = (j present left) to jpr = (j present right), which constitutes a well formed Bracket-expression, into a linear S-expression form called B-expression. The translation replaces:

Bracket-expression ⁺	-with-	B-expression
$\left\{ \begin{array}{c} * A * \\ B \\ \vdots \\ * \quad * \end{array} \right\}$		(ORR A B ...)
and		
(A)		(ORR A XO)

Requires: Scanbox, Par, Mkatom

Example: Bracket-expression

```

R16  *   *   INT  * IFPL * (OTHER) (REL CMP)
      *   TOT  *
      * ONE          *
```

+ The braces implied by the four asterisks (the "asterisk box") are included in the figure for purposes of legibility, but do not actually appear in the input or output of the system.

becomes: B-expression

```
( (R16 ( (ORR ( (ORR INT TOT) IFPL) ONE) ( (ORR OTHER
XO) (ORR (REL CMP) XO) ) ) ) )
```

Remark: See Appendix C, (the output of Scan is the input of Expander, in the function Buread1)

2.8 Mkatom [i; j; jpr] (Make atom)

Arguments: i = current row position
j = current column position
jpr = column right-hand bound.

Description: Returns as value the atom whose first character starts in column j of row i of matrix A1, but cannot exceed jpr. Creates an atomic symbol from a sequence of non-blank characters.

2.9 Par [i; j] (Find Right Parenthesis)

Description: Scans row i of A1 from column j until an unbalanced ')' is encountered. The column number of that ')' is the value of Par.

2.10 Scanbox [i; j; iu; il; jl; jr]

Arguments: i and j are row and column indexes, respectively.
iu and jl are the coordinates of the upper left hand * of an * box. il and jr are vacuous.

Description: A subfunction of Scan for translating an * box.
A function on the matrix A1.

Requires: Scanboxb.

2.11 Scanboxb [i; j; iup; ilp; jlp; jrp]

Arguments: i, j are current row and column indexes.
iup = row number of the top of the * box
ilp = row number of the bottom of the * box
jlp = column number of the left side of the * box

jrp = to be determined, the right side of the box.

Description: Returns as value the B-expression for the implied * box.

Requires: Testar, Scan.

2.12 Testar [iup; i; j]

Description: Predicate tests if an * occurs in the jth column of A1 above row i, but not above iup. (row 2 is considered to be above row 3.)

2.13 Expandr [x]

Description: Where x = a B-expression such as produced by Scan [...]. The value is the result of removing all "XO" from each substring of the neet form of Expand [x]; [i. e., Mknt [Expand [x]]] unless that substring consists only of "XO", in which case the substring will appear as "NIL". We shall call this a list of proper simple expressions.

Remark: See Appendix C.

2.14 Mknt [x] (Make Neet)

Description: Transforms x of the form (((A(B(C(D))))(E F G))(H I J)) into the neet form: ((A B C D)(E F G)(H I J)). (The output of Expand is the above form rather than the neet form.)

2.15 Efiace [x; y]

Description: Removes all expressions Equal to x from list y.

2.16 Expand [u]

Description: Accepts as input a B-expression u such as produced by the Scan [...] function, and produces as output a list of alternative "simple expressions" implied by u. This subfunction of Expandr expands a compact B-expression into its list of implied strings.

Requires: L91

2.17 L91 [x; w1]

Description: A subfunction of Expand. x is current position in w1, the B-expression that is to be reduced to a list of simple expressions. Returns as value the implied list of simple expressions.

Requires: Expand, L912.

Remark: Modifies the free variable expandlst.

2.18 L912 [k; r; n]

Argument: n = list of B-expressions to be expanded
(may be partially expanded)

r = a list of the form (ORR a₁ a₂ a₃ ...)

k = a list of the form (a_i a_{i+1} ...)

where the a's are any B-expression.

Description: Returns as value a list whose elements are Expand[n']'s, where n' is the result of substituting a_i for r on n. Expand[n'] is done for all a_i in k to produce the resultant list.

Requires: Expand.

Remark: Modifies the free variable expandlst

2.19 Group [lst; n]

Description: Where lst is a list of proper simple expressions; returns as value a list of grouped and sorted simple expressions. The groups are numbered sequentially from n. The input simple expressions are grouped according to the IF's that they contain as described in the proposal. Within each group the set of simple expressions are ordered low to high.

Requires: Sortlohigh, Noifgt, Ifnoif, Ifnoifeq, Selectif, Remif.

Remark: The effect of Group can be seen as the value of

Buread1 in the examples in Appendix C.

2.20 Sortlohigh [x]

Description: Where x is a list of lists. Reorders x and numbers from the free variable runo.

Requires: Sortlohi

Example: if runo = 2

Sortlohigh [((ABC)(B CUT)(A))] = ((2 A)(3 ABC)(4 B CUT))

2.21 Sortlohi [x] synonymous with Sort2 [x] or Sort1 [x]

Description: Where x is a list of lists to be ordered alphabetically in ascending order.

Sort2 [((ABC)(B CUT)(A))] = ((A)(ABC)(B CUT))

Requires: Lohi

Remark: The sort order can be observed in the printout of the expanded grammar, Appendix A, part II.

2.22 Lohi [x; y]

Description: Predicate tests if list x is alphabetically lower than list y.

Requires: Lap (LISP assembly language)^[3] subroutine

Lohiprintnm

Example: Lohi [(A B); (A C)] = T

Lohi [(A); (A C)] = T

2.23 Noifgt [lst; n]

Description: Predicate is true if there is no atom IF/ followed by a number greater than n on the S-expression lst.

2.24 Ifnoif [lst]

Description: Predicate is true if the atomic symbol IF/ does not occur on the S-expression lst.

2.25 Ifnoifeq [lst; n]

Description: Predicate is true if there is no atom IF/

followed by the number n on the s-expression lst.

2.26 Selectif [lst; n]

Description: Returns as value all lists in the s-expression lst which contain an atom IF/ followed by the number n.

Requires: Member1

2.27 Member1 [x; y; z]

Description: Predicate is true if y immediately follows x on list z.

2.28 Remif [lst; n]

Description: Removes all lists z on the s-expression lst for which Member1 [IF/; n; z] is true.

2.29 Mkpairsfrmgrp [x]

Description: Numbers the groups of simple expressions x from the free variable number ngrp. x is usually the output of Group [...].

2.30 Printarray [a1; i; j; ii; jj]

Description: Print array a1 from row i to row ii from column j to jj.

Remark: The results of Printarray calls in Buread 1 can be seen in Appendix C.

2.31 Carrchararray [x; i] (Card Character Array)

Arguments: x is first character in current startread buffer,^[3]
i denotes current row number.

Description: Reads cards until an EOF or a period in column 1 is encountered. Stores each character of the card as an element of the array A1. Ordinarily cards are read and stored in columns 1-71 of A1, but if a '/' appears in column 1, the subsequent characters go into 72-119.

Requires: Charray1

2.32 Charray1 [x; i; v]

Description: Stores away in A1 each character of the current card. That is, it fills a row or half-row.

2.33 Stots [] (Store Transformation)

Description: A function to read a transformational rule and convert it to the standard form for the Grammar tree. A subfunction of Gutsin.

Requires: Nextnonblankafterblank, Findatoms-dropplus, and Expandr.

Example: EEE \$ X $\left\{ \begin{array}{c} *T \quad * \\ \text{ING} \\ *EN \quad * \end{array} \right\} \left\{ \begin{array}{c} *M \quad * \\ V \\ \text{HAVE} \\ *BE \quad * \end{array} \right\} \quad X \quad \$$

0	1	2	3	4	5	6
0	1	2	4+-	3	5	6

Value is

```
((6) ($) (6) (5) (X) (5) (4) ((M) (V) (HAVE) (BE)) (3)
(3) ((T) (ING) (EN)) (4 -) (2) (X) (2) (1) ($) (1)
(0) (EEE) (0))
```

Remark: The transformational rules are either a single (Kx72) matrix of characters or a (Kx72) matrix followed by a card with a '/' ("slash") followed by an (Nx72) character matrix.

2.34 Nextnonblank [u; j]

Description: Finds the next non-blank character object in row u of the matrix of character elements A1 (10, 72), starting in column j. The value of the function is j', the column containing the first non-blank encountered to the right. If no non-blank is encountered by column 71, the value of the function is 71. A subfunction of Stots [] .

2.35 Nextnonblankafterblank [j; u]

Description: Has as value the first column j', of row u of

Matrix A1 (10, 72) which has a non-blank character object after the character object blank to the right of column j. If such a column is not found by column 71, the value of the function is 71. A subfunction of Stots.

2.36 Findatoms-dropplus [j; jr; row]

Description: A subfunction of Stots []. Operates on matrix of character objects A1 (10, 72). Its value is the list of all objects separated by plus '+' and terminated by 'blank' on row = row between and including columns j to jr. If a blank is encountered, or jr = j, the value is NIL.

2.37 Transf [x]

Argument: x is an S-expression representing the transformational rule, usually prepared by Stots.

Description: This function numbers and groups the input into a form compatible with the general form of the expanded grammar. The s-expression for this compatible form is the output of the function.

Requires: Mkpairsfrmgrp, Group, Mknt, Expand.

2.38 Treescan []

Description: Reads in one character matrix containing a two-dimensional tree; returns as value a standard s-expression representation of the tree.

Requires: Strmtx, Printarray, Triescan.

Example: A-B-C becomes (A(B(C))(D)(E(F)))



Remark: See example in Appendix C.

2.39 Strmtx [x; i]

Arguments: x is initial character on the i^{th} card.

Description: Atomic symbols on the i^{th} card are stored in matrix A1 as the $(\underline{i}, \underline{j})^{\text{th}}$ element; where $\underline{j}$ is the column of the first character of the atom. A subfunction of Treescan.

Remark: Works only for literal atoms (no numbers).

2.40 Triescan [cui; cuj; ilw]

Arguments: cui, cuj = the current row and column indexes in A1.

ilw = the current lower extent (in rows) of the tree.

Description: A subfunction of Treescan. Operates on the symbol array A1 to produce the standard s-expression for the contained tree.

Requires: Extntf.

2.41 Extntf [i; j; ilw]

Description: Finds the extent of the $(\underline{i}, \underline{j})^{\text{th}}$ node (i. e., the number of rows of A1 required by the descendants of A1 ($\underline{i}, \underline{j}$)).

Remark: For the example given with Treescan (in Appendix C), the extent of the A is 3 (i. e., Extntf [1; (the col. no. of A); 4] = 3).

3.0 FUNCTION DESCRIPTIONS FOR THE DERIVATION TESTER

3.1 Deriv []

Description: Tests a given packet of derivation cards, which specify the application of grammar rules, for consistency with the current expanded grammar, which is bound to the free variable symbol expgrm. The process is terminated when it encounters an OUTPUT, OUTP, or ENDUP card or whenever it exhausts the grammar or encounters a fatal error. This function does CS rule testing and then passes control to Derivtrans [] to test the transformational rules.

Requires: Read6, Curtrstr, Fstsubrlof, Spread, Nomorecs, Evalcs, Groupapplies, Getpsg, Tiftst, Applycr, Bp1, Derivtrans.

Remarks: Figure 4 illustrates the flow of control in Deriv [].

See Appendix B for examples of the use of this function.

3.2 Curtrstr [] (Current Tree and String)

Description: Creates initial current string, bound to the free variable cs. Creates the initial current tree bound to the free variable ctr. Returns a list of these two structures as value, but is executed primarily for its effect on cs and ctr.

Requires: Getpsg.

3.3 Getpsg [x; pr; pg; psg] (Get Present Subgroup)

Description: This function gets the subgroup for x (for example: CS) with rule number pr, group number pg, subgroup psg, out of the expanded grammar expgrm. If no such item is found, it returns the value NIL.

Example: Getpsg [CS; 4; 2; 2] = (2 VP = V NP).

3.4 Fstsubrlof [cr; cg] (First subrule of)

Description: Returns Getpsg [CS; cr; cg; 1]; that is, the first subgroup of group cg of CS rule cr.

3.5 Spread [x]

Argument: x = a list of symbols read from a derivation card by Read6[].

Description: If x = (OUTPUT), then the function returns the value NIL. If x is of the normal form for a derivation card (that is: a sequence number followed by the rule type, rule number, and optionally an R followed by the group and subgroup numbers), then the following free variables get their respective assignments:

seqno (sequence number), pr (rule number), and optionally pg (group number), psg (subgroup number).

3.6 Nomorecs [] (No More CS)

Description: Tests if there is no rule equal to CS cr.

Requires: Fstsubrlof.

Remark: See Figure 4.

3.7 Evalcs []

Description: Predicate tests whether the symbol on the left side of current subgroup, csg, is contained in the current string cs.

3.8 Groupapplies []⁺

Description: A predicate which tests if all the IF/n's* that can be found in csg apply to the current string cs. Uses function Listif [cdddr [csg]; IF/] to form a list of the IF numbers present in csg. The predicate Applyif[u] is used

+ This function will no doubt be changed in the future.

* IF rules express context restrictions on the current string cs.

on each IF number present in csg until either it fails, in which case Groupapplies fails, or it exhausts all IF's in csg, in which case Groupapplies succeeds. On success, the free variable iflst is assigned the list of cells in cs for which all IF's succeed.

Requires: Getcells, Listif, Applyif.

3.9 Getcells [u]

Description: A subfunction of Applyif [u]. Forms a list of all sublists of cs which begin with u.

Example: If cs = (A B A C A),
then Getcells [A] = ((A B A C A)(A C A) (A)).

3.10 Listif [x; u]

Description: Lists all n_i in u which follow an x, where the n_i are numbers.

Example: Listif [IF/; (A = B IF/ 1 C IF/ 2)] = (1 2)

3.11 Applyif [u] ⁺

Description: Tests if all IF/n_i apply to cs, where u =
(n₁ ... n_i ...).

Requires: Member3⁺, Member2⁺, Efface2, Getassocif.

3.12 Getassocif [u]

Description: Get rule IF/u associated with rule cr. A subfunction of Applyif.

3.13 Member2 [x; y]

Arguments: x, a list of symbols given as a pattern;
y, a list in which the pattern match is to find a
match (usually cs).

⁺ These functions will no doubt be redone in the future. At present, they are little tested and possibly not well defined.

Input free variables: aff--the current iflst, or list of cells (normally of cs) which contain the current symbol as car[] and have passed previously applied context restrictions; u2--the current IF rule, which contains a *** element, which must match the current symbol; cs--the current string.

Description: This function returns as value the list of elements of aff which corresponded to the *** in u2 when x (the pattern part of u2 with the current symbol substituted for ***) is made to overlay all segments of y that it exactly matches. This admittedly complicated function is used to test if a given context occurs and, furthermore, if a particular element appearing in that context is a member of the set of possible current symbols.

Requires: Member 3.

3.14 Member3 [x; y]

Description: Lists all cells of y which are the concatenation of list x with anything. Used to test for the occurrence of a given context.

Example: Member3 [(A, B); (R, A, B, C, A, B, A, B)]
= ((A, B, C, A, B, A, B)(A, B, A, B)(A, B))

3.15 Efface2 [x; y]

Description: Removes all items on list x from list y. [every occurrence]

3.16 Tiftst []*

Description: Tests whether CS pr R pg, psg does not hold for

* Tiftst is not complete at this time, but will be rigged to always give the value NIL. The main reason for this is that, as of yet, no TIF rules exist in the CS section of the current test grammar.

all TIF's* mentioned in the rule. Removes from iflst (the list of instances which survived all IF tests) those cells which the TIF conditions eliminate.

Requires: Treematch.

3.17 Treematch [x; y]

Description: Where: x is a pattern tree
y is an ordered list of input trees

Predicate tests if tree x can be found among y. If the pattern x contains an ***, which can match any single node, then the matched node, if found, is returned as value. There are three possible values:

NIL (cannot be found)

T (found, but x does not contain a ***)
 (node).

Uses Treetop, which requires terminalst.

3.18 Treetop [x; y]

Description: Where: x is ordered list of pattern trees,
y is an ordered list of input trees

Predicate tests if all pattern trees can be fitted, in a left-right ordering, to the ordered input trees. The fitting must be anchored at the head of each matched pair. Requires free variable list terminalst (a list of terminal cells of y). If there is a terminal *** in x, the matched terminal node is given as the value. There are three possible values:

NIL

T

(node)

* TIF rules are context restrictions on the current tree.

3.19 Applycr [] =

Description: The purpose of this function is to rewrite the current string cs and the current tree ctr. This is done by inserting on cs the rewrite section of csg for the first cell of cs that appears on list iflst (the list of cs terms which survived all IF/ and TIF/ rules).

Requires: Member4, Treeterminal, Subsk.

3.20 Member4 [x; y]

Description: Similar to Member3, but works on a list of terminal nodes of the tree y. Also gives first occurrence only.

Example: Member4 [(A, B); ((R)(A) (B)(C)(A)(B)(A)(B))]
= ((A)(B)(C)(A)(B)(A)(B)).

3.21 Treeterminal [x]

Description: Lists all terminal list nodes of x. Treeterminal [(L(A((C)(D))(B(C))) (A(C)(B(C(D)))) (A(B(C)))))]
= ((C)(D)(C)(C)(D)(C)).

3.22 Subsk [x; y; z]

Description: Substitute x for y on z. One occurrence only.

3.23 Bpl [x]

Description: Print tree x in the form shown in the example below. This form is called the subscript tree print.

Example: Bpl [(A (B) (C (D)))] prints as:

```
(  B  ( D))
A      C
```

3.24 Derivtrans []

Description: This function tests the transformational rules called for in the derivation packet and tests for their consistency with respect to ctr and the current expanded

grammar.

Requires: Finddeepest, Sonof, Getpg, Upair, Spread, Read6,
Properanalysis, Dotrn, Bpl, Members.

Remarks: See also flowchart of this function (Fig. 5).

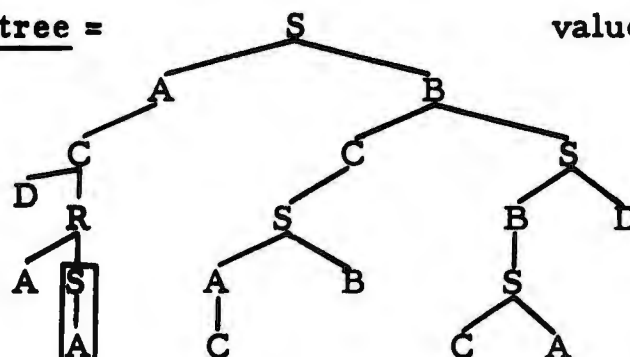
See Appendix B for samples of this function's action.

3.25 Finddeepest [e; tree; l]

Description: This function gives as value the leftmost of the set of deepest subtrees in tree which have root e and which are not on list l.

Example: Finddeepest [S; (S(A(C(D))(R(A)(S(A))))
(B(C(S(A(C))(B)))(S(B(S(C)(A))(D))))); ()]

that is, tree = value is (S(A))



Requires: Memberr

3.26 Memberr [l; x]

Description: True if x is Equal to some member of list l.

3.27 Sonof [k]

Description: The value of this function is the list of node pairs $((\underline{x}_1 \cdot \underline{y}_1) \dots (\underline{x}_i \cdot \underline{y}_i))$ such that node $\underline{x}_i$ has the parent $\underline{y}_i$ in the ancestral tree k. This value is used in the Proper-analysis function.

Example: Sonof [((A(B(C)(D))(E)))]
= (((A(B(C)(D))(E)).((A(B(C)(D))(E))))
((B(C)(D)).(A(B(C)(D))(E)))
((C).(B(C)(D)))
((D).(B(C)(D)))
((E).(A(B(C)(D))(E))))

3. 28 Getpg [n; cr9] (Get present group for transformational rules)

Arguments: n is either UT or BT
cr9 is a rule number.

Description: This function returns as value the s-expression which represents transformational rule n cr9.

Requires: Sass9c

Example: The external Bracket-expression is:

UT9

Q	(WP)	AUX
1	2	3
3 + 1	2	0

then the value of Getpg [UT; 9] is

((1 Q WP AUX)(2 Q NIL AUX))((1 3 1)(2 2)(3 0))

3. 29 Sass9c [x; y; u]

Description: Identical to Lisp function Sassoc, except that it uses Equal to test for equality rather than Eq.

3. 30 Upair [x] (Unpair)

Description: Where x is an s-expression of the following form:

((a₀ · (b₀₀ b₀₁ ...)) ... (a_n · (b_{n0} b_{n1} ...)))

Upair returns as value:

((a₀ a₁ ... a_n)((b₀₀ b₀₁ ...) ... (b_{n0} b_{n1} ...)))

Example:

Upair [((1·(3 1))(2·(2))(3·(0)))] = ((1 2 3)((3 1) (2) (0)))

3. 31 Properanalysis [v]

Arguments: v = cons[tree; cons[names; patterns]]

Description: This function provides a rather powerful ability to do pattern matching in trees. Within the scope of the derivation testing, its purpose is to test if the list given

as patterns can be considered a proper analysis in tree. If it can, then the corresponding nodes of the tree are paired with the corresponding names and the resulting list of pairs is given as output. Otherwise, the output is NIL. After establishing the free variables patterns, names, and mpairs, a test is made to see if the atoms which occur in patterns occur in tree. If this test fails, Proper-analysis fails. Otherwise control is passed to Fndpropanalysis. This function and its subfunctions are of sufficient interest to be discussed separately in more detail in Section 4.

Requires: Part, Listofatoms, Fndpropanalysis.

3. 32 Part [y; z]

Description: Tests if each atom of list y is a part of s-expression z.

Example: Part [(A); A] = T

Part [(A B); (B (A (C B)))] = T

Part [(A B); (B (E (C B)))] = F

3. 33 Listofatoms [x]

Description: Makes a list of all atoms in s-expression x (one occurrence only).

Requires: Union

3. 34 Union [m; n]

Description: Where m and n are lists representing sets.

Returns a list representing the union of the two sets.

Example: Union [(A B C A); (A C E)] = (A B C E)

3. 35 Fndpropanalysis [u] (Find proper analysis)

Arguments: u is the current tree.

free variables: patterns, names, and mpairs.

Description: This function tries to find a match for the first elements of patterns for which $P-a[\dots]$ can find matches for all the rest. The variable mpairs will contain the final value. $Fndpropanalysis[\underline{u}]$ tries all nodes of the tree $\underline{u}$, until it succeeds or exhausts the tree. The order in which the nodes are tried is given by the recursion:

```

Fndpropanalysis [u]
[make tests; if they succeed return T, otherwise:
r:= Fndpropanalysis [Car [u]];
r→return [r];
otherwise T→Fndpropanalysis [Cdr[u]].

```

Requires: P-a.

3. 36 P-a [x; p; n; mpairs]

Arguments: x is current tree

p is current list of pattern elements

n is current list of names

mpairs is current list of mached pairs of pattern elements and matching nodes.

Definitions of the pattern elements of p are given in section 4. 0.

Description: If p_1 (the current first pattern element) is a literal pattern element, then P-a must find a match on the content of the node x or the content of its leftmost descendant node x' ($cadr[\underline{x}]$) or $cadr[\underline{x}']$, etc. If no match can be found, a terminal node will be encountered and P-a fails. If, however, a match is found at some node $\underline{x}^i$, then $\underline{x}^i$ is paired with $\underline{n}_1$ ($car[\underline{n}]$) and appended to the list of matched pair (mpairs). Then the function returns the value of

P-a [Nxtcontiguous [xⁱ]; cdr[p]; cdr[n]; mpairs].

When P-a finally exhausts p, it returns the value of mpairs. In the case where p₁ is an indefinite pattern element, successively longer sequences of contiguous nodes will be paired with p₁ (starting with none (NIL)). If P-a continues to fail for all these cases, then this process will be repeated for x', x" ... xⁱ, etc. until either P-a succeeds or all such combinations are exhausted.

Requires: Nxtcontiguous, Comitrin, Gtname.

Remarks: For additional information about this matching facility, see section 4.0.

3. 37 Nxtcontiguous [x; u] (Next contiguous node to x)

Arguments: x is a node in a tree and u is a list of son-father pairs for some tree.

Description: Returns as value the next node to the right of u, that is, the right-hand sibling if it exists; otherwise, the right-hand sibling of the most immediate ancestor which has siblings.

Requires: Memberr.

Remarks: "Contiguous" spelled "contiguous" in the program.

3. 38 Comitrin [left; order]

Description: A function obtained from the METEOR report^[4].

This function is employed here to evaluate arguments of matching functions which occurred as pattern elements in Properanalysis.

Requires: Gtname.

3. 39 Gtname [name; prs]

Description: Evaluates the expression name with respect to the list of variable bindings prs.

Requires: Copytp.

3. 40 Copytp [x]

Description: Makes a copy of list x.

3. 41 Dotran [u; tr; ct] (Do the Transformation)

Arguments: u = a list of dotted pairs of the form:

((name_n · treepart_n)(name_{n-1} · treepart_{n-1}) ...
(name₁ · treepart₁))

tr = a list of pairs of the form:

((name₁ · (name_{x1} ...))(name₂ · (name_{y1} ...)) ...)

ct = the current tree in which the named tree parts are found.

Description: This function finds the associated treepart in ct given by u, of the left member of each pair of tr and replaces it with the list of treeparts given in the right member of the pair. Thus Dotran has the effect of transforming ct according to tr. For a given left member of tr, if either its associated right member on u is null, or its associated right member on tr has only one element and that element is identical with the left member, then no change in ct is made. Deletion is signified by a single right member of tr, that is, either EEE or zero. Any member of a right member list of tr which has no associated member on u will be inserted as a literal node.

Requires: Getallnode.

Example: Arguments of Dotran

u = ((6 +) (5 (IP (TO) (NP (N (PROP (DAVE)))) (MAN (BY)
(NP (N (PROP (DORITA)))) (4 V (HAND)) (3 ED) (2 (NP
(DET (THE)) (N (COM (BOOK)))) (AUX (T) (BE))) (1 +))

tr = ((1 1) (2 2) (3 0) (4 4 3) (5 5) (6 6))

ct = ((S (+) (NP (DET (THE)) (N (COM (BOOK))))) (AUX (]) (BE))
 (ED) (VP (V (HAND)) (IP (TO) (NP (N (PROP (DAVE)))))
 (MAN (BY) (NP (N (PROP (DORITA))))) (+)))

Value of Dotran

((S (+) (NP (DET (THE)) (N (COM (BOOK))))) (AUX (T)
 (BE)) (VP (V (HAND)) (ED) (IP (TO) (NP (N (PROP
 (DAVE))))) (MAN (BY) (NP (N (PROP (DORITA))))) (+)))

3.42 Getallnode [x; y; z]

Arguments: x is a node about to be replaced.

y is a list of the names of the nodes that are to
 replace x.

z is a list of dotted pairs (usually the u in Dotran).

Description: This function returns as value the node that is
 to replace x.

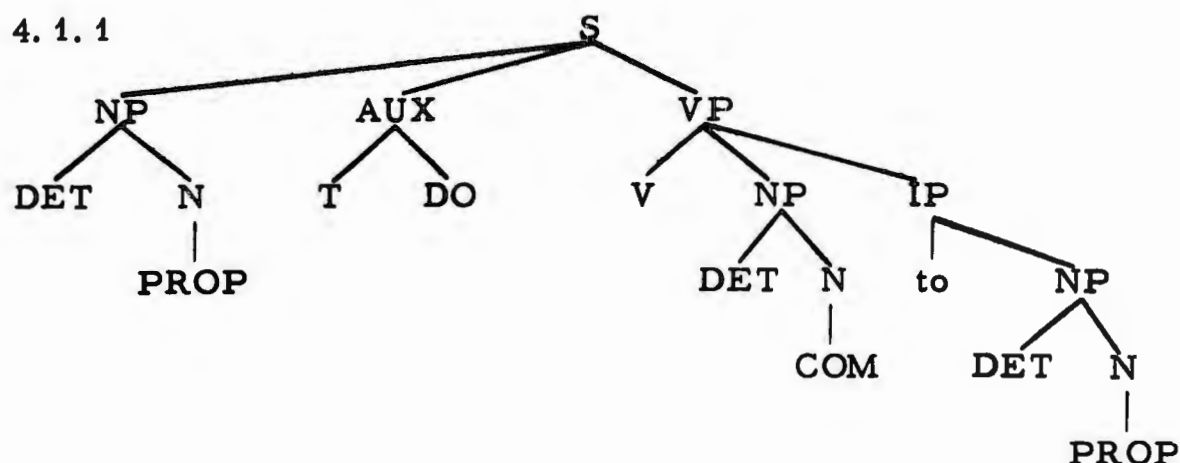
4.0 ON MATCHING ARBITRARY PROPER STRINGS IN TREES

A general pattern matching capability similar to that provided by the string pattern manipulating languages SNOBOL^[1], COMIT^[2], etc. has been developed within the context of a set of strings, known as proper strings, which is definable on a tree. The facility gives the user the capacity to carry out a rather powerful set of pattern matching operations in trees. The algorithm is programmed in LISP 1.5^[3] and is utilized as part of a computer system for testing transformational grammars, where it serves to perform the pattern matching involved in checking for applicability of transformational rules.

4.1 Introduction

One of the more interesting, if not entirely new, ideas in the field of procedure-oriented languages is that one can base a general-purpose programming language upon the concept of pattern identification. Just how this can be achieved is not going to be described in this note, but, for those who wish to pursue this topic, there are some references in the bibliography. Before proceeding to a description of a small extension of current capabilities, one general statement will be made in this connection: namely, that the specification of pattern criteria, the identification of the pattern elements, and the subsequent construction of new patterns which contain previously identified elements, is a natural way for humans to describe processes.

The use of trees as models of structures is well known and has wide applicability. One such model is that in which each node identifies, classifies, or otherwise describes the relationship of the set of nodes it dominates. For example, from linguistics we get the following incomplete model of a sentence:



This model implies, for instance, that:

(NP AUX V NP to NP)

is a legitimate sentence form; or, in more detail, a noun phrase (NP) followed by an auxiliary (AUX) followed by a verb (V) followed by (NP to NP) is a sentence form that can be generated in English. Note the importance that the order of nodes has. For example, (AUX NP V NP to NP) is not implied by the above tree, but (NP AUX V NP to NP) is. Note further that (NP DET AUX V NP to NP) is not implied by the tree, as the DET is a part of the NP. In this case, NP is said to dominate DET. Also note the (NP VP) is never an adequate description of a fragment of the sentence modeled by the tree, as there must be an AUX after NP and before the VP. Thus we have examples of the constraints of order, contiguity, and non-dominance.

In testing for applicability of transformational rules, it is typically required to be able to identify the occurrence of a particular sentence fragment: e. g., (DET N) in the current structure. This fragment does, in fact, satisfy the above-mentioned constraints of order, non-dominance, and contiguity. Any arrangement of the nodes of the tree fulfilling

these three requirements will be called a proper substring, a concept which linguists refer to by the term "proper analysis".

Two nodes, $\underline{x}$ and $\underline{y}$, in a tree $\underline{s}$, are said to constitute a proper substring of $\underline{s}$ if $\underline{y}$ is to the right of $\underline{x}$ and is either (a) the node that can be reached from $\underline{x}$ by ascending in the tree until a right-adjacent branch is found and then descending one level on that branch or (b) one of the set of nodes that can be reached from $\underline{x}$ by continuing down the path described in (a), always taking only leftmost branches. For example, in starting from the leftmost DET node in (4.1.1), a right-adjacent branch is found at the NP node immediately above it, and the nodes N and PROP are encountered in following the downward path; thus, (DET N) and (DET PROP) are proper substrings. In the case of the leftmost PROP node, the ascending portion of the path continues all the way to the S node before a right-adjacent branch (the one leading to AUX) is found; the proper substrings here are (PROP AUX) and (PROP T).

Given the above definition, the set of proper substrings for a tree can be defined recursively as follows: An ordered set of nodes $(n_1, n_2, \dots, n_k)$ constitutes a proper substring if, and only if:

$k = 1$ or

$n_i n_{i+1}$ is a proper substring for $i = 1$ to $k-1$.

One can order the set of proper substrings according to two criteria: first by depth, then by leftmost-ness. Observing this ordering, a partial listing of proper substrings for the above example tree (4.1.1) is as follows:

4. 1. 2 S *

NP

NP AUX

NP AUX VP

NP AUX V

NP AUX V NP

NP AUX V NP IP

NP AUX V NP to

NP AUX V NP to NP

NP AUX V NP to DET

NP AUX V NP to DET N

NP AUX V NP to DET PROP

NP AUX V DET

:

:

DET N

N

PROP

Having defined an ordered set, it is meaningful to pick the first member of the set for which some criterion holds. In this case of the set of proper substrings, the criterion employed is equivalence to some specified pattern.

4.2 Pattern Specification

A pattern is specified as a sequence of elements which must occur in the same sequence in the object string. Pattern matching consists of identifying with each pattern element this corresponding portion of the object string.

(* In this example only the content of the node is given, but what is actually implied is the node and all it dominates.)

Example:

sample pattern	sample matching string
'A' 'B'	A B
'A' x 'B'	$\left\{ \begin{array}{l} A B \\ A B B \\ A C B \end{array} \right\}$

where x is a variable which matches any string, including the null string, and 'A', 'B' are literal elements. Another way of expressing the pattern 'A' x 'B' is:

A \$ B
1 X 3

where each pattern element is identified by the assigned name or numeral appearing immediately below it. The \$ is called the indefinite pattern element and matches any string, including the null string. This notation is derived from COMIT^[1] and, more immediately, from METEOR^[4]. A variation of this notation is used for specifying transformational rules: The top line represents the pattern elements of a proper substring and the second line contains the names to be given to the substring fragments that they match. A third line is provided for specifying replacement.

A LISP function Properanalysis has been written which provides us with the facility to specify a string pattern and to identify its pattern elements with the nodes of the corresponding proper substring of a tree--the corresponding substring being the first one of the ordered set described in 4.1.2 for which an exact match can be established. The list of nodes corresponding to a given element is herein called a fragment.

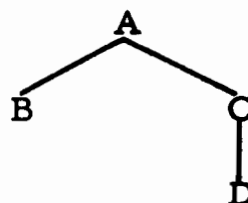
4.3 The Function Properanalysis [u]

Where u is a list of the form: (tree, (names·patterns))

patterns = a list of pattern elements, ex: (p_1 p_2 ... p_k)

names = a list of names for the associated elements,
ex: (n_1 n_2 ... n_k)

tree = the ordered list of nodes whose ancestors' siblings
are as yet unmatched, ex: ((A (B) (C (D))))
for the node



If a match cannot be found, the value of the function is NIL.

Otherwise, the list mpairs (for matched pairs) is returned as value. mpairs is a list of the form:

((name₁·matching-structure₁)...),

where name₁ is the i^{th} element of the list names, and matching-structure is that list of nodes of tree (and/or its ancestors' siblings) which matches the i^{th} member of the list patterns for the successful match. Allowable member types of the list patterns and their meanings are:

4.3.1 Literal match:

If the pattern element is an atom, and that atom is not the name of a previously matched fragment of the proper substring, then the atom will match only an identical contiguous atom of the proper substring.

4.3.2 Back-referenced match:

If the pattern element is the name of a previously matched fragment of a proper substring, then the element will match only an identical contiguous fragment.

4.3.3 Indefinite pattern or "Dollars" match:

The pattern element "\$" will match any contiguous proper substring fragment, including the empty fragment.

Note: In the future, a "fixed number of indefinite pattern elements" match might be included. This would be of the form $(\$ \cdot n)$ where n specifies the number of elements to match. Currently, we have had no occasion to use this type of capability, but it might well be useful in a more general environment.

4.3.4 The pattern element NIL:

In this case, the matching process simply continues as though this pattern element did not exist; i. e., it goes on to the next pattern element.

4.3.5 LISP functions for pattern matching:

A pattern element of the form $(FN \text{ function } \text{ name}_1, \text{ name}_2 \dots \text{ name}_k)$ is called a matching function. The idea of pattern matching functions is probably due to D. Bobrow, and their use here is practically identical to that described in the METEOR report^[4].

If the above matching function appeared as a pattern element, the LISP function "function" would have to be defined, and it would continue the pattern matching. (Note: this would allow violations of the set ordering, which might be desirable.)

function is a function of $k+1$ arguments, the first of which is x, the current node in the tree at which matching is to take place. The arguments, name₁ ... name_k, can be other matching-functions, names of previously matched pattern fragments, atoms or expressions. The matching function must obey the following conventions:

The entire tree, of which x is a node, is described by the list of (node ancestor) pairs given by the free variable son.

Any segment of the tree which is matched (m-patterns) should be paired with car[name] and mpairs set equal to cons[cons[car[name]; m-patterns]; mpairs], where m-patterns is assumed to be the list of nodes which constitute that matched segment.

If the function fails to find a match, it must return the value NIL.

4.4 Some More General Comments on Matching Proper Strings in Trees

While the operation of finding contiguous segment patterns in a structure is an important operation when a tree structure model is employed, it is by no means generally sufficient. For instance, the requirement of non-dominance makes it impossible to match many specifiable tree patterns. This requirement can, to a large extent, be overcome by the use of matching-functions. Matching-functions do not, however, meet the requirement for a natural and powerful means of pattern specification. Certainly the facility to match complete trees as literal pattern elements can be obtained immediately. But what if such a pattern-element tree-literal matched completely the top of a more developed subtree node? Would this be considered a pattern match? This may be made clearer by the following example:

Consider the sample tree, 4.1.1 and the pattern specified:

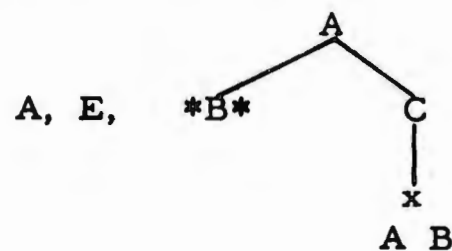


It is obvious that the above pattern occurs in the tree. But would one always agree that the following occurs?

(NP AUX VP)
 T

This brings up the even more general question of whether to allow tree-pattern elements with variables. A sample request for such pattern matching might be:

Identify the pattern:



in the current tree, where

B must be part of the current proper string; that is, (A E B) is a proper substring and B is immediately dominated by an A which dominates only a C which dominates an indefinite amount of structure called x which has a terminal node A to the left of a terminal node B, where A is not necessarily contiguous to B.

It appears that we are at present on the threshold of this kind of pattern matching facility. That is, a suitable set of matching functions could be defined to do the required matching. A more convenient notation than matching functions should be devised, but at least the basic facility is there.

4.5 The Question of Efficiency and the Combinatorial Problems Involved

The present proper-analysis function is a highly recursive procedure with certain tricks to make it more efficient. More such tricks will be added. It is critically important to identify the certainty of failure at the earliest possible instance. It is

precisely in the case of trying to find a pattern which does not exist that the worst combinatorial problems arise. Accurate timing data for illustrative pattern matching cases is not yet available, but should become so as an outgrowth of applying the Properanalysis function to the problem of transforming linguistic structures.

4.6 Conclusion

The proper-substring pattern matching in trees described in this paper, along with the ability to add the generality of LISP matching-functions, provide a usable way to identify patterns in trees. Application of this technique to testing generative transformational grammars has been tried and should provide interesting timing data in the future. Depending on the result of future studies involving fast-fail tricks, it may be possible to attempt an even more general matching of trees in trees.

REFERENCES

1. Farber, D. S., Griswold, R. E., and Polonsky, I. P., "SNOBOL, A string manipulation language". Journal of the ACM, Vol. 11, No. 1, January 1964.
2. An introduction to COMIT programming. The Research Laboratory of Electronics and the Computation Center, MIT, 1961.
3. LISP 1.5 Programmer's Manual, Computation Center and Research Laboratory for Electronics, MIT, August 1962.
4. Bobrow, D. G. "METEOR: A LISP Interpreter for String Transformations," THE PROGRAMMING LANGUAGE LISP: Its Operation and Applications. Berkeley and Bobrow (eds.), Information International, Inc., March 1964.

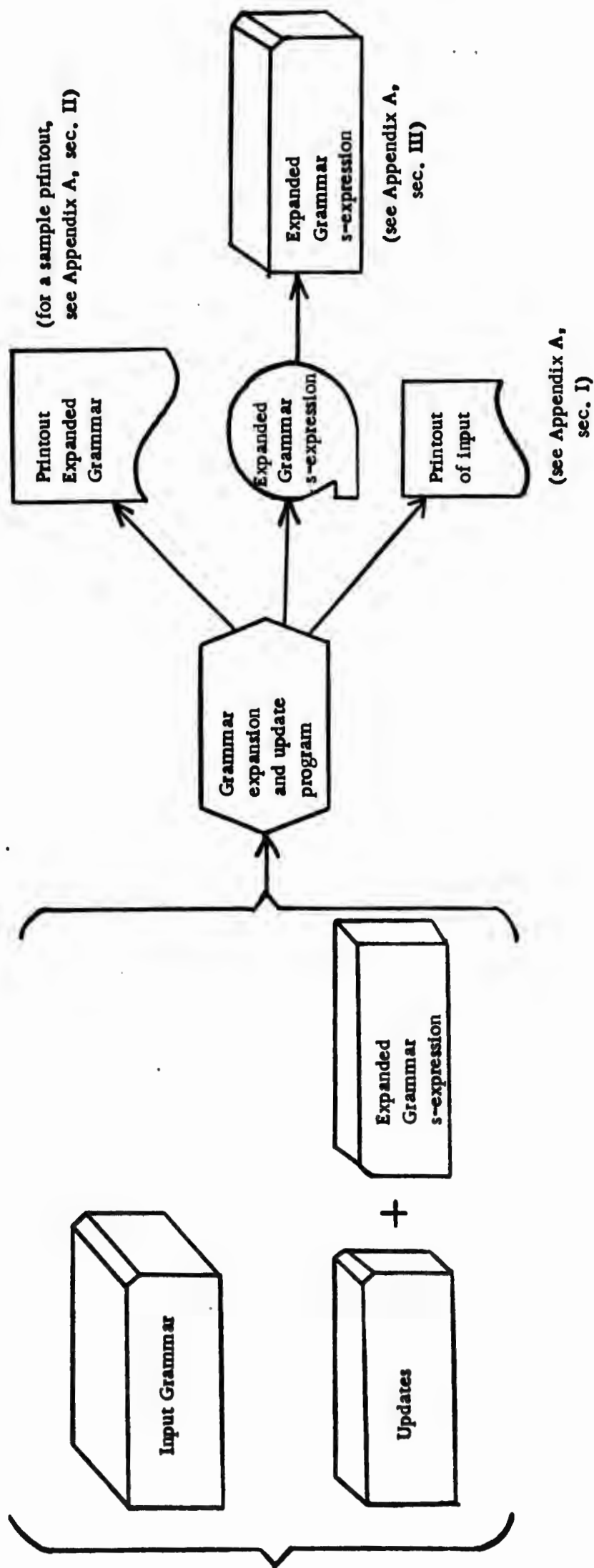
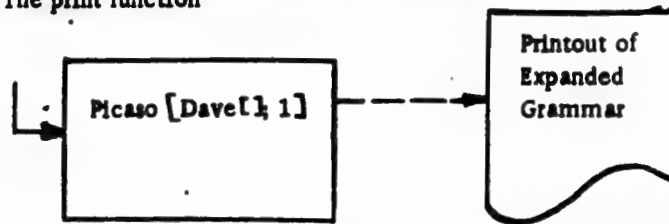


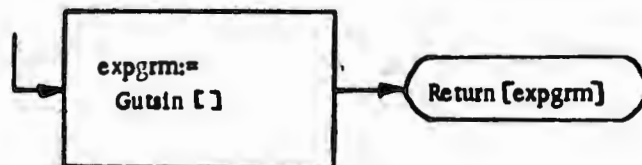
FIGURE 1. Simple schematic of the process of translating a set of grammar rules into a machine processable grammar.

FIGURE 2. Grammar Expansion and Update Program: An Outline

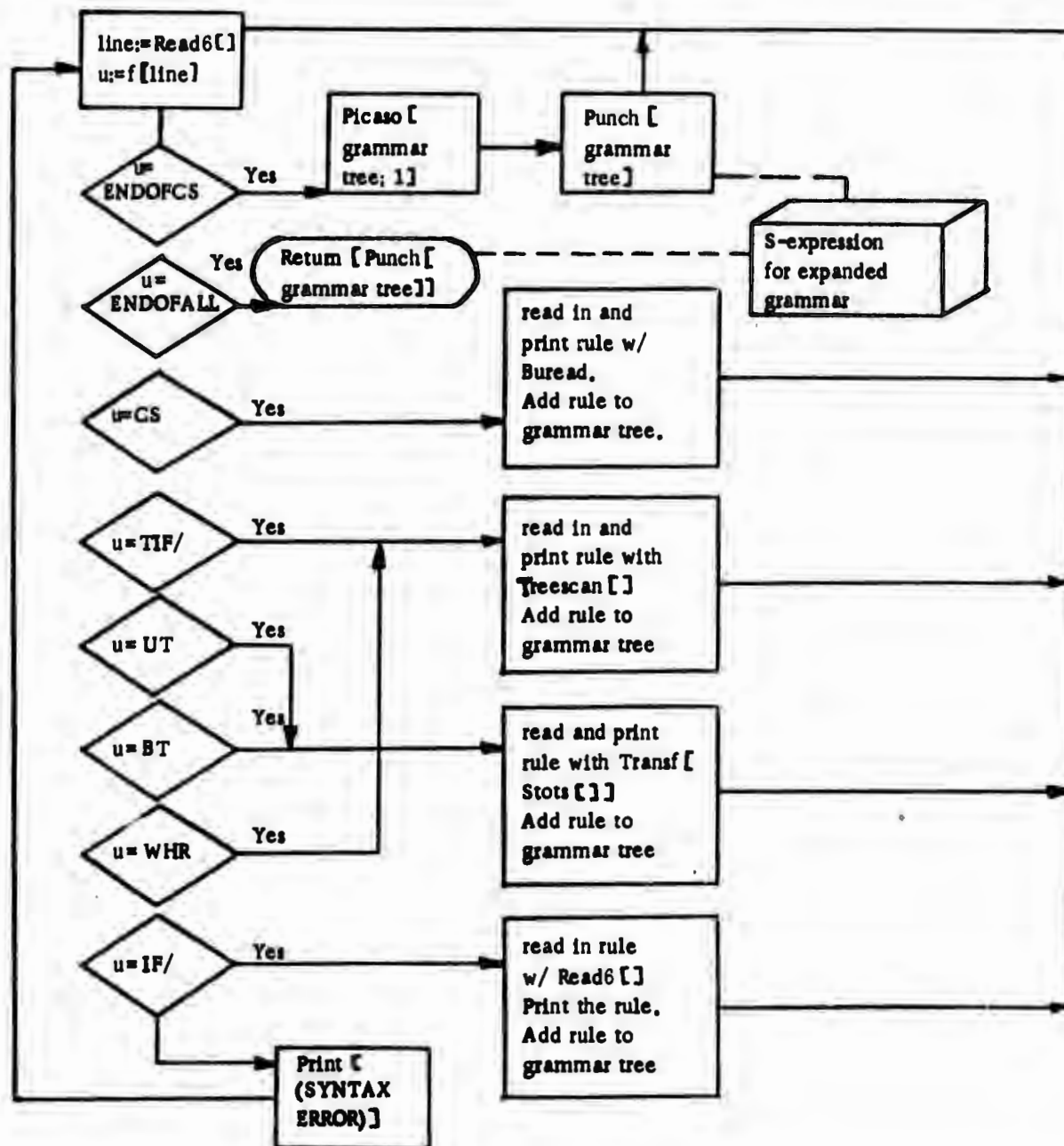
Picasso: The print function

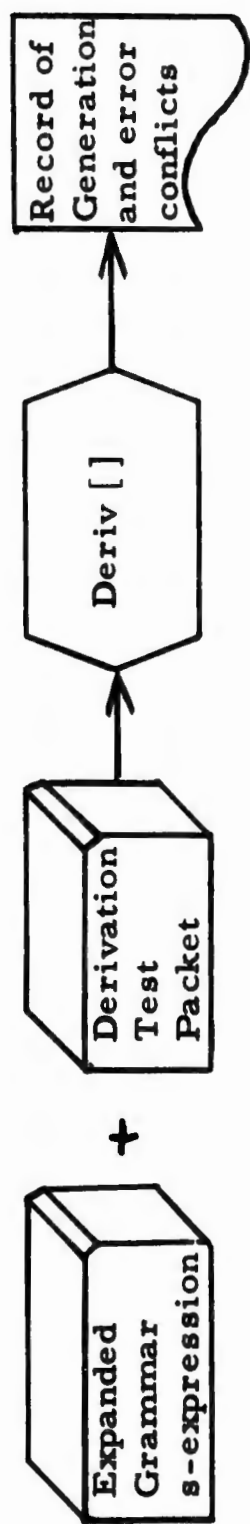


Dave: A function to cset the free variable expgrm to the value of the function Gutsin (the S-expression for an expanded grammar).



Gutsin: A function to read in grammar rules until an ENDOFALL card is encountered.



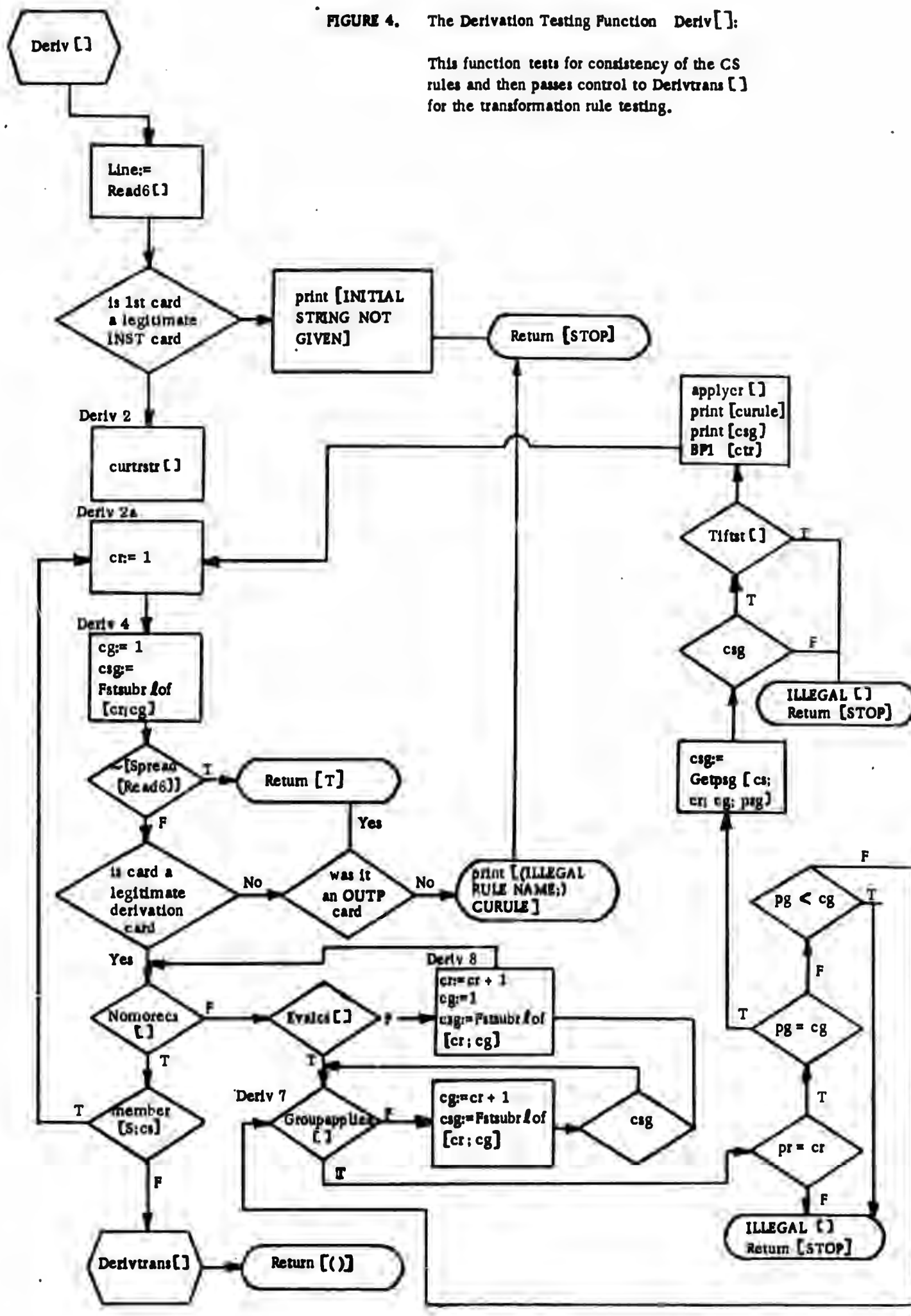


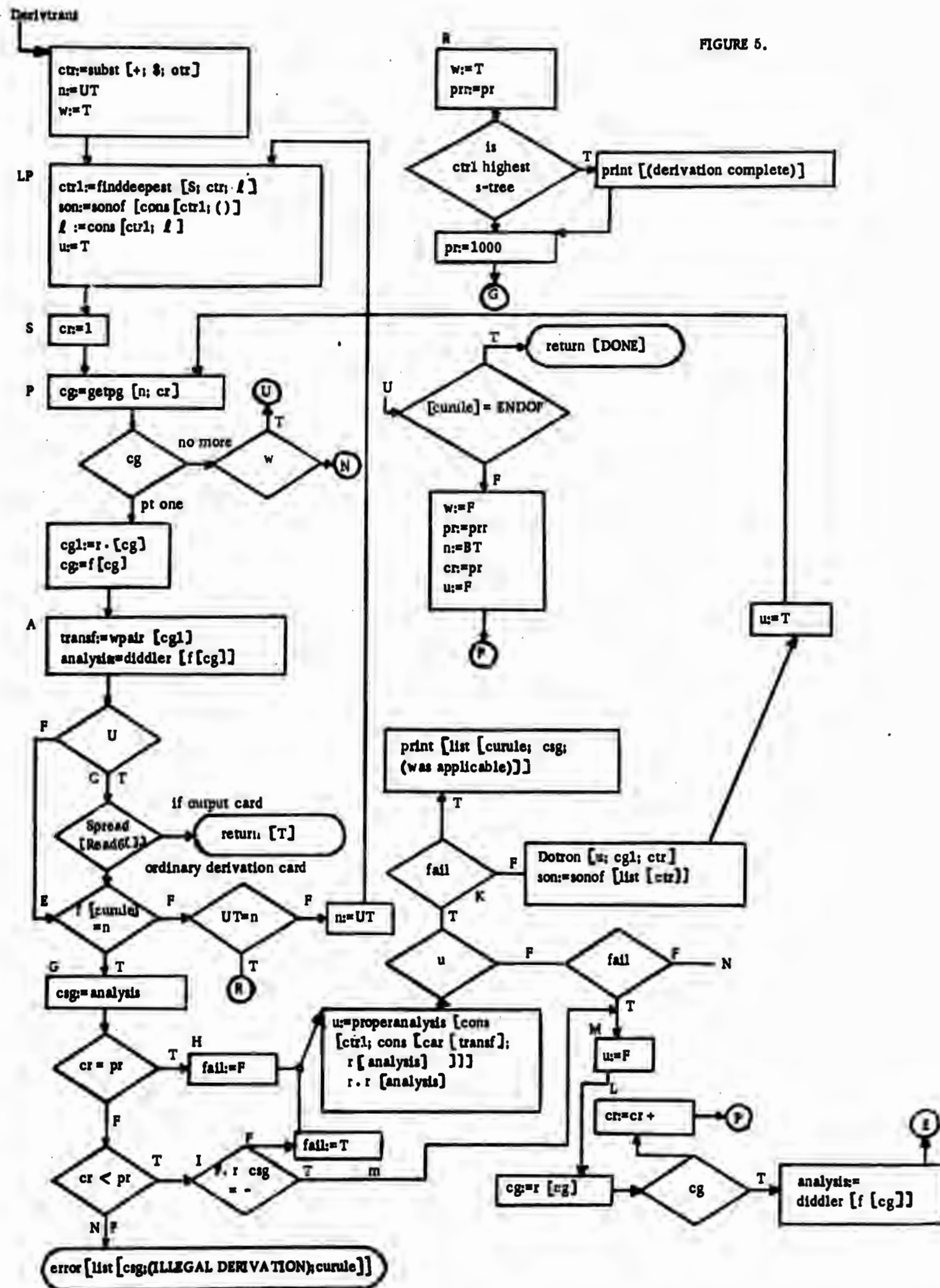
Simple schematic of the process of testing a grammar for consistency with an assumed derivation.

Figure 3

FIGURE 4. The Derivation Testing Function `Deriv[]`:

This function tests for consistency of the CS rules and then passes control to Derivtrans [] for the transformation rule testing.





APPENDIX A, I

PRINTOUT OF AN INPUT GRAMMAR

(From Gutsin [])

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
(LAMBDA NIL (PICASSO (DAVE) 1))
NIL

(STRT 1)

INST1 = S

(CS 1)

S = \$ (PRE) NP AUX VP \$

(CS 2)

PRE = (Q) (NEG)

(CS 3)

AUX = T DO (*M*)

I

(CS 4)

VP = (HAVE EN) (BE ING) *V (*NP*) (*PP*) (MAN) (TIME)

*

PP

NP

*S *

*BE ADJ

*

(CS 5)

NP = (DP) N (S)

(CS 6)

PP = PREP NP

(CS 7)

MAN = *BY P*

*ADV *

(CS 8)

TIME = (FREQ) (TIM) (DUR)

(CS 9)

PLACE = *(DEST) (ORIG)*

LOC

*DIST

*

(CS 10)

DP = (PD) DET

(CS 11)

DET = ART (S)

(CS 12)

ART = *INDEF*

*DEF *

(CS 13)

N = *SINCERITY*

BOY

IT

THING

CNE

MAN

HOOK

TABLE

PLAN

*FACT

*

(CS 14)

V = *CONDESCEND*

DISLIKE

LEAVE

BELIEVE

HAPPEN

TEMPT

GIVE

SEE

DECIDE

*CONVINCE *

(CS 15)

ADJ = *HONEST*

GREEN

TALL

DEEP

*FIRM *

(CS 16)

M = *UGHT TO*

CAN

WILL

*MUST *

(CS 17)

PREP = *ON*

OF

(UT 1 - INDIRECT OBJECT INVERSION)

\$ X NP IP Y \$

1 2 3 4 5 6

1 2 4 3 5 6

(UT 2 PASSIVE)

\$ NP AUX V (TO) NP Y BY SUM Z \$

1 2 3 4 5 6 7 8 9 10 11 12

1 2 7 4+BE+ED 5 6 0 8 9 3 11 12

(UT 3 TO DELETION)

\$ X V TO Y \$

1 2 3 4 5 6

1 2 3 0 5 6

(UT 4 MOVE)

\$ X T DO *M * Y \$

BE

1 2 3 4 5 6 7

1 2 3 5 0 6 7

(UT 5 QUESTION)

\$ X Q NP AUX Y \$

1 2 3 4 5 6 7

1 2 5 4 0 6 7

(UT 6 DEWDROP)

\$ X CG V Y \$

1 2 3 4 5 6

1 2 0 4 5 6

(ENDCFALL)

11

APPENDIX A, II

PICASO PRINT OF S-EXPRESSION
FOR AN EXPANDED GRAMMAR

[illegible]

7 (1 VP = HAVE EN BE ING V NP PP MAN TIME)
7 (2 VP = BE ADJ)
7 (3 VP = BE ING BE ADJ)
7 (4 VP = BE ING V)
7 (5 VP = BE ING V TIME)
7 (6 VP = BE ING V MAN)
7 (7 VP = BE ING V MAN TIME)
7 (8 VP = BE ING V S)
7 (9 VP = BE ING V S TIME)
7 (10 VP = BE ING V S MAN)
7 (11 VP = BE ING V S MAN TIME)
7 (12 VP = BE ING V NP)
7 (13 VP = BE ING V NP TIME)
7 (14 VP = BE ING V NP MAN)
7 (15 VP = BE ING V NP MAN TIME)
7 (16 VP = BE ING V PP)
7 (17 VP = BE ING V PP TIME)
7 (18 VP = BE ING V PP MAN)
7 (19 VP = BE ING V PP MAN TIME)
7 (20 VP = BE ING V PP)
7 (21 VP = BE ING V PP TIME)
7 (22 VP = BE ING V PP MAN)
7 (23 VP = BE ING V PP MAN TIME)
7 (24 VP = BE ING V PP S)
7 (25 VP = BE ING V PP S TIME)
7 (26 VP = BE ING V PP S MAN)
7 (27 VP = BE ING V PP S MAN TIME)
7 (28 VP = BE ING V PP NP)
7 (29 VP = BE ING V PP NP TIME)
7 (30 VP = BE ING V PP NP MAN)

7	(31 VP = BE ING V PP NP MAN TIME)
7	(32 VP = BE ING V PP PP)
7	(33 VP = BE ING V PP PP TIME)
7	(34 VP = BE ING V PP PP MAN)
7	(35 VP = BE ING V PP PP MAN TIME)
7	(36 VP = BE ING V NP)
7	(37 VP = BE ING V NP TIME)
7	(38 VP = BE ING V NP MAN)
7	(39 VP = BE ING V NP MAN TIME)
7	(40 VP = BE ING V NP S)
7	(41 VP = BE ING V NP S TIME)
7	(42 VP = BE ING V NP S MAN)
7	(43 VP = BE ING V NP S MAN TIME)
7	(44 VP = BE ING V NP NP)
7	(45 VP = BE ING V NP NP TIME)
7	(46 VP = BE ING V NP NP MAN)
7	(47 VP = BE ING V NP NP MAN TIME)
7	(48 VP = BE ING V NP PP)
7	(49 VP = BE ING V NP PP TIME)
7	(50 VP = BE ING V NP PP MAN)
7	(51 VP = BE ING V NP PP MAN TIME)
7	(52 VP = HAVE EN BE ADJ)
7	(53 VP = HAVE EN BE ING BE ADJ)
7	(54 VP = HAVE EN BE ING V)
7	(55 VP = HAVE EN BE ING V MAN)
7	(56 VP = HAVE EN BE ING V MAN TIME)
7	(57 VP = HAVE EN BE ING V NP)
7	(58 VP = HAVE EN BE ING V NP MAN)
7	(59 VP = HAVE EN BE ING V NP MAN TIME)
7	(60 VP = HAVE EN BE ING V NP)
7	(61 VP = HAVE EN BE ING V NP MAN)
7	(62 VP = HAVE EN BE ING V NP MAN TIME)
7	(63 VP = HAVE EN BE ING V NP NP)
7	(64 VP = HAVE EN BE ING V NP NP TIME)
7	(65 VP = HAVE EN BE ING V NP NP MAN)
7	(66 VP = HAVE EN BE ING V NP NP MAN TIME)
7	(67 VP = HAVE EN BE ING V NP PP)
7	(68 VP = HAVE EN BE ING V NP PP MAN)
7	(69 VP = HAVE EN BE ING V NP PP TIME)
7	(70 VP = HAVE EN BE ING V NP S)
7	(71 VP = HAVE EN BE ING V NP S MAN)
7	(72 VP = HAVE EN BE ING V NP S MAN TIME)
7	(73 VP = HAVE EN BE ING V NP S TIME)
7	(74 VP = HAVE EN BE ING V NP TIME)
7	(75 VP = HAVE EN BE ING V NP TIME)
7	(76 VP = HAVE EN BE ING V PP)
7	(77 VP = HAVE EN BE ING V PP)
7	(78 VP = HAVE EN BE ING V PP MAN)
7	(79 VP = HAVE EN BE ING V PP MAN)
7	(80 VP = HAVE EN BE ING V PP MAN TIME)
7	(81 VP = HAVE EN BE ING V PP MAN TIME)
7	(82 VP = HAVE EN BE ING V PP NP)
7	(83 VP = HAVE EN BE ING V PP NP MAN)
7	(84 VP = HAVE EN BE ING V PP NP MAN TIME)
7	(85 VP = HAVE EN BE ING V PP NP TIME)
7	(86 VP = HAVE EN BE ING V PP PP)
7	(87 VP = HAVE EN BE ING V PP PP MAN)
7	(88 VP = HAVE EN BE ING V PP PP MAN TIME)
7	(89 VP = HAVE EN BE ING V PP PP TIME)
7	(90 VP = HAVE EN BE ING V PP S)

7	(91 VP = HAVE EN BE ING V PP S MAN)
7	(92 VP = HAVE EN BE ING V PP S MAN TIME)
7	(93 VP = HAVE EN BE ING V PP S TIME)
7	(94 VP = HAVE EN BE ING V PP TIME)
7	(95 VP = HAVE EN BE ING V PP TIME)
7	(96 VP = HAVE EN BE ING V S)
7	(97 VP = HAVE EN BE ING V S MAN)
7	(98 VP = HAVE EN BE ING V S MAN TIME)
7	(99 VP = HAVE EN BE ING V S TIME)
7	(100 VP = HAVE EN BE ING V TIME)
7	(101 VP = HAVE EN V)
7	(102 VP = HAVE EN V MAN)
7	(103 VP = HAVE EN V MAN TIME)
7	(104 VP = HAVE EN V NP)
7	(105 VP = HAVE EN V NP)
7	(106 VP = HAVE EN V NP MAN)
7	(107 VP = HAVE EN V NP MAN)
7	(108 VP = HAVE EN V NP MAN TIME)
7	(109 VP = HAVE EN V NP MAN TIME)
7	(110 VP = HAVE EN V NP NP)
7	(111 VP = HAVE EN V NP NP MAN)
7	(112 VP = HAVE EN V NP NP MAN TIME)
7	(113 VP = HAVE EN V NP NP TIME)
7	(114 VP = HAVE EN V NP PP)
7	(115 VP = HAVE EN V NP PP MAN)
7	(116 VP = HAVE EN V NP PP MAN TIME)
7	(117 VP = HAVE EN V NP PP TIME)
7	(118 VP = HAVE EN V NP S)
7	(119 VP = HAVE EN V NP S MAN)
7	(120 VP = HAVE EN V NP S MAN TIME)
7	(121 VP = HAVE EN V NP S TIME)
7	(122 VP = HAVE EN V NP TIME)
7	(123 VP = HAVE EN V NP TIME)
7	(124 VP = HAVE EN V PP)
7	(125 VP = HAVE EN V PP)
7	(126 VP = HAVE EN V PP MAN)
7	(127 VP = HAVE EN V PP MAN)
7	(128 VP = HAVE EN V PP MAN TIME)
7	(129 VP = HAVE EN V PP MAN TIME)
7	(130 VP = HAVE EN V PP NP)
7	(131 VP = HAVE EN V PP NP MAN)
7	(132 VP = HAVE EN V PP NP MAN TIME)
7	(133 VP = HAVE EN V PP NP TIME)
7	(134 VP = HAVE EN V PP PP)
7	(135 VP = HAVE EN V PP PP MAN)
7	(136 VP = HAVE EN V PP PP MAN TIME)
7	(137 VP = HAVE EN V PP PP TIME)
7	(138 VP = HAVE EN V PP S)
7	(139 VP = HAVE EN V PP S MAN)
7	(140 VP = HAVE EN V PP S MAN TIME)
7	(141 VP = HAVE EN V PP S TIME)
7	(142 VP = HAVE EN V PP TIME)
7	(143 VP = HAVE EN V PP TIME)
7	(144 VP = HAVE EN V S)
7	(145 VP = HAVE EN V S MAN)
7	(146 VP = HAVE EN V S MAN TIME)
7	(147 VP = HAVE EN V S TIME)
7	(148 VP = HAVE EN V TIME)
7	(149 VP = V)
7	(150 VP = V MAN)

7	(151 VP = V MAN TIME)
7	(152 VP = V NP)
7	(153 VP = V NP)
7	(154 VP = V NP MAN)
7	(155 VP = V NP MAN)
7	(156 VP = V NP MAN TIME)
7	(157 VP = V NP MAN TIME)
7	(158 VP = V NP NP)
7	(159 VP = V NP NP MAN)
7	(160 VP = V NP NP MAN TIME)
7	(161 VP = V NP NP TIME)
7	(162 VP = V NP PP)
7	(163 VP = V NP PP MAN)
7	(164 VP = V NP PP MAN TIME)
7	(165 VP = V NP PP TIME)
7	(166 VP = V NP S)
7	(167 VP = V NP S MAN)
7	(168 VP = V NP S MAN TIME)
7	(169 VP = V NP S TIME)
7	(170 VP = V NP TIME)
7	(171 VP = V NP TIME)
7	(172 VP = V PP)
7	(173 VP = V PP)
7	(174 VP = V PP MAN)
7	(175 VP = V PP MAN)
7	(176 VP = V PP MAN TIME)
7	(177 VP = V PP MAN TIME)
7	(178 VP = V PP NP)
7	(179 VP = V PP NP MAN)
7	(180 VP = V PP NP MAN TIME)
7	(181 VP = V PP NP TIME)
7	(182 VP = V PP PP)
7	(183 VP = V PP PP MAN)
7	(184 VP = V PP PP MAN TIME)
7	(185 VP = V PP PP TIME)
7	(186 VP = V PP S)
7	(187 VP = V PP S MAN)
7	(188 VP = V PP S MAN TIME)
7	(189 VP = V PP S TIME)
7	(190 VP = V PP TIME)
7	(191 VP = V PP TIME)
7	(192 VP = V S)
7	(193 VP = V S MAN)
7	(194 VP = V S MAN TIME)
7	(195 VP = V S TIME)
7	(196 VP = V TIME)
4	(5)
5	(GROUP)
6	(1)
7	(1 NP = DP N S)
7	(2 NP = DP N)
7	(3 NP = N)
7	(4 NP = N S)
4	(6)
5	(GROUP)
6	(1)
7	(1 PP = PREP NP)
4	(7)
5	(GROUP)
6	(1)

7	(1 MAN = BY P)
7	(2 MAN = ADV)
4	(8)
5	(GROUP)
6	(1)
7	(1 TIME = FREQ TIM DUR)
7	(2 TIME =)
7	(3 TIME = DUR)
7	(4 TIME = FREQ)
7	(5 TIME = FREQ DUR)
7	(6 TIME = FREQ TIM)
7	(7 TIME = TIM)
7	(8 TIME = TIM DUR)
4	(9)
5	(GROUP)
6	(1)
7	(1 PLACE = DEST ORIG)
7	(2 PLACE =)
7	(3 PLACE = DEST)
7	(4 PLACE = DIST)
7	(5 PLACE = LOC)
7	(6 PLACE = ORIG)
4	(10)
5	(GROUP)
6	(1)
7	(1 DP = PD DET)
7	(2 DP = DET)
4	(11)
5	(GROUP)
6	(1)
7	(1 DET = ART S)
7	(2 DET = ART)
4	(12)
5	(GROUP)
6	(1)
7	(1 ART = INDEF)
7	(2 ART = DEF)
4	(13)
5	(GROUP)
6	(1)
7	(1 N = SINCERITY)
7	(2 N = FACT)
7	(3 N = PLAN)
7	(4 N = BOOK)
7	(5 N = MAN)
7	(6 N = ONE)
7	(7 N = IT)
7	(8 N = BOY)
7	(9 N = TABLE)
7	(10 N = THING)
4	(14)
5	(GROUP)
6	(1)
7	(1 V = CONDESCEND)
7	(2 V = BELIEVE)
7	(3 V = CONVINCE)
7	(4 V = DECIDE)
7	(5 V = DISLIKE)
7	(6 V = GIVE)
7	(7 V = HAPPEN)

7	(8 V = LEAVE)
7	(9 V = SEE)
7	(10 V = TEMPT)
4	(15)
5	(GROUP)
6	(1)
7	(1 ADJ = HONEST)
7	(2 ADJ = FIRM)
7	(3 ADJ = DEEP)
7	(4 ADJ = GREEN)
7	(5 ADJ = TALL)
4	(16)
5	(GROUP)
6	(1)
7	(1 M = OUGHT TO)
7	(2 M = MUST)
7	(3 M = CAN)
7	(4 M = WILL)
4	(17)
5	(GROUP)
6	(1)
7	(1 PREP = ON)
7	(2 PREP = OF)
3	(UT)
4	(1)
5	(GROUP)
6	(.....)
7	(1 - \$ X NP IP Y \$)
6	(1 1)
6	(2 2)
6	(3 4)
6	(4 3)
6	(5 5)
6	(6 6)
4	(2)
5	(GROUP)
6	(.....)
7	(1 \$ X NP AUX V TO NP Y BY SUM Z \$)
7	(2 \$ X NP AUX V NIL NP Y BY SUM Z \$)
6	(1 1)
6	(2 2)
6	(3 7)
6	(4 4 BE ED)
6	(5 5)
6	(6 6)
6	(7 0)
6	(8 8)
6	(9 9)
6	(10 3)
6	(11 11)
6	(12 12)
4	(3)
5	(GROUP)
6	(.....)
7	(1 \$ X V TO Y \$)
6	(1 1)
6	(2 2)
6	(3 3)
6	(4 0)

6	(5 5)
6	(6 6)
4	(4)
5	(GROUP)
6	(.....)
7	(1 \$ X T DO M Y \$)
7	(2 \$ X T DO BE Y \$)
6	(1 1)
6	(2 2)
6	(3 3)
6	(4 5)
6	(5 0)
6	(6 6)
6	(7 7)
4	(5)
5	(GROUP)
6	(.....)
7	(1 \$ X C NP AUX Y \$)
6	(1 1)
6	(2 2)
6	(3 5)
6	(4 4)
6	(5 0)
6	(6 6)
6	(7 7)
4	(6)
5	(GROUP)
6	(.....)
7	(1 \$ X DO V Y \$)
6	(1 1)
6	(2 2)
6	(3 0)
6	(4 4)
6	(5 5)
6	(6 6)
3	(BT)
3	(STRT)
4	(1)
5	(GROUP)
6	(1)
7	(1 INST1 = S)

APPENDIX A, III

LISTING OF THE PUNCHED CARDS FOR THE
S-EXPRESSION FOR THE EXPANDED GRAMMAR

((CS (1 (GROUP (1 (1 S = \$ PRE NP AUX VP \$) (2 S = \$ NP AUX VP \$)))) (2 LISP0010
 (GROUP (1 (1 PRE = Q NEG) (2 PRE =) (3 PRE = NEG) (4 PRE = 0)))) (3 (GROLISP0020
 UP (1 (1 AUX = T DO M) (2 AUX = T DO) (3 AUX = T DO 1)))) (4 (GROUP (1 (LISP0030
 1 VP = HAVE EN BE ING V NP PP MAN TIME) (2 VP = BE ADJ) (3 VP = BE ING BLISP0040
 E ADJ) (4 VP = BE ING V) (5 VP = BE ING V TIME) (6 VP = BE ING V MAN) (7LISP0050
 VP = BE ING V MAN TIME) (8 VP = BE ING V S) (9 VP = BE ING V S TIME) (11LISP0060
 0 VP = BE ING V S MAN) (11 VP = BE ING V S MAN TIME) (12 VP = BE ING V NLISP0070
 P) (13 VP = BE ING V NP TIME) (14 VP = BE ING V NP MAN) (15 VP = BE ING LISP0080
 V NP MAN TIME) (16 VP = BE ING V PP) (17 VP = BE ING V PP TIME) (18 VP = LISP0090
 BE ING V PP MAN) (19 VP = BE ING V PP MAN TIME) (20 VP = BE ING V PP) (LISP0100
 21 VP = BE ING V PP TIME) (22 VP = BE ING V PP MAN) (23 VP = BE ING V PPLISP0110
 MAN TIME) (24 VP = BE ING V PP S) (25 VP = BE ING V PP S TIME) (26 VP = LISP0120
 BE ING V PP S MAN) (27 VP = BE ING V PP S MAN TIME) (28 VP = BE ING V PLISP0130
 P NP) (29 VP = BE ING V PP NP TIME) (30 VP = BE ING V PP NP MAN) (31 VP LISP0140
 = BE ING V PP NP MAN TIME) (32 VP = BE ING V PP PP) (33 VP = BE ING V PPLISP0150
 PP TIME) (34 VP = BE ING V PP PP MAN) (35 VP = BE ING V PP PP MAN TIME)LISP0160
 (36 VP = BE ING V NP) (37 VP = BE ING V NP TIME) (38 VP = BE ING V NP MLISP0170
 AN) (39 VP = BE ING V NP MAN TIME) (40 VP = BE ING V NP S) (41 VP = BE ILISP0180
 NG V NP S TIME) (42 VP = BE ING V NP S MAN) (43 VP = BE ING V NP S MAN TLISP0190
 IME) (44 VP = BE ING V NP NP) (45 VP = BE ING V NP NP TIME) (46 VP = BE LISP0200
 ING V NP NP MAN) (47 VP = BE ING V NP NP MAN TIME) (48 VP = BE ING V NP LISP0210
 PP) (49 VP = BE ING V NP PP TIME) (50 VP = BE ING V NP PP MAN) (51 VP = LISP0220
 BE ING V NP PP MAN TIME) (52 VP = HAVE EN BE ADJ) (53 VP = HAVE EN BE INLISP0230
 G BE ADJ) (54 VP = HAVE EN BE ING V) (55 VP = HAVE EN BE ING V MAN) (56 LISP0240
 VP = HAVE EN BE ING V MAN TIME) (57 VP = HAVE EN BE ING V NP) (58 VP = HLISP0250
 AVE EN BE ING V NP MAN) (59 VP = HAVE EN BE ING V NP MAN TIME) (60 VP = LISP0260
 HAVE EN BE ING V NP) (61 VP = HAVE EN BE ING V NP MAN) (62 VP = HAVE EN LISP0270
 BE ING V NP MAN TIME) (63 VP = HAVE EN BE ING V NP NP) (64 VP = HAVE EN LISP0280
 BE ING V NP NP TIME) (65 VP = HAVE EN BE ING V NP NP MAN) (66 VP = HAVE LISP0290
 EN BE ING V NP NP MAN TIME) (67 VP = HAVE EN BE ING V NP PP) (68 VP = HALISP0300
 VE EN BE ING V NP PP MAN) (69 VP = HAVE EN BE ING V NP PP TIME) (70 VP = LISP0310
 HAVE EN BE ING V NP S) (71 VP = HAVE EN BE ING V NP S MAN) (72 VP = HAVLISP0320
 E EN BE ING V NP S MAN TIME) (73 VP = HAVE EN BE ING V NP S TIME) (74 VPLISP0330
 = HAVE EN BE ING V NP TIME) (75 VP = HAVE EN BE ING V NP TIME) (76 VP = LISP0340
 HAVE EN BE ING V PP) (77 VP = HAVE EN BE ING V PP) (78 VP = HAVE EN BE LISP0350
 ING V PP MAN) (79 VP = HAVE EN BE ING V PP MAN) (80 VP = HAVE EN BE ING LISP0360
 V PP MAN TIME) (81 VP = HAVE EN BE ING V PP MAN TIME) (82 VP = HAVE EN BLISP0370
 E ING V PP NP) (83 VP = HAVE EN BE ING V PP NP MAN) (84 VP = HAVE EN BE LISP0380
 ING V PP NP MAN TIME) (85 VP = HAVE EN BE ING V PP NP TIME) (86 VP = HAVLISP0390
 E EN BE ING V PP PP) (87 VP = HAVE EN BE ING V PP PP MAN) (88 VP = HAVE LISP0400
 EN BE ING V PP PP MAN TIME) (89 VP = HAVE EN BE ING V PP PP TIME) (90 VPLISP0410
 = HAVE EN BE ING V PP S) (91 VP = HAVE EN BE ING V PP S MAN) (92 VP = HLISP0420
 AVE EN BE ING V PP S MAN TIME) (93 VP = HAVE EN BE ING V PP S TIME) (94 LISP0430
 VP = HAVE EN BE ING V PP TIME) (95 VP = HAVE EN BE ING V PP TIME) (96 VPLISP0440
 = HAVE EN BE ING V S) (97 VP = HAVE EN BE ING V S MAN) (98 VP = HAVE ENLISP0450
 BE ING V S MAN TIME) (99 VP = HAVE EN BE ING V S TIME) (100 VP = HAVE ELISP0460
 N BE ING V TIME) (101 VP = HAVE EN V) (102 VP = HAVE EN V MAN) (103 VP = LISP0470
 HAVE EN V MAN TIME) (104 VP = HAVE EN V NP) (105 VP = HAVE EN V NP) (10LISP0480
 6 VP = HAVE EN V NP MAN) (107 VP = HAVE EN V NP MAN) (108 VP = HAVE EN VLISP0490
 NP MAN TIME) (109 VP = HAVE EN V NP MAN TIME) (110 VP = HAVE EN V NP NPLISP0500
) (111 VP = HAVE EN V NP NP MAN) (112 VP = HAVE EN V NP NP MAN TIME) (11LISP0510
 3 VP = HAVE EN V NP NP TIME) (114 VP = HAVE EN V NP PP) (115 VP = HAVE ELISP0520
 N V NP PP MAN) (116 VP = HAVE EN V NP PP MAN TIME) (117 VP = HAVE EN V NLISP0530

P PP TIME) (118 VP = HAVE EN V NP S) (119 VP = HAVE EN V NP S MAN) (120 VLPSP0540
VP = HAVE EN V NP S MAN TIME) (121 VP = HAVE EN V NP S TIME) (122 VP = RLSP0550
AVE EN V NP TIME) (123 VP = HAVE EN V NP TIME) (124 VP = HAVE EN V PP) (LISP0560
125 VP = HAVE EN V PP) (126 VP = HAVE EN V PP MAN) (127 VP = HAVE EN V PLSP0570
P MAN) (128 VP = HAVE EN V PP MAN TIME) (129 VP = HAVE EN V PP MAN TIME) LISP0580
(130 VP = HAVE EN V PP NP) (131 VP = HAVE EN V PP NP MAN) (132 VP = HAVE LISP0590
E EN V PP NP MAN TIME) (133 VP = HAVE EN V PP NP TIME) (134 VP = HAVE EN LISP0600
V PP PP) (135 VP = HAVE EN V PP PP MAN) (136 VP = HAVE EN V PP PP MAN LISP0610
IME) (137 VP = HAVE EN V PP PP TIME) (138 VP = HAVE EN V PP S) (139 VP = LISP0620
HAVE EN V PP S MAN) (140 VP = HAVE EN V PP S MAN TIME) (141 VP = HAVE ELISP0630
N V PP S TIME) (142 VP = HAVE EN V PP TIME) (143 VP = HAVE EN V PP TIME) LISP0640
(144 VP = HAVE EN V S) (145 VP = HAVE EN V S MAN) (146 VP = HAVE EN V SLISP0650
MAN TIME) (147 VP = HAVE EN V S TIME) (148 VP = HAVE EN V TIME) (149 VPLISP0660
= V) (150 VP = V MAN) (151 VP = V MAN TIME) (152 VP = V NP) (153 VP = VLISP0670
NP) (154 VP = V NP MAN) (155 VP = V NP MAN) (156 VP = V NP MAN TIME) (LISP0680
57 VP = V NP MAN TIME) (158 VP = V NP NP) (159 VP = V NP NP MAN) (160 VPLISP0690
= V NP NP MAN TIME) (161 VP = V NP NP TIME) (162 VP = V NP PP) (163 VP LISP0700
= V NP PP MAN) (164 VP = V NP PP MAN TIME) (165 VP = V NP PP TIME) (166 LISP0710
VP = V NP S) (167 VP = V NP S MAN) (168 VP = V NP S MAN TIME) (169 VP = LISP0720
V NP S TIME) (170 VP = V NP TIME) (171 VP = V NP TIME) (172 VP = V PP) (LISP0730
173 VP = V PP) (174 VP = V PP MAN) (175 VP = V PP MAN) (176 VP = V PP MALISP0740
N TIME) (177 VP = V PP MAN TIME) (178 VP = V PP NP) (179 VP = V PP NP MALISP0750
N) (180 VP = V PP NP MAN TIME) (181 VP = V PP NP TIME) (182 VP = V PP PPLISP0760
) (183 VP = V PP PP MAN) (184 VP = V PP PP MAN TIME) (185 VP = V PP PP LISP0770
IME) (186 VP = V PP S) (187 VP = V PP S MAN) (188 VP = V PP S MAN TIME) LISP0780
(189 VP = V PP S TIME) (190 VP = V PP TIME) (191 VP = V PP TIME) (192 VPLISP0790
= V S) (193 VP = V S MAN) (194 VP = V S MAN TIME) (195 VP = V S TIME) (LISP0800
196 VP = V TIME)))) (5 (GROUP (1 (1 NP = DP N S) (2 NP = DP N) (3 NP = NLISP0810
) (4 NP = N S)))) (6 (GROUP (1 (1 PP = PREP NP)))) (7 (GROUP (1 (1 MAN = LISP0820
BY P) (2 MAN = ADV)))) (8 (GROUP (1 (1 TIME = FREQ TIM DUR) (2 TIME =) LISP0830
(3 TIME = DUR) (4 TIME = FREQ) (5 TIME = FREQ DUR) (6 TIME = FREQ TIM) (LISP0840
7 TIME = TIM) (8 TIME = TIM DUR)))) (9 (GROUP (1 (1 PLACE = DEST ORIG) (LISP0850
2 PLACE =) (3 PLACE = DEST) (4 PLACE = DIST) (5 PLACE = LOC) (6 PLACE = LISP0860
ORIG)))) (10 (GROUP (1 (1 DP = PD DET) (2 DP = DET)))) (11 (GROUP (1 (1 LISP0870
DET = ART S) (2 DET = ART)))) (12 (GROUP (1 (1 ART = INDEF) (2 ART = DEF LISP0880
)))) (13 (GROUP (1 (1 N = SINCERITY) (2 N = FACT) (3 N = PLAN) (4 N = BOL LISP0890
OK) (5 N = MAN) (6 N = ONE) (7 N = IT) (8 N = BOY) (9 N = TABLE) (10 N = LISP0900
THING)))) (14 (GROUP (1 (1 V = CONDESCEND) (2 V = BELIEVE) (3 V = CONV LISP0910
NCE) (4 V = DECIDE) (5 V = DISLIKE) (6 V = GIVE) (7 V = HAPPEN) (8 V = LISP0920
EAVE) (9 V = SEE) (10 V = TEMPT)))) (15 (GROUP (1 (1 ADJ = HONEST) (2 ADJ LISP0930
J = FIRM) (3 ADJ = DEEP) (4 ADJ = GREEN) (5 ADJ = TALL)))) (16 (GROUP (1 LISP0940
(1 M = OUGHT TO) (2 M = MUST) (3 M = CAN) (4 M = WILL)))) (17 (GROUP (1 LISP0950
(1 PREP = ON) (2 PREP = OF)))) (UT (1 (GROUP (1 - 5 X NP IP Y \$) (1 LISP0960
1) (2 2) (3 4) (4 3) (5 5) (6 6))) (2 (GROUP ((1 3 X NP AUX V TO NP Y BY LISP0970
SUM Z \$) (2 \$ X NP AUX V NIL NP Y BY SUM Z \$) (1 1) (2 2) (3 7) (4 4 LISP0980
E ED) (5 5) (6 6) (7 0) (8 8) (9 9) (10 3) (11 11) (12 12))) (3 (GROUP (LISP0990
(1 \$ X V TO Y \$) (1 1) (2 2) (3 3) (4 0) (5 5) (6 6))) (4 (GROUP ((1 \$ LISP1000
X T DO M Y \$) (2 \$ X T DO BE Y \$) (1 1) (2 2) (3 3) (4 5) (5 0) (6 6) (LISP1010
7 7))) (5 (GROUP ((1 \$ X QNP AUX Y \$) (1 1) (2 2) (3 5) (4 4) (5 0) (6 LISP1020
6) (7 7))) (6 (GROUP ((1 \$ X DO V Y \$) (1 1) (2 2) (3 0) (4 4) (5 5) (LISP1030
6 6)))) (BT) (STRT (1 (GROUP (1 (1 INST1 = S)))))) LISP1040

APPENDIX B

SAMPLE DERIVATION TEST

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..

DERIV

NIL

```
(INST 1 R 1 1)
(CS 1 1 2)
(2 S = $ NP AUX VP $)
( $ NP AUX VP $)
S
(CS 2 1 1)
(1 VP = V NP IP MAN)
( $ NP AUX ( V NP IP MAN ) $ )
S
VP
(CS 3 1 2)
(2 AUX = T DO)
( $ NP ( T DO ) ( V NP IP MAN ) $ )
S
AUX
VP
(CS 4 1 1)
(1 IP = TO NP)
( $ NP ( T DO ) ( V NP ( TO NP ) MAN $ ) )
S
AUX
VP
IP
(CS 5 1 1)
(1 MAN = BY SUM)
( $ NP ( T DO ) ( V NP ( TO NP ) ( BY SUM ) ) $ )
S
AUX
VP
IP
MAN
(CS 6 1 1)
(1 NP = DET N)
( $ ( DET N ) ( T DO ) ( V NP ( TO NP ) ( BY SUM ) ) $ )
S
NP
AUX
VP
IP
MAN
(CS 6 1 1)
(1 NP = DET N)
( $ ( DET N ) ( T DO ) ( V ( DET N ) ( TO NP ) ( BY SUM ) ) $ )
S
NP
AUX
VP
NP
IP
MAN
(CS 6 1 1)
(1 NP = DET N)
( $ ( DET N ) ( T DO ) ( V ( DET N ) ( TO ( DET N ) ( BY SUM ) ) $ ) )
S
NP
AUX
VP
NP
IP
MAN
(CS 7 1 1)
(1 N = PROP)
( $ ( DET ( PROP ) ) ( T DO ) ( V ( DET N ) ( TO ( DET N ) ( BY SUM ) ) $ ) )
S
NP
N
AUX
VP
NP
IP
MAN
```

(CS 7 1 2)
 (2 N = COM)
 (\$ (DET (PROP)) (T DO) (V (DET (COM)) (TO (DET N)) (BY SUM)) \$)
 S NP N AUX VP NP N IP NP MAN
 (CS 7 1 1)
 (1 N = PROP)
 (\$ (DET (PROP)) (T DO) (V (DET (COM)) (TO (DET (PROP))) (BY SUM)) \$)
 S NP N AUX VP NP N IP NP N MAN
 (CS 8 1 1)
 (1 PROP = DORITA)
 (\$ (DET ((DORITA))) (T DO) (V (DET (COM)) (TO (DET (PROP))) (BY SUM)) \$)
 S NP N PROP AUX VP NP N IP NP N MAN
 (CS 8 1 2)
 (2 PROP = DAVE)
 (\$ (DET ((DORITA))) (T DO) (V (DET (COM)) (TO (DET ((DAVE)))) (BY
 S NP N PROP AUX VP NP N IP NP N PROP MAN
 SUM)) \$)
 (CS 9 1 1)
 (1 DET = THE)
 (\$ ((THE) ((DORITA))) (T DO) (V (DET (COM)) (TO (DET ((DAVE)))) (BY
 S NP DET N PROP AUX VP NP N IP NP N PROP MAN
 BY SUM)) \$)
 (CS 9 1 1)
 (1 DET = THE)
 (\$ ((THE) ((DORITA))) (T DO) (V ((THE) (COM)) (TO (DET ((DAVE)))))
 S NP DET N PROP AUX VP NP DET N IP NP N PROP
 (BY SUM)) \$)
 MAN
 (CS 9 1 1)
 (1 DET = THE)
 (\$ ((THE) ((DORITA))) (T DO) (V ((THE) (COM)) (TO ((THE) ((BY SUM))) \$)
 S NP DET N PROP AUX VP NP DET N IP NP DET N PROP
 DAVE))))
 MAN
 (CS 10 1 2)
 (2 COM = BOOK)
 (\$ ((THE) ((DORITA))) (T DO) (V ((THE) ((BOOK))) (TO ((THE) (BY SUM))) \$)
 S NP DET N PROP AUX VP NP DET N COM IP NP DET N
 (DAVE))))
 MAN
 PROP

(CS 11 1 2)
 (2 V = HAND)
 (\$ ((THE) ((DORITA))) (T DO) ((HAND) ((THE) ((BOOK))) (TO ((S NP DET N PROP N COM NP DET IP NP DET THE) ((DAVE)))) (BY SUM)) \$)
 N PROP MAN
 89823
 (UT 2 1 2)
 (2 + \$ NP AUX V NIL NP \$ BY SUM \$ +)
 (+ ((THE) ((BOOK))) (T DO) BE ED ((HAND) (TO ((THE) ((DAVE)))) S NP DET N COM NP DET VP V IP NP DET N PROP
 (BY ((THE) ((DORITA)))) +)
 MAN NP DET N PROP
 90045
 ((UT 4 1 2) CR = 3 (1 + \$ V TO \$ +) (WAS APPLICABLE))
 (UT 4 1 2)
 (2 + \$ T DO BE \$ +)
 (+ ((THE) ((BOOK))) (T BE) ED ((HAND) (TO ((THE) ((DAVE)))) S NP DET N COM NP DET VP V IP NP DET N PROP
 BY ((THE) ((DORITA)))) +)
 NP DET N PROP
 90051
 (UT 7 1 1)
 (1 + \$ DET PROP \$ +)
 (+ ((THE) ((BOOK))) (T BE) ED ((HAND) (TO ((THE) ((DAVE)))) BY S NP DET N COM NP DET VP V IP NP DET N PROP
 (((DORITA)))) +)
 NP N PROP
 90058
 (UT 7 1 1)
 (1 + \$ DET PROP \$ +)
 (+ ((THE) ((BOOK))) (T BE) ED ((HAND) (TO ((DAVE)))) BY ((S NP DET N COM NP DET VP V IP NP N PROP
 DORITA)))) +)
 90062
 (UT 8 1 1)
 (1 + \$ ED V \$ +)
 (+ ((THE) ((BOOK))) (T BE) ((HAND) ED (TO ((DAVE)))) BY ((S NP DET N COM NP DET VP V IP NP N PROP
 DORITA)))) +)
 90105
 END OF EVALQUOTE, VALUE IS ..
 NIL

APPENDIX C

SELECTED EXAMPLES

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
BUREAD
((A1))

ARGUMENTS OF BUREAD1

5

A = B * C (* D S1 E *) IF/1 F * (G) IF/2 (IF/4)
H TIF/1
* I J S2 *
K S3 TIF/2
* L M IF/3 (N) S4 *

ARGUMENTS OF EXPANDR [output of scan]
((A 1 = (B ((ORR (C ((ORR (ORR (D (S1 E)) (H (TIF/ 1)) (I (J S2))) XO) ((IF/ 1) F))) (K (S3 (TIF/ 2))) (L (M
(IF/ 3) ((ORR N XO) S4)))))) ((ORR G XO) ((IF/ 2) (ORR (IF/ 4) XO))))))

VALUE OF EXPANDR

((A = B C D S1 E IF/ 1 F G IF/ 2 IF/ 4) (A = B C D S1 E IF/ 1 F G IF/ 2) (A = B C D S1 E IF/ 1 F IF/ 2 IF/ 4)
(A = B C D S1 E IF/ 1 F IF/ 2) (A = B C H TIF/ 1 IF/ 1 F G IF/ 2 IF/ 4) (A = B C H TIF/ 1 IF/ 1 F G IF/ 2)
(A = B C H TIF/ 1 IF/ 1 F IF/ 2 IF/ 4) (A = B C H TIF/ 1 IF/ 1 F IF/ 2) (A = B C I J S2 IF/ 1 F IF/ 2 IF/ 4)
(A = B C I J S2 IF/ 1 F G IF/ 2) (A = B C I J S2 IF/ 1 F IF/ 2 IF/ 4) (A = B C I J S2 IF/ 1 F IF/ 2)
(A = B C IF/ 1 F G IF/ 2 IF/ 4) (A = B C IF/ 1 F G IF/ 2) (A = B C IF/ 1 F IF/ 2 IF/ 4) (A = B C IF/ 1 F IF/ 2)
(A = B K S3 TIF/ 2 G IF/ 2 IF/ 4) (A = B K S3 TIF/ 2 G IF/ 2) (A = B K S3 TIF/ 2 IF/ 2 IF/ 4) (A = B K S3 TIF/
2 IF/ 2) (A = B L M IF/ 3 N S4 G IF/ 2 IF/ 4) (A = B L M IF/ 3 N S4 G IF/ 2) (A = B L M IF/ 3 N S4 IF/ 2 IF/ 4)
(A = B L M IF/ 3 N S4 IF/ 2) (A = B L M IF/ 3 S4 G IF/ 2 IF/ 4) (A = B L M IF/ 3 S4 G IF/ 2) (A = B L M IF/ 3
S4 IF/ 2 IF/ 4) (A = B L M IF/ 3 S4 I' / 2))

VALUE OF BUREAD1

((1 (73 A = B C D S1 E IF/ 1 F G IF/ 2 IF/ 4) (74 A = B C D S1 E IF/ 1 F IF/ 2 IF/ 4) (75 A = B C H TIF/ 1 IF/
1 F G IF/ 2 IF/ 4) (76 A = B C H TIF/ 1 IF/ 1 F IF/ 2 IF/ 4) (77 A = B C IF/ 1 F G IF/ 2 IF/ 4) (78 A = B C
IF/ 1 F IF/ 2 IF/ 4) (79 A = B C I J S2 IF/ 1 F G IF/ 2 IF/ 4) (80 A = B C I J S2 IF/ 1 F IF/ 2 IF/ 4) (81 A
= B C D S1 E IF/ 1 F G IF/ 2) (82 A = B C D S1 E IF/ 1 F IF/ 2) (83 A = B C H TIF/ 1 IF/ 1 F G IF/ 2) (84 A =
B C H TIF/ 1 IF/ 1 F IF/ 2) (85 A = B C IF/ 1 F G IF/ 2) (86 A = B C IF/ 1 F IF/ 2) (87 A = B C I J S2 IF/ 1 F
G IF/ 2) (88 A = B C I J S2 IF/ 1 F IF/ 2)) (3 (89 A = B L M IF/ 3 N S4 G IF/ 2 IF/ 4) (90 A = B L M IF/ 3 N S4
IF/ 2 IF/ 4) (91 A = B L M IF/ 3 S4 G IF/ 2 IF/ 4) (92 A = B L M IF/ 3 S4 IF/ 2 IF/ 4) (93 A = B L M IF/ 3
N S4 G IF/ 2) (94 A = B L M IF/ 3 N S4 IF/ 2) (95 A = B L M IF/ 3 S4 G IF/ 2) (96 A = B L M IF/ 3 S4 IF/ 2))
(5 (97 A = B K S3 TIF/ 2 G IF/ 2 IF/ 4) (98 A = B K S3 TIF/ 2 IF/ 2 IF/ 4) (99 A = B K S3 TIF/ 2 G IF/ 2)
(100 A = B K S3 TIF/ 2 IF/ 2)))

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..

TREESCAN

NIL

A B C D
E
F G
F G
R S
T

END OF EVALQUOTE, VALUE IS ..
((A (R (T) (S)) (B (F (G)) (F) (C (E) (D))))))

FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..

STOPS

NIL

EEE	\$	IT	\$	THAT	WH	X	CONST	X	\$	AUX	V
0	1	2	3	4	5	6	7	8	9	10	11
	1	2	0	4	5	PRE+7	6	8	0	10	11

/

CLEFT X	\$
12	13
7	13

ARGUMENTS OF EXPANDR
(EEE)

VALUE OF EXPANDR
(EEE)

ARGUMENTS OF EXPANDR
(\$)

VALUE OF EXPANDR
((\$))

ARGUMENTS OF EXPANDR
((((\$ (THAT (WH (X (CONST (X (\$ (AUX V))))))))))

VALUE OF EXPANDR
((((\$ (THAT (WH (X (CONST (X (\$ (AUX V))))))))

END OF EVALQUOTE, VALUE IS ..
(((14) ((((\$ (THAT (WH (X (CONST (X (\$ (AUX V)))))))) (14)) (13) ((14)) ((13)) (12) ((LEE)) (7

PART IV

"COMPUTER SUPPORT FOR LEXICON
DEVELOPMENT AND USE"

D. Lieberman

D. Lochak

This Part has been accepted for presentation at the Fourth
Annual Meeting of the AMTCL, July 26-27, 1966
Los Angeles, California

Computer Support for Lexicon Development and Use

Contents

	<u>Page</u>
Introduction	1
Internal Data Organization	2
Tape Organization	5
Facilities	6
Overall (Supervisory) Control	18
Figures	follow 21

Introduction

A large lexicon, consisting initially of about 7000 words, is being constructed as part of a transformational grammar of English being developed at the IBM Research Center (see Part I, section 2.2, this report). The form of this lexicon is an adaptation of ideas set forth in Chapter 2 of Aspects of the Theory of Syntax by N. Chomsky (Ref. 1).

The required computer programs fall into three main classes:

1. The usual clerical programs involved in any kind of lexical development--additions, deletions, modifications, partial or complete printouts in convenient formats, scans of various sorts to assist the lexicographer, etc.
2. programs to apply the redundancy rules either for experimental purposes, or to form a full derived lexicon.
3. programs to provide partial or full, base or derived lexicons, in a format suitable for input to a grammar tester currently under development.

Only the first of the above three classes of programs is described in this paper.

Internal Data Organization

The discussion below assumes some familiarity with the linguistic basis for the lexicon as presented in Ref. 1.

The data could be organized in a straightforward way by storing with each word all of its features, with each feature appropriately marked + or -. However, a much more compact and manipulable representation can be achieved by taking advantage of certain properties of the lexicon. Specifically, each word has a set of classes of features associated with it, and within each class, all features are relevant, i. e., each feature is marked either + or - for that word--there is no "does not apply" mark. Thus, if each class is listed in full just once, then for each word in the lexicon only the features marked plus for that word need be a part of the entry for that word. The features marked minus are automatically recoverable by reference to the full list of features for the given class--all features not marked plus for that word are automatically minus. We could as well have used the convention that only minuses would be listed with each word and recover the pluses automatically. However, since it is expected that there will be many more minuses than pluses in the lexicon, it is more economical to use the convention that only features marked plus will be explicitly listed with each word.

As a further economy, one can number the features in each class, and then use the numbers rather than the full

feature specification for coding the individual words.

The lexicon contains five types of features:

SC - Syntactic category

SS - Strict subcategorization (categorical contexts)

CO - Co-occurrence (feature contexts)

IN - Inherent

RU - Transformational rule selection

There is only one class of SC features, but for each SC feature there is a separate class of each of the other types of features.

This point becomes important here only because numbers rather than full feature specifications will be used for coding.

For example, the code SS3 will not represent the same feature when used to code a noun as it will when used to code a verb.

Thus, it would seem that each class would have to be uniquely labeled and the coding for each word in the lexicon would have to include a reference to the relevant classes. However, one can again take advantage of a particular property of the lexicon, namely, that among all of the features marked plus for any word, there can be only one of type SC. Thus, if the full lists of features are organized with respect to the SC features, there is no need for additional labeling. The SC code associated with each word will automatically reference the appropriate lists for the other types of features.

The programming was done in SNOBOL, with extensive use of the tree functions. The tree-like formats described below were motivated by the use of the SNOBOL tree functions and do not necessarily have linguistic significance.

The GLOSSARY (the full lists of feature specifications) as described above could have the format shown in Figure 1, with

the SC type of feature in a special position. However, the programming and the input format for glossary modification can be simplified if all the feature types are represented uniformly. Therefore, at the expense of some redundancy, the glossary format actually used is as shown in Figure 2.

The format for the LEXICAL ENTRIES would follow almost directly from the linguistic theory were it not for the existence of homographs. If we decided not to give homographs any special treatment, one might use the lexical entry format shown in Figure 3. However, programming considerations made it desirable to define a logical unit in such a way as to explicitly connect homographs. This was done by establishing a dummy root, with the lexical entries as branches. Thus, a two-member homograph has the internal format shown in Figure 4. Since homographs were represented by using a dummy root, it turned out to be convenient to also use a dummy root with non-homographic entries. Thus, a non-homographic entry has the internal format shown in Figure 5. The cost of the dummy node on tape is just two characters (open and close parentheses in the SNOBOL linear representation of trees). Since each lexical entry will require 50 to 100 characters, the additional storage space occupied by the dummy node is negligible.

It is perhaps worth mentioning here that the above discussion of formats is concerned only with internal machine (and tape) formats. As far as the lexicographer is concerned, each entry is a separate entity whether or not it is homographic with some other entry. The dummy nodes are "unseen" by the lexicographer.

The Redundancy Rules are represented by strings. Each string consists of the characters RD, followed by a number (the redundancy rule number), followed by a period, followed by a sequence of feature types and numbers separated by commas, followed by a slash, followed by a sequence of feature types and numbers separated by commas. The interpretation is that any entry which satisfies the first segment of feature types and numbers is modified in accordance with the second set of features. For example, if SC 1 represents the feature NOUN, IN 1 the feature HUMAN, and IN 2 the feature ANIMATE, then a redundancy rule (for example, Rule 23) that all +HUMAN nouns are also +ANIMATE would be represented internally in the form:

RD23.SC1, IN1/IN2.

Each feature type can be preceded by a minus. The absence of a minus sign is interpreted as a plus.

Tape Organization

The first physical tape record is the glossary. The second physical tape record contains all the redundancy rules in numerical order with the individual rules separated by a double dash. The third physical record is the beginning of the lexicon proper (the lexical entries). The logical units (of the form of Figure 4 or Figure 5 in the SNOBOL linear representation of trees) are blocked ten to a physical record, with a double slash as a separator between logical units. Also, to increase processing speed, each block, as it is formed, is automatically provided with a 4-digit key representing the first two letters of the last lexical entry in the block. The key

is determined by the mapping:

A → 02

B → 03

⋮ ⋮

Z → 27

The last lexical entry is the fictitious word ZZZZZZ. Its purpose is to simplify the routine for insertion of new words into the lexicon. With ZZZZZZ as the last entry, all new words will be inserted somewhere within the lexicon, i. e., there will be no occasion to tack a new word onto the end of the lexicon. Thus, the algorithm for inserting a new word is simply: insert the new word in front of the first entry in the lexicon which is alphabetically greater than it. There is no need to add the proviso: unless the end of file is reached, in which case put the new word after the last word. If, using our procedure, we do reach the end of file, it is an indication of an error condition, and an error comment is printed out.

Facilities

The following facilities have been programmed.

GLMOD	glossary modifications
GLPNT	printout of glossary
RDMOD	redundancy rule modification
RDPNT	redundancy rule printout
LXMOD }	lexicon entry modification (provision is made for two forms of input data organization)
LXMODA }	
LXPNT	printout of all lexical entries
LXSCN	printout of lexical entries with specified feature patterns
LXSFT	printout of all different feature patterns

The programming for application of the redundancy rules is not yet completed.

The facilities listed above and the corresponding input data formats and control cards are described in detail immediately below.

GLMOD - glossary modification

This facility is used to build the glossary through the use of additions and deletions. The glossary is initialized by forming (with a special program) a tape record consisting of the characters GLOS, which are interpreted as the node contents of the root of the glossary tree (see Figure 2). Once this record is formed by the special initializing program, the remainder of the glossary can be built by using the GLMOD facility.

The input for a glossary modification consists of the following cards:

1. A control card consisting of \$GLMOD starting in column 1.
2. A sequence of packets (one or more), each packet consisting of:
 - a. A header card consisting of (-)SCn starting anywhere after column 1. The parentheses are used to indicate that the "-" sign is optional, and are not part of the input. Any number (including zero) of blanks may separate any of the input characters, but the order of characters must be as above. The header card indicates which node immediately dominated by GLOS leads to the part of the glossary to be modified by the packet it heads. If the "-" sign is present, the indicated node is deleted (along with everything it dominates). If the "-" sign is not used and the indicated

node is not present, it is created at this point in the procedure and is then available while the remaining cards in the packet are being processed.

b. One or more cards each containing one expression of the form

$$\left\{ \begin{array}{c} \text{SC} \\ (-) \left\{ \begin{array}{c} \text{SS} \\ \text{CO} \\ \text{IN} \\ \text{RU} \end{array} \right\} (-)n \end{array} \right\} \text{ /string of char (but no "(", ")", or ",")}$$

where the braces and parentheses are used according to the usual linguistic conventions used in Part I (2.2) of this report, and are not a part of the input. The string of characters following the slash is a feature specification. Column 1 must not be blank. However, blanks may be used anywhere up to the slash and will be ignored by the program. Blanks to the right of the slash will be treated as part of the input data. The characters "(", ")", and "," cannot be used as node contents because they are used by the SNOBOL system as separators in the SNOBOL linear representation of trees.

The optional minus to the left of the inner braces is used to delete an entire category type. When this minus is used, it is meaningless to have anything to the right of the inner braces since it would be deleted. If a card starts with a minus and has additional data following the symbol indicating category type, the card is not rejected as being ill-formed. Rather, the additional data is ignored. If the category type to be deleted is not present

in the portion of the glossary indicated by the header card, an error comment is printed out.

The optional minus to the right of the inner braces is used to delete the feature number (and the associated feature specification it dominates) indicated by *n*. Again, anything on the card to the right of *n* is meaningless, if *n* is preceded by minus, and is ignored by the program.

If neither minus is used, the feature number and feature specification will be added to the glossary. If the number is already in the glossary, it will not be duplicated. The new feature specification will replace the old one, or will simply be added if there was no old one.

Note that the header card only indicates or creates a node with contents *n* immediately dominated by GLOS. In order to assign a feature specification to an SC type of feature, a card of the type currently being described must be used.

To illustrate the use of GLMOD, suppose we start with the initial record GLOS and the sequence of cards:

\$GLMOD

SC 8

SC 8 / STR 1

SS 1 / STR 2

SS 4 / STR 3

IN 2 / STR 4

SC 3

CO 7 / STR 4

SS 4 / STR 5

CO 5 / STR 6

SS 2 / STR 7

SC 3 / STR 8

Then the resulting glossary is as shown in Figure 6.

Note that sibling nodes with numerical contents are ordered from left to right in increasing numerical value independently of the order in which they were entered into the glossary, but sibling nodes whose contents are feature types are ordered from left to right in the order in which they were entered into the glossary. If a feature type under a particular node is deleted and then added in again, it will appear as the rightmost branch of the particular node dominating it. The program can, if necessary, be easily modified so as to order the feature type nodes within a sibling set in some pre-determined way (for example, alphabetically), but there does not seem to be any need for it.

GLPNT - Printout of the glossary

The control card for obtaining a printout of the glossary consists of \$GLPNT starting in column 1. The current printout format is shown in Figure 7.

RDMOD - Redundancy rule modification

The redundancy rules are initialized by a special program which forms a tape record (following the glossary record) consisting of a real or fictitious redundancy rule followed by a double slash. If a fictitious rule is used for initialization, it can be replaced by a real rule later by using the RDMOD facility. Actually, all that is needed for initialization is the string RD1.//.

The input for redundancy rule modification consists of the following cards:

1. A control card consisting of \$RDMOD starting in column 1.

2. A sequence of cards, each containing one redundancy rule in the form described above in the section on Internal Data Organization, except that blanks may be inserted anywhere (to make the card more readable). This sequence of cards must be ordered according to redundancy rule number (low to high).

No distinction is made in the input between modification and addition. If a new rule has the same rule number as one of the old rules, the old rule is replaced by the new one. Otherwise, the new rule is simply inserted in its proper numerical order. The double dash separators are not included in the input--they are inserted automatically by the program.

RDPNT - Printout of redundancy rules

The control card is \$RDPNT starting in column 1. The present printout format is identical to the internal format since very little work has been done so far with the redundancy rules. If necessary, a more readable printout format will be provided in the future.

LXMOD - Lexicon entry modification - Form 1

The minimum initialization required for the lexicon is a record (the third tape record) consisting of the characters ZZZZZZ representing the fictitious last word of the lexicon. The reasons for using this fictitious word as a part of the lexicon were discussed above.

The input data format for LXMOD was determined by the manner in which the lexicographers carried out their work. Rather than code a given word all at once for all relevant features, the lexicographers preferred to consider one feature at a time and to code all relevant words with respect to that

feature. The work was done directly at a key-punch machine. For each feature, the result is a packet of cards consisting of a header card identifying the given feature, and a sequence of cards containing the words which are to be marked + with respect to that feature. However, before the packet can be used as input data, some additional information must be supplied to distinguish among homographs. For example, if nouns were being coded with respect to the feature SS3, there is nothing in the header card containing SS3 to indicate that the coding is meant to apply only to the noun form of a homographic set. In order to distinguish among homographs, an additional card is used containing more information regarding the feature content of the words which are to be marked + with respect to the feature indicated on the header card. For the above example, the additional card would specify that the word must be a noun. However, the distinction may be at a lower level, as for example, in words like bat (baseball bat, or the bat which flies), in which case the condition card might contain +NOUN and +ANIMATE (SC 1, IN 2) to select the flying bat or +NOUN and -ANIMATE (SC 1, -IN 2) to select the baseball bat.

The input cards for the LXMOD form of lexicon entry modification (for words already in the lexicon--new word additions will be described later) consist of the following:

1. A control card with \$LXMOD starting in column 1.
2. A sequence of packets, each packet consisting of:
 - a. A header card indicating the feature type and number to be added. Column 1 must be blank. The feature type symbol must consist of two characters and must occupy columns 2-3. Only one feature number is permitted, but

it can be anywhere on the card following the feature type symbol.

b. Optionally, a condition card specifying sufficient feature content to permit correct selection among homographs. Column 1 must be blank. The characters IF must appear in columns 2-3. The feature types and numbers start anywhere after column 3. If more than one feature type and number is specified, a comma is used as a separator. Blanks may appear anywhere within the series of feature types and numbers. Only one card of this type may be used (at present).

c. A sequence of word cards containing the words to be marked. Each card must start in column 1. There can be any number of words on a card, with one or more blanks as separators. A word cannot be continued from one card to the next.

3. The last packet must be followed by a card with XX in columns 1-2.

The first step in the LXMOD routine is alphabetic sorting and rearrangement of the input data. If only one packet were allowed, the need for sorting could be eliminated by requiring that the sequence of words in the input data be in alphabetic order. This, in fact, is what is done in the second form of lexical entry modifications, called LXMODA, described below. However, the purpose of the LXMOD facility is to permit the processing of several packets in one run. It would be very inefficient to run through the lexicon separately for each packet. Therefore, as the first step in the LXMOD routine, the input data is rearranged. For each word in the input data, a logical

record is formed consisting of a number indicating the word length, the word itself, the material in the header card of the packet containing the word, and the material on the condition card if one was used. All blanks are deleted. The number is separated from the word by a comma, and the condition is separated from the header by a slash. For example, given the packet

SS 18	Header Card
IF SC 20, SS 4, IN 7	Condition Card
⋮	
WORD	Word Card
⋮	

one of the logical records, after rearrangement, would be

4, WORDSS18/SC20, SS4, IN7.

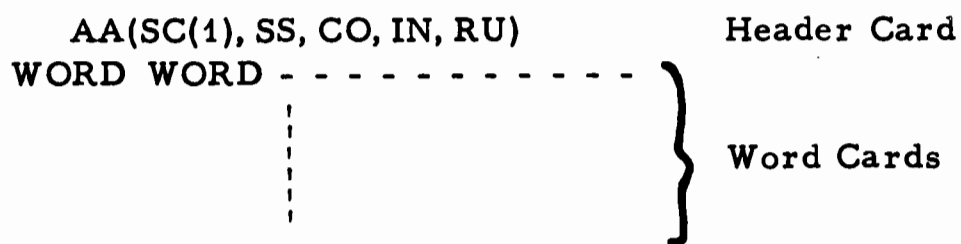
The logical records are then sorted in a field consisting of the word and (for reasons discussed below in connection with additions of new words) the two characters immediately following the word. The inclusion of the two additional characters does not affect the ordering of different word forms, but only the ordering of different occurrences of the same form. Multiple occurrences of the same form could result from the same word occurring in more than one packet, or from homographs in the same or different packets.

After the sort, the resulting list is compared with the existing lexical entries in one pass and the appropriate modifications are made.

If a word (with specified feature content if a condition is used) is not present in the lexicon, a comment to that effect is printed out. The word is not automatically inserted. This

policy was adopted in order to prevent some of the typographical errors in the input data from entering the lexicon. Similar precautions were not taken in the glossary modification procedure because the glossary is small enough so that error detection and correction involves very little work. The volume of lexical entries, on the other hand, is large enough to make error detection and correction a significant problem and should be taken into account at every stage of the processing.

New words (which are not necessarily new forms) are added by including a packet with a header card which, in place of a feature type and number, contains the characters AA followed by a tree in SNOBOL linear form. This tree becomes the part of the lexical entry dominated by the new form (see Figures 4 and 5). The tree must contain the branches SC, SS, CO, IN, RU, and as much additional coding as required to distinguish the new form from homographic forms, if present. It is advisable, although not required by the programs, to include at least the coding for the SC type of feature. To illustrate, a packet for entering new noun forms (SC 1) might consist of:



It was mentioned above that in sorting the input words, the sort field was extended to include the first two non-blank characters on the header card associated with a given word. The reason for this can be given, now that the procedure for entering new words has been described. It is desirable to permit

new words to receive additional coding, during the run in which they are first entered into the lexicon, by permitting them to appear in other packets simultaneously with their appearance in a new-word packet. But since additional coding can be applied only to words already in the lexicon, a word appearing in a new-word packet must be processed before occurrence of the same word in other packets. This sequence of processing is achieved by extending the sort field as described above and using the characters AA as the first non-blank characters (columns 2-3) on the new-word packet header card.

The LXMOD facility can also be used to print out selected lexical entries. This is done by forming a packet with a header card containing the characters ZZ in columns 2-3. The optional condition card can also be used. The use of ZZ as the symbol for selected printout ensures that printout will be the last process applied to a given word, i. e., the printout will contain all modifications which may have been applied to the given word during the run in which the word was selected for printout. The format of the printout is the SNOBOL linear representation of the subtree dominated by the word being printed out (see Figures 4, 5).

Since sorting the input words requires that all of the input be in core at once, there are space limitations on the number of words that can be handled in one run. The precise number depends on the amount of coding and conditions associated with the various words. However, the limit appears to be in the range of 500-700 words. The running time for processing 500-700 words is somewhat under 5 minutes (roughly, 1/3 of the

time for compilation, 1/3 for sorting, and 1/3 for the actual modification of the lexical entries).

LXMODA - Lexicon entry modifications - Form 2

The LXMODA facility performs the same functions as the LXMOD facility, but takes sorted input data and can therefore bypass the sorting routine. Thus, processing time is decreased, and more importantly, space limitations are eliminated. LXMODA is used for the initial loading of large word classes, and for the coding of large subclasses. Since the lexicographers normally work with alphabetized word lists, no extra work is required to obtain sorted input data.

The input format for LXMODA is the same as for LXMOD (described above) except that the control card contains \$LXMODA starting in column 1, and there can be only one packet per run.

LXPNT - Printout of all lexical entries

The control card for printout of all lexical entries contains \$LXPNT starting in column 1. The present printout format is shown in Figure 8.

LXSCN - Printout of lexical entries with specified feature patterns

The input cards are:

1. A control card containing \$LXSCN starting in column 1.
2. A card with a sequence of feature types and numbers, with a comma as a separator between successive feature-number specifications. A minus may precede a feature type. Otherwise, plus is implied. Blanks may occur anywhere.

A lexical entry is considered as satisfying the specification if it does not contradict it, i. e., the feature pattern is considered

to be a partial rather than an exhaustive description. For example, if the feature specification were SC 1, all lexical entries marked +SC 1 would be printed out regardless of the remainder of their coding. If the specifications were -SC 1, all entries except those marked +SC 1 would be printed out.

The printout format is the SNOBOL linear representation of the subtree dominated by the word being printed out (see Figures 4, 5).

LXSFT - Printout of all different feature patterns

If each feature pattern is thought of as defining a class of lexical entries, then LXSFT provides a list of all the classes and the corresponding feature patterns.

The control card is \$LXSFT starting in column 1.

The printout format is the SNOBOL linear representation of the subtree dominated by the first word in the lexicon with the given feature pattern.

Overall (Supervisory) Control

The non-modificational facilities (GLPNT, RDPNT, LXPNT, LXSCN, LXSFT) require only an input tape. The modificational facilities (GLMOD, RDMOD, LXMOD, LXMODA) require an input tape (the current lexicon) and an output tape (the modified lexicon). If only one section of the lexicon is modified, the other sections must be copied onto the output tape in the correct order. For example, if a run involves only redundancy rule modification (RDMOD), the glossary must be copied onto the output tape before the redundancy rule modification is carried out, and the lexical entries must be copied after the redundancy rule modification. Also, if both modificational and non-modificational facilities are

used in the same run, some means must be provided for selecting the version of the lexicon (old or new) to which the non-modificational facilities are to be applied. These aspects of system management are handled by the following control cards and conventions.

The input tape (the current lexicon) is mounted on channel B5 (logical unit 10).

The output tape (the modified lexicon), if used, is mounted on channel A5 (logical unit 9).

In a non-modificational run, only an input tape need be used. An output tape can be used simultaneously, if desired, to obtain a copy of the lexicon. If only an input tape is used, the only additional control card required is a card with \$NDRUN starting in column 1. This card is the final control card. Its function is to rewind and unload the tape. The remaining control cards can be in any order, but it will save some running time to carry out the processing in the order: glossary--redundancy rules--lexical entries.

In a modificational run, an additional control card is required for each segment of the lexicon which does not undergo modification. These cards are \$NDGL, \$NDRD, and \$NDLX for the glossary, the redundancy rules, and the lexical entries, respectively, and all start in column 1. The function of these cards is to copy the respective segment of the lexicon onto the output tape. If a given segment of the lexicon is modified, the presence of the corresponding \$NDXX card will have no effect, provided that it follows (not necessarily immediately) the modification card and associated data cards in the input data.

Modifications or copies must be done in the order: glossary--

redundancy rules--lexical entries. Non-modificational control cards can be placed anywhere, so long as they do not separate some other control card from its associated data cards. However, it will save running time if all of the glossary processing is done before all of the redundancy rule processing, and the lexical entry processing is done last. The final card (just as in the non-modificational case) is \$NDRUN starting in column 1. A recommended outline of input data and control card sequence is shown in Figure 9.

A run with only the following control cards:

\$NDGL
\$NDRD
\$NDLX
\$NDRUN

will yield a copy of the lexicon.

In a run involving both modificational and non-modificational facilities, the selection of the version of the lexicon to be used with a given non-modificational control card is governed by the convention that the most recent version of the relevant segment of the lexicon is used. Thus, if at the time the non-modificational control card is processed, the relevant segment of the lexicon has undergone modification, the modified version is used. Otherwise, the unmodified version is used. For example, if it is desired to print out the current glossary, then modify it and print out the modified version, the following sequence of cards is used:

\$GLPNT

\$GLMOD

|
|
|
|
|

}
|
|
|
|
|

input data for glossary
modification

\$GLPNT

|
|
|
|
|

}
|
|
|
|
|

additional processing

If no additional processing is desired, the second \$GLPNT would be followed by the sequence:

\$NDGL

\$NDRD

\$NDLX

\$NDRUN

The second \$GLPNT card could be placed anywhere before the \$NDRUN card, but the above sequence is preferable because it keeps the rewinding of the tape to a minimum.

Reference

1. N. Chomsky, "Aspects of the Theory of Syntax", The M.I.T. Press, Cambridge, Massachusetts, 1965.

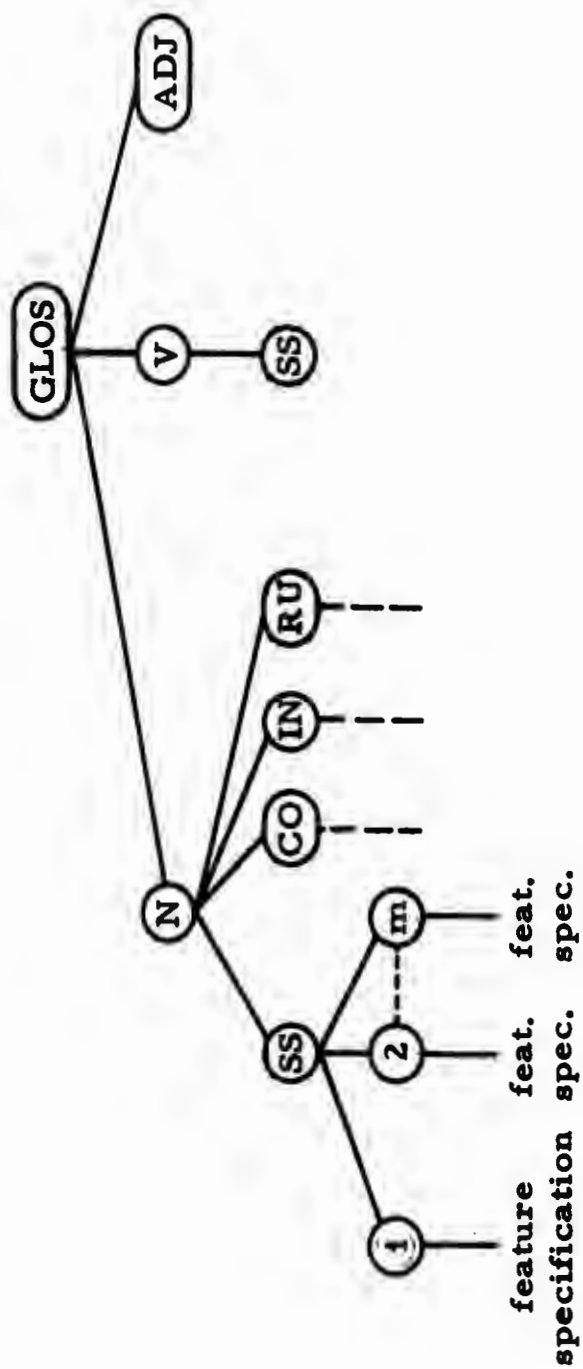


Figure 1. A possible glossary format with the SC type of feature in a special position.

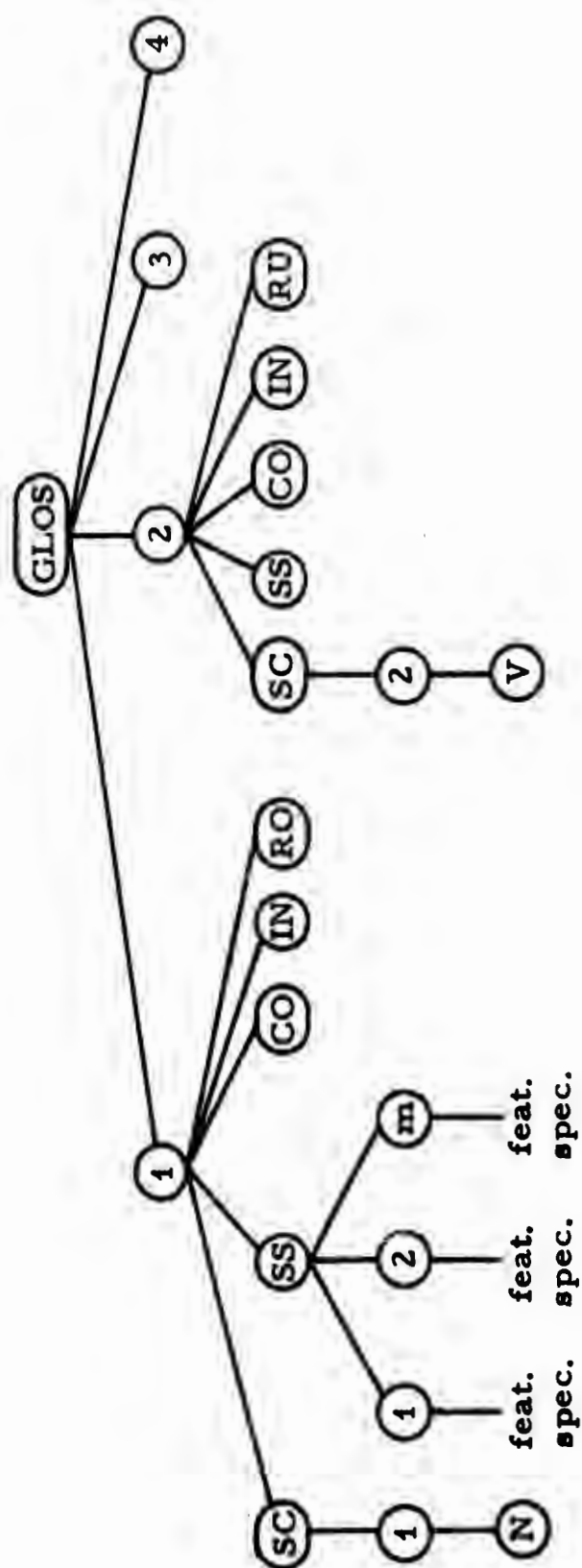


Figure 2. The glossary format actually used, with all feature types represented uniformly.

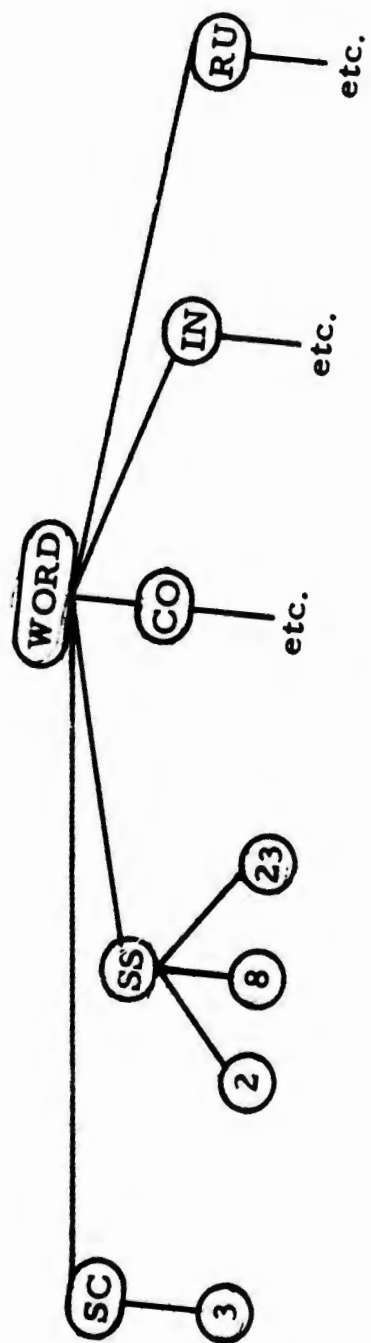


Figure 3. A possible lexical entry format.

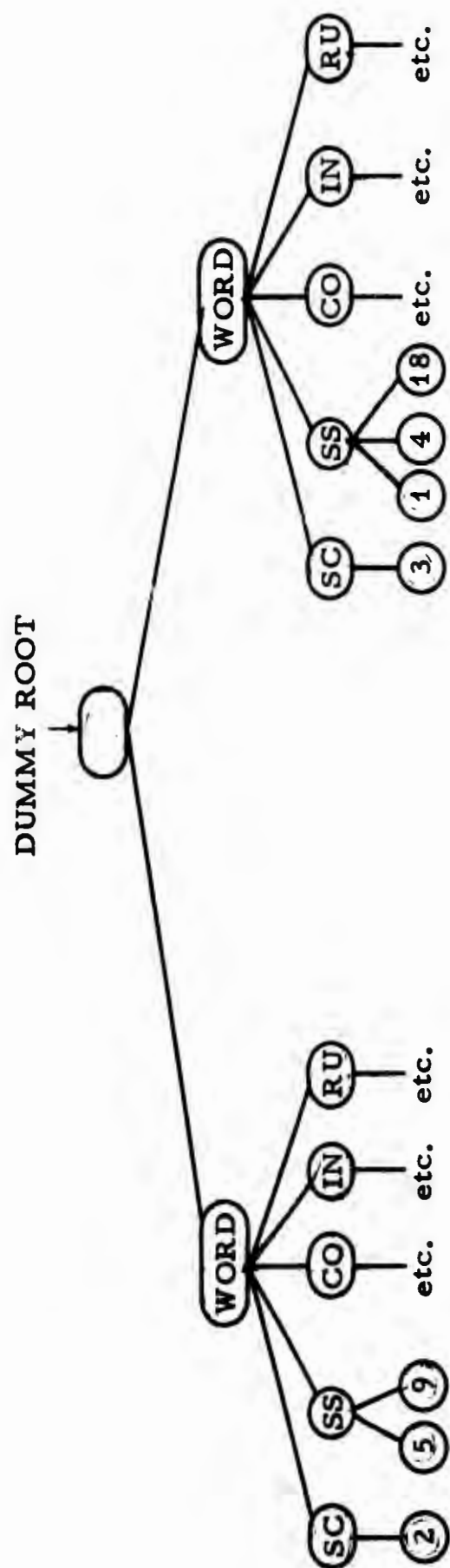


Figure 4. The lexical entry format actually used, illustrating the representation of homographic entries.

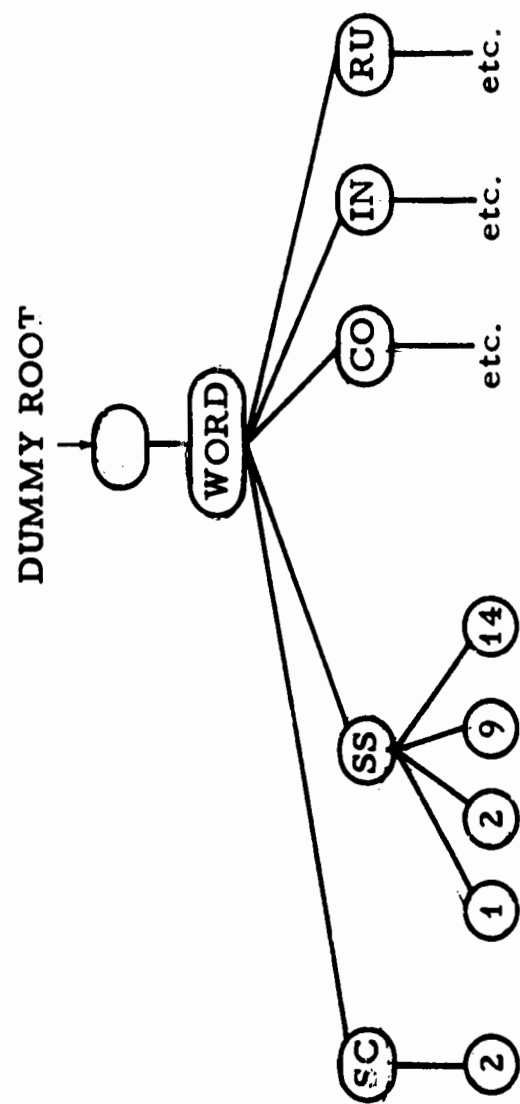


Figure 5. The lexical entry format actually used, illustrating the representation of non-homographic entries.

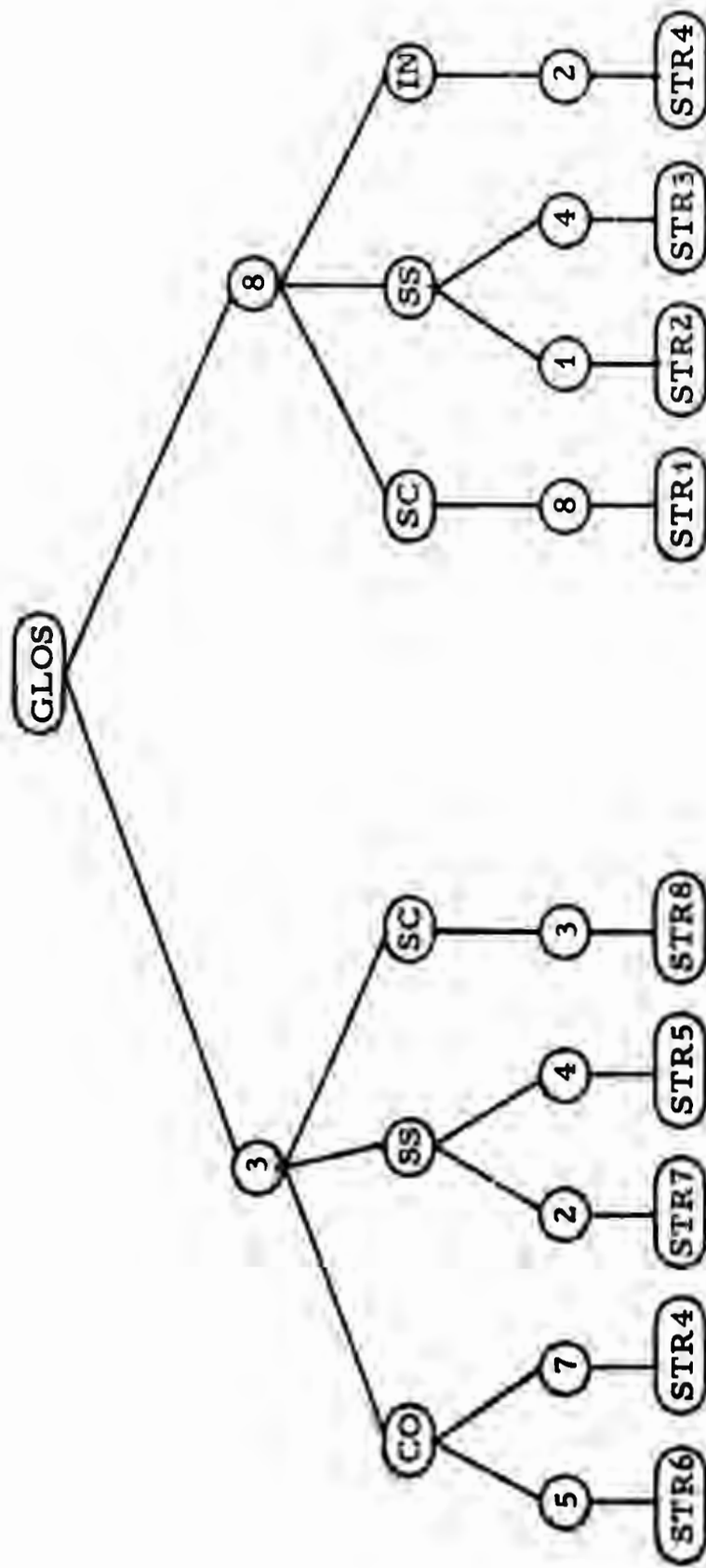


Figure 6

SC 1	NOUN/	
	IN 1	=HUMAN= /
	IN 2	=ANIMATE= /
	IN 3	=ABSTRACT= /
	IN 4	=PROPER= /
	SS 1	DET/ ***/
	SS 2	DET/ ***/S/
	SS 3	*** /S/
	SS 4	*** /
SC 2	VERB/	
	SS 1	*** /NP/
	SS 2	*** /S/
	SS 3	*** /NP/S/
	SS 4	*** /PP/

Figure 7. Glossary Printout Format

ADAM /SC/ 1 /SS/ /CO/ /IN/ 1 /RU/
ADDITION /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADDRESS /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADJUSTMENT /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADMINISTRATION /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADMIRAL /SC/ 1 /SS/ /CO/ /IN/ 1 /RU/
ADMIRATION /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADMISSION /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADOPTION /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADULT /SC/ 1 /SS/ /CO/ /IN/ 1 /RU/
ADVANTAGE /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADVENTURE /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADVERTISEMENT /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADVERTISING /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADVICE /SC/ 1 /SS/ /CO/ /IN/ /RU/
ADVISER /SC/ 1 /SS/ /CO/ /IN/ 1 /RU/
ADVOCATE /SC/ 1 /SS/ /CO/ /IN/ 1 /RU/
AEROPLANE /SC/ 1 /SS/ /CO/ /IN/ /RU/

Figure 8. Lexical entry printout format.

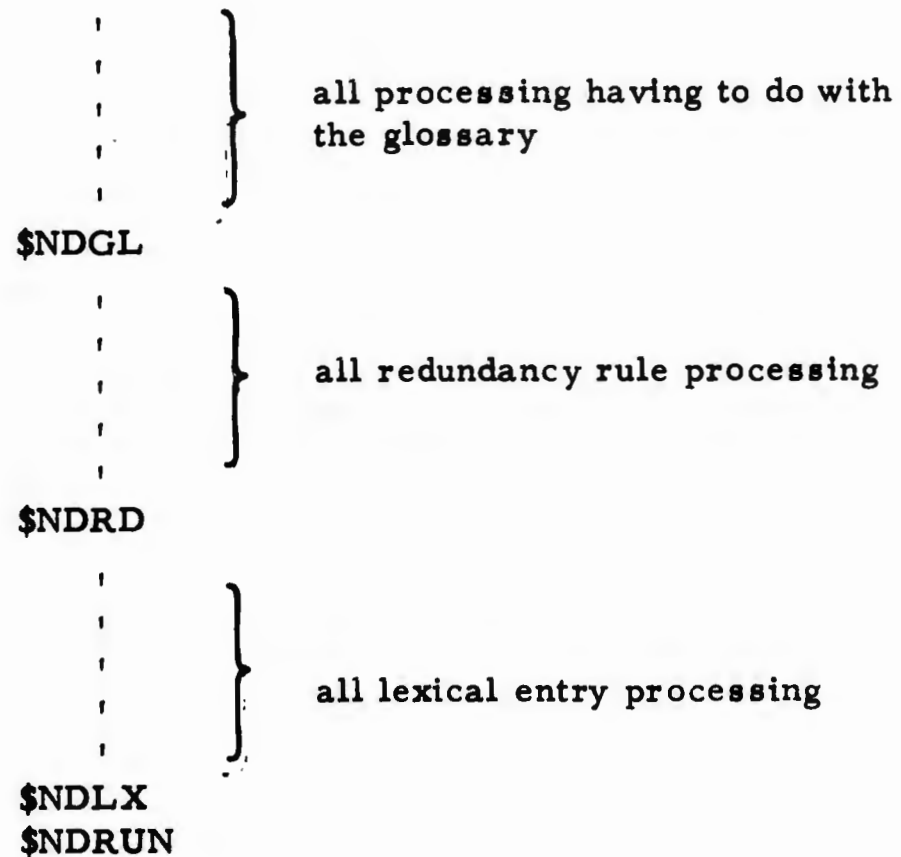


Figure 9. Outline of recommended input data and control card sequence for a modificational run.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)		
1. ORIGINATING ACTIVITY (Corporate author) IBM- T. J. Watson Research Center Yorktown Heights, New York 10598		2a. REPORT SECURITY CLASSIFICATION Unclassified
		2b. GROUP 1
3. REPORT TITLE "Specification and Utilization of a Transformational Grammar."		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Scientific Report		
5. AUTHOR(S) (Last name, first name, initial) Lieberman, David V. et al		
6. REPORT DATE April 1966	7a. TOTAL NO. OF PAGES 308	7b. NO. OF REFS 12
8a. CONTRACT OR GRANT NO. AF 19(628)-5127	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.		
10. AVAILABILITY/LIMITATION NOTICES		
11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Electronic Systems Division Air Force Systems Command - USAF Laurence G. Hanscom Field, Mass.	
13. ABSTRACT Scientific Report No. 1 contains four (4) parts: Part I - <u>The IBM Core Grammar of English.</u> Our current grammar of English is presented in full, and numerous derivations are carried out in detail to illustrate the current generative power of the grammar. Part II - <u>Design of a Grammar Tester.</u> The design considerations on which the present version of the tester was based are discussed, and a set of tentative input, output and control formats are presented. Part III - <u>Programming for the Grammar Tester.</u> A LISP implementation of the grammar tester is presented. The overall flow of control and the various special functions are described. Part IV - <u>Computer Support for Lexicon Development.</u> A program package (programmed in SNOBOL) to facilitate the compilation, modification, scanning, etc., of the lexicon is described. (D. V. Lieberman)		

DD FORM 1473
1 JAN 64

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Language Processing						
Linguistics						
Grammar						
Syntax						
Lexicography						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

- (1) "Qualified requesters may obtain copies of this report from DDC."
- (2) "Foreign announcement and dissemination of this report by DDC is not authorized."
- (3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."
- (4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."
- (5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (TS), (S), (C), or (U).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

IBM - Research Division
P. O. Box 218
Yorktown Heights, New York 10598

THOMAS J. WATSON
RESEARCH CENTER

July 22, 1966

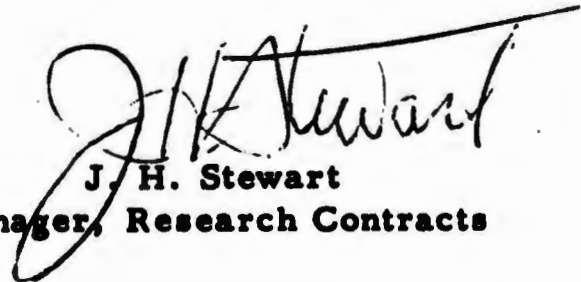
MEMORANDUM TO: Distribution List

SUBJECT: Scientific Report No. 1 - Interim
entitled, "Specification and
Utilization of a Transformational
Grammar," by D. Lieberman

ENCLOSURE 1: Form DD 1473

Copies of the subject Scientific Report, prepared for the Air Force Cambridge Research Laboratories, Office of Aerospace Research, United States Air Force, under Contract AF 19(628)-5127, which was recently disseminated, contained a Form DD 1473 with incomplete information.

Accordingly, you are requested to substitute the enclosed Form DD 1473 for the one contained in the subject Scientific Report.


J. H. Stewart
Manager, Research Contracts

JHS/et
Enclosure

AD 635-520

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)		
1 ORIGINATING ACTIVITY (Corporate author)		2a. REPORT SECURITY CLASSIFICATION
IBM - T. J. Watson Research Center Yorktown Heights, New York 10598		Unclassified
		2b. GROUP
3 REPORT TITLE		
"Specification and Utilization of a Transformational Grammar."		
4 DESCRIPTIVE NOTES (Type of report and inclusive dates)		
Scientific Interim Report		
5 AUTHOR(S) (Last name, first name, initial)		
Lieberman, David V. et al		
6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1966	308	12
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)	
AF19(628)-5127		
b. PROJECT NO.		
4641, 02		
c. DOD Element	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
62405304		
d. DOD Subelement	AFCRL-66-270	
674641		
10 AVAILABILITY/LIMITATION NOTICES		
Distribution of this document is unlimited.		
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY
		Hq. AFCRL, OAR (CRB) United States Air Force L. G. Hanscom Field, Bedford, Mass.
13 ABSTRACT		
Scientific Report No. 1 contains four (4) parts: Part I - <u>The IBM Core Grammar of English.</u> Our current grammar of English is presented in full, and numerous derivations are carried out in detail to illustrate the current generative power of the grammar. Part II - <u>Design of a Grammar Tester.</u> The design considerations on which the present version of the tester was based are discussed, and a set of tentative input, output and control formats are presented. Part III - <u>Programming for the Grammar Tester.</u> An LISP implementation of the grammar tester is presented. The overall flow of control and the various special functions are described. Part IV - <u>Computer Support for Lexicon Development.</u> A program package (programmed in SNOBOL) to facilitate the compilation, modification, scanning, etc., of the lexicon is described. (D. V. Lieberman)		

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Language Processing						
Linguistics						
Grammar						
Syntax						
Lexicography						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those

imposed by security classification, using standard statements such as:

- (1) "Qualified requesters may obtain copies of this report from DDC."
- (2) "Foreign announcement and dissemination of this report by DDC is not authorized."
- (3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."
- (4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."
- (5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (TS), (S), (C), or (U).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Unclassified

Security Classification