CS - 95
AF - 24

A FORMAL SYNTAX FOR TRANSFORMATIONAL GRAMMAR

BY

JOYCE FRIEDMAN and ROBERT W. DORAN

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

STANFORD UNIVERSITY COMPUTER SCIENCE DEPARTMENT

COMPUTATIONAL LINGUISTICS PROJECT

MARCH 1968

A FORMAL SYNTAX FOR TRANSFORMATIONAL GRAMMAR

by

Joyce Friedman and Robert W. Doran*

This research was supported in part by the United States Air Force Electronic Systems Division, under Contract F196828-C-0035.

*Present address: Mathematics Department, University of Bristol, Bristol, England.

CS - 95
AF - 24

February 1968

A FORMAL SYNTAX FOR TRANSFORMATIONAL GRAMMAR

by

Joyce Friedman and Robert W. Doran

Abstract

A formal definition of the syntax of a transformational grammar is given using a modified Backus Naur Form as the metalanguage. Syntax constraints and interpretation are added in English. The underlying model is that presented by Chomsky in Aspects of the Theory of Syntax. Definitions are given for the basic concepts of tree, analysis, restriction, complex symbol, and structural change, as well as for the major components of a transformational grammar, phrase structure, lexicon, and transformations. The syntax was developed as a specification of input formats for the computer system for transformational grammar described in [24]. It includes as a subcase a fairly standard treatment of transformational grammar, but has been generalized in many respects.

Remark

A major purpose of formalization is to provide explicit subject matter for discussion. Any comments on the material here will be gratefully received. A revised version will be submitted for publication.

CONTENTS

INTRODUCTION

In Syntactic Structures[3], Noam Chomsky writes: "We can determine the adequacy of a linguistic theory by developing rigorously and precisely the form of grammar corresponding to the set of levels contained within this theory and then investigating the possibility of constructing simple and revealing grammars of this form for natural languages." While the linguistic theory of transformational grammar has developed and changed rapidly since this was written, the criterion is still relevant.

In this paper we address ourselves to the first part of the requirement, that is, we develop rigorously and precisely a form in which the syntax of natural language can be described by the transformational model. We do this in a way which we hope will make it easier for linguists to construct and examine grammars of natural language.

The linguistic theory which we are modeling is transformational generative grammar of the syntax of natural language, basically as presented in Chomsky's Aspects of the Theory of Syntax [4]. We have also taken into consideration recent linguistic work which is based on Aspects and which applies or extends the model.

In the development of the syntax it was decided to be inclusive and general, rather than to try to limit the power of the syntax to the exact amount likely to be required. We have tried to make the syntax powerful enough so that it will not require augmentation by special devices; as a result, many linguists may find the syntax too general for their purposes. However, the metalanguage offers a relatively easy way to define a suitable and clean subset.

The syntax is described in a metalanguage which is a modification of Backus Naur Form (BNF), which is widely used in the description of programming languages. BNF is a formalism for presenting a context-free language. In the first section we describe in detail the modification of BNF used throughout the rest of the paper.

Following the description of the metalanguage, we proceed immediately to the formal presentation of transformational grammar. We give formats for transformational grammar; for the basic concepts of tree, analysis, restriction, complex symbol and structural change; and then for the major components, phrase structure, lexicon and transformations. For each format, we specify precise interpretations and give examples.

For the parts of transformational grammar which are relatively standard and well-understood, the presentation is complete. However, in one case where the work is frankly experimental (the control program for transformations), we have referred to other papers for more details.

The formal definition of the syntax of transformational grammar is part of the development of a computer system for transformational grammar which we describe in [24]. A prior reading of the system description is recommended, since it provides a wider context for this paper and also points out some of the novel features of the syntax. The system is being implemented on the 360/67 computer so that it can be used to aid in writing and investigating transformational grammars. While the present paper may be considered simply as a definition of a syntax for transformational grammar, it also defines the format in which a grammar can be read into the computer system.

This paper is a formal presentation of the syntax of a transformational grammar; it is not intended to be read as an introduction to the theory. The reader is assumed to be familiar with <u>Aspects</u>.

METALANGUAGE

The syntax is described in a modification of Backus Naur Form (BNF) [1, 14], with syntax constraints and interpretation (semantics) added in English. BNF is familiar to computer scientists as the metalanguage used in the description of Algol 60. As we will use the symbols |, ⟨ and ⟩ in transformational grammars, we modify the usual BNF by replacing angular brackets by underlining, e.g., "transformation" rather than "⟨transformation⟩", and using "or" in place of "|".

For linguists unfamiliar with BNF, it should suffice to say that

1. the modified-BNF production " A : := B C or D or E " expresses the context-free rewriting rule " $A \rightarrow \left\{ \begin{array}{c} B\ C \\ D \\ E \end{array} \right\}$ ",
2. the nonterminal symbols of modified-BNF are denoted by the underlined name of the constructs, viz.,

transformational grammar : := phrase structure lexicon transformations $END

3. symbols not underlined are used autonymously, (e.g., " $END "),

4. juxtaposition in the object language is indicated by juxtaposition in the metalanguage.

We refer to the constructs of the metalanguage as "formats", because they are in fact the free-field formats of a computer system.[1] We have carried the underlining of format names into the text of the paper.

[1] The character set of the object language is restricted to that of the IBM 029 keypunch. Thus, since the set lacks square brackets, we use them only in the metalanguage, and use only angular brackets in the object language.

Basic to the syntax are the two formats word and integer. A word is a contiguous string of alphanumeric characters beginning with a letter; an integer is a contiguous string of digits. Except in these two formats, spaces may be used freely.

If a BNF description is to elucidate a language, it should not introduce names for intermediate formats which do not have meaning. In order to avoid additional formats where possible, and to simplify the description, we have introduced into the metalanguage six operators. In each case the operand is given within square brackets following the operator. The operators are:[1, 2]

---

[1] In the LISP 1.5 documentation[13] a similarly modified BNF is used. There are notations corresponding to the first three operators above, as follows:

list[integer]	integer ... integer
clist[integer]	integer, ..., integer
opt[integer]	(integer)

[2] Modified BNF includes BNF. It is (weakly) equivalent to BNF, although it does not give the same structure. It is easy to show that each occurrence of an operator can be deleted, possibly by the introduction of one or more intermediate formats.

1. list[b] can be replaced by the new format bl, where bl is defined by bl ::= b or bl b .
2. clist[b] can be replaced by the new format bl, where bl is defined by bl ::= b or bl , b .
3. sclist[b] is similar to clist[b] .
4. opt[b] can be removed by replacing any string α opt [b] β by α b β or α β (for any strings α, β in the metalanguage).
5. booleancombination[b] can be replaced by bl, where bl ::= b or bl | bl or bl ∧ bl or ¬bl or (bl) if we are not concerned with obtaining structure, or, if we are, by Boolean expression defined as in Algol 60:

   Boolean primary ::= b or (Boolean expression)
   Boolean secondary ::= Boolean primary or ¬ Boolean primary
   Boolean factor ::= Boolean secondary or
   Boolean factor ∧ Boolean secondary
   Boolean term ::= Boolean factor or
   Boolean term | Boolean factor
   Boolean expression ::= Boolean term
6. choicestructure[b] can be replaced by bl, where bl ::= b or (clist[bl]), and clist[bl] is then replaced as above.

1. list

a ::= list[integer]

would allow a to be

1 2 6 9171 3 20

A list may be of length 1 but may not be empty.

2. clist (comma list)

a ::= clist[integer]

allows a to be

1, 2, 6, 9171, 3, 20

A clist may be of length 1 but may not be empty.

3. sclist (semicolon list)

a ::= sclist[word]

allows a to be

CAT; DOG1 ; MOUSE

A sclist may be of length 1 but may not be empty.

4. opt (option)

a ::= opt[integer] word

allows a to be either

3 NP or NP

5. booleancombination

a ::= booleancombination[word]

would allow a to be

A & (B | ¬ D & C)

The logical operators ¬, &, | (not, and, or) are allowed in a boolean combination. Parentheses may be used to override the precedence order. The precedence order of the operators is: ¬ is stronger than & is stronger than | .

6. choicestructure

rule right ::= choicestructure[list[word]]

would allow rule right (the right-hand side of a phrase structure rule) to be

B (C)((D E,F))

The choicestructure operator is used to represent choices and options in the object language. An entity within parentheses represents an option; two or more entities within parentheses and separated by commas represent a choice. The expression above could be used as part of an object language rule A = B (C)((D E,F)) to abbreviate the six subrules A = B, A = B C, A = B D E, A = B F, A = B C D E and A = B C F . The usual linguistic representation for this rule is

$$A \rightarrow B\ (\ C\ )\ (\left\{\begin{matrix} D\ E \\ F \end{matrix}\right\})\ .$$

Syntax constraints. In various places below, the syntax description is augmented by syntax constraints. In almost all cases, the use of constraints could have been avoided. However, where the choice appeared to be between a simple constraint and an alternative introduction of several intermediate formats, we have felt that clarity was best served by the use of the constraint.

Transformational Grammar

The model of transformational grammar we use is a version of the one presented by Noam Chomsky in Aspects of the Theory of Syntax [4]. The components of the grammar are a set of phrase structure rules, a lexicon, and a set of transformations. The phrase structure rules are used to generate base trees, the lexicon to insert the vocabulary words and complex symbols into a completed base tree, and the transformations to transform the base tree into a surface tree with terminal symbols which represent a sentence of the language. Since the remainder of the paper is essentially an expansion of this brief description, we proceed immediately to the first rule of the syntax:

0.01

transformational grammar ::= phrase structure lexicon transformations $END

Basic to the formats for the three components are trees, analyses, restrictions, complex symbols, and structural change. We will define these basic concepts first (in terms of one another) and then give the definitions of the components.

For each set of formats the presentation follows a fixed order: syntax description in the metalanguage, syntax constraints in English, examples of the syntax, and interpretation of the formats. Remarks are interspersed between these sections.

Appendix I contains an example of a transformational grammar. Appendix II is a listing of the full syntax.

BASIC FORMATS

1. Tree

Syntax

1.01 tree specification ::= tree opt[, clist[word tree]] .

1.02 tree ::= node opt[complex symbol] opt[⟨ list[tree] ⟩]

1.03 node ::= word or sentence symbol or boundary symbol

1.04 sentence symbol ::= S

1.05 boundary symbol ::= #

Constraint*

1.04 The sentence symbol is distinguished. It may not be used as a word.

Interpretation*

1.01 The tree specification $tree_0$ $word_1$ $tree_1$ $word_2$ $tree_2$ ... $word_n$ $tree_n$ is interpreted to mean that the occurrence of $word_1$ in $tree_0$ is to be replaced by $tree_1$. The process continues, always applying to the result of the previous steps.

1.02 A complex symbol following a node is attached to that node as a list of properties. A list of trees within brackets following a node is the (left-to-right) list of the daughter sub-trees of the node.

Examples*

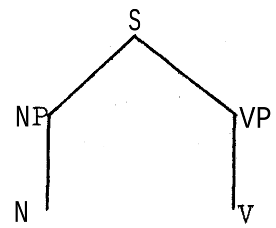
1.01 tree specification

---

*Constraints interpretations, and examples are numbered to correspond to the syntax.

S⟨S1S2⟩, S1 NP ⟨N⟩, S2 VP (A), A V .

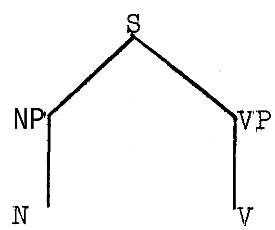
represents the tree



1.02 <u>tree</u>

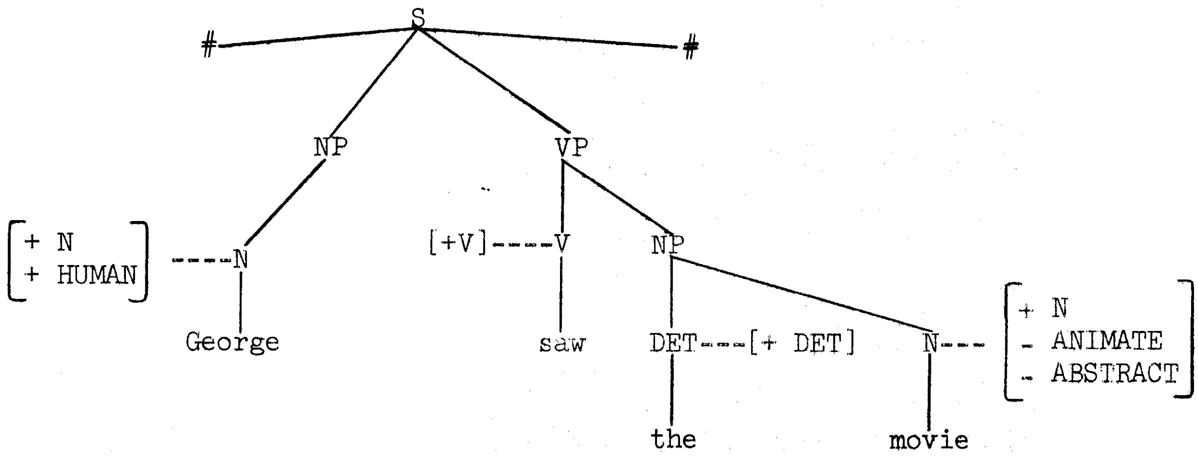
S ⟨NP ⟨N⟩ VP ⟨V⟩⟩

This represents the same <u>tree</u> given above:



<u>tree</u>

S ⟨# NP ⟨N | + N + HUMAN | ⟨GEORGE⟩⟩ VP ⟨V | +V | (SAW)
NP ⟨DET | + DET | (THE) N | + N -ANIMATE - ABSTRACT |
⟨MOVIE⟩⟩ # ⟩



Note that a complex symbol following an element is attached to that element as shown in the diagram. The vocabulary word GEORGE is treated as a daughter node of the category symbol N, while the complex symbol |+ N + HUMAN| is attached to N rather than to GEORGE . The alternatives would have been to attach the complex symbol directly to the vocabulary word or to include the vocabulary word as part of the complex symbol. The advantages to be gained from our treatment are first that it allows complex symbols to be attached to any node of the tree, and second that it makes the vocabulary words into node names which are then available for mention in a transformation.

Reference

Formats for trees are discussed further in [31] where a fixed-field format is also given.

2. Analysis

Syntax

2.01 analysis ::= list[opt[integer] term]

2.02 term ::= structure or skip or choice

2.03 structure ::= element opt[complex symbol]

opt[opt[¬] opt[/]⟨ analysis ⟩]

2.04 element ::= node or * or __

2.05 choice ::= (clist[analysis])

2.06 skip ::= % opt[opt[¬] opt[&] ⟨ clist[structure] ⟩]

Constraints

2.01 In the implementation, integers in an analysis must be greater than 0 and less than or equal to 50 .

2.02 Two adjacent terms of an analysis may not both be skips.

2.04 The element __ may appear only in an analysis in a contextual feature (4.8) and must appear there precisely once.

2.05 Each analysis in the clist[analysis] of a choice must contain at least one term which is not a skip.

Alternative

2.06a skip ::= % opt[opt[¬]⟨ structure ⟩]

At the present time 2.06a rather than 2.06 has been implemented.

Examples

2.01 analysis

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

% N ⟨3 *|*C|⟩ S⟨4 NP %⟩ %

S⟨# NP ⟨% N|+ ABSTRACT|⟩ VP⟨__%⟩ # ⟩

2.03 structure

N ⟨3 *|*C| ⟩

N |+ ABSTRACT|

VP⟨ _% ⟩

#

2 . 0 5 choice

(PREP)

(BE, HAVE)

2.06 skip

% ¬ ⟨A, B⟩

% ¬ & ⟨C, D⟩

Remark

A tree (1.02) is a special case of an analysis in which none of the symbols ( ) ¬ % & nor integers occur.

Interpretation

An analysis specifies a template against which a tree may be matched. If the match succeeds, the tree is said to "satisfy" the analysis. The match may succeed in more than one way--the order in which the matches are taken is specified by the analysis algorithm (see [28]).

An analysis matches a tree as follows: Each term in the analysis matches a section of the tree. All leaves of the tree are part of some term's match. Left-to-right order in the analysis corresponds to left-to-right order in the tree.

2.01 The integers in an analysis are labels for the terms which follow and are used in restrictions and structural changes to refer to the subtrees defined by those terms. An integer should not normally be used more than once in an analysis, unless at most one of the terms so labeled will be matched, viz. A (1 B, 1 C) D . This is equivalent to A 1 (B, C) D .

2.03 A structure matches a subtree if:

A. the element is a node which is the name of the top node of the subtree,

the element is * ,

the element is __ and the top node of the subtree is the location at which lexical insertion is currently being attempted.

B. the complex symbol matches the complex symbol of the subtree. (For analyses in contextual features the test used is nondistinctness. For analyses in the structural description indusion appears to be the appropriate test.)

C. all restrictions referring to the integer (and to an integer preceding a choice in which this structure is the first term of an analysis, etc.) are met.

There is an ⟨ analysis ⟩ on this structure and:

i. it is /⟨ and the subtree matches the analysis

ii. it is ¬/⟨ and the subtree cannot match the analysis

iii. it is ⟨ and the subtree matches the analysis, with the further requirement that each sub-subtree matched by a structure not inside further ⟨ ⟩'s be headed by an immediate daughter of this subtree's head.

iv. it is ¬⟨ and no type-iii match can be found.

2.04 The element * is an unspecified single node, which will match any one node in the tree. The is also an unspecified single node, and defines the location for lexical insertion in a contextual feature.

2.05 A choice matches a part of the tree if at least one analysis in its clist matches. If it has only one analysis, it also matches a null part of the tree.

2.06 A skip replaces the variable nodes in the more usual treatment.

2.06a A skip matches a region bounded by two subtrees if:

A. all restrictions are met [see C. under structure],

B. there is a ⟨ structure ⟩ and

i. it is ⟨ and there is a matching subtree somewhere in this region,

ii. it is ¬⟨ and there is no matching subtree in the region.

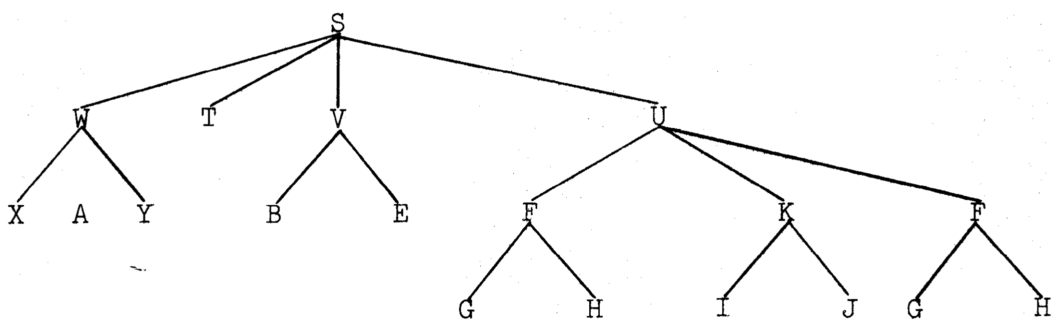
Note: An analysis cannot succeed with two adjacent skips; a skip must be bounded on both left and right by structures or by edges of (sub)trees.

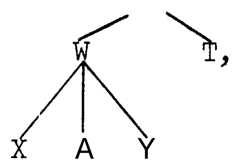
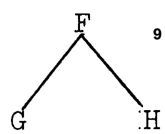
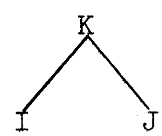
In a skip the clist[structure] is a list of subtrees to be matched with the skip. If the skip is simply %⟨ clist[structure] ⟩ then at least one of the structures must be matched. If % ¬ ⟨ clist[structure] ⟩ then none of them must be matched. If

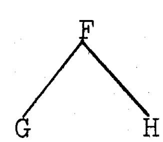
% & ⟨ clist[<u>structure</u>] ⟩ all must be matched, and, finally, if % ¬ & ⟨ clist[<u>structure</u>] ⟩ then at least one must fail to match.

<u>Examples</u>

An analysis % ⟨A⟩ B (C) (D,E) 2F⟨G H⟩(% ¬ ⟨I⟩ J,K)39F with restriction RES 2 EQ 39. matches the tree.



% ⟨A⟩ matches the region  since this includes a node A . B matches the subtree B . (C) matches a null portion of the tree. (D,E) matches the subtree E . 2F⟨G H⟩ matches the subtree  , since G matches subtree G and H matches subtree H . (% ¬ ⟨I⟩ J, K) matches the subtree  .

Note that only K matches; if % ¬ ⟨I⟩ J matches, I would have to match subtree J, leaving region I for % ¬ ⟨I⟩, which would fail because I matches I . 39F matches the subtree  since this and the subtree 2 are equal.

Reference

Analyses and the analysis algorithm will be discussed further in a forthcoming paper by Friedman and Martner[28].

3. Restriction

Syntax

3.01 restriction ::= booleancombination[condition]

3.02 condition ::= unary condition or binary condition

3.03 unary condition ::= unary relation integer

3.04 binary condition ::= integer binary tree relation node designator or integer binary complex relation complex symbol designator

3.05 node designator ::= integer or node

3.06 complex symbol designator ::= complex symbol or integer

3.07 unary relation ::= TRM or NTRM or NUL or NNUL or NONREP

3.08 binary tree relation ::= EQ or NEQ or DOM or NDOM

3.09 binary complex relation ::= INC1 or NINC1 or INC2 or NINC2 or CSEQ or NCSEQ or NDST or NNDST

Constraints

3.05 For the binary conditions with EQ and NEQ, the subtree designator must be an integer. For DOM and NDOM the subtree designator must be a node.

Examples

3.01 RES 1 EQ 2 | 3 NEQ 5 .

3002 ¬ 1 EQ 2 & 3 NEQ 5

3.03 NUL 2

3.07 3 INC1 | + HUMAN |

Remark

The conditions listed above are examples; the list can easily be expanded.

Interpretation

Restrictions are tested during an attempt to match an analysis.

Where both NXXX and XXX are relations, NXXX is the negation of XXX . It is generally more efficient to use A NXXX B rather than ¬ A XXX B, but the result is identical.

Integers in conditions refer to nodes of the tree which match the correspondingly numbered terms of the analysis. Integers in subtree designators refer to the subtree headed by the numbered term,

The meanings of the conditions listed are as follows:

3.03 unary relations

TRM integer means that the subtree corresponding to integer consists of a single (terminal) node.

NUL integer means that the label integer is unassigned (null).

NONREP integer means that the subtree corresponding to integer must not be the same for two applications in a set of applications of the analysis. This is appropriate to transformations with the parameter C or CNR .

3.08 binary tree relations

integer EQ integer means that the subtrees corresponding to the two integers are equal and have nondistinct complex symbols.

integer DOM node means that the subtree corresponding to integer contains at least one occurrence of node. Note that A / ⟨ % B % ⟩ is exactly equivalent to 1 A RES 1 DOM B . The former is generally to be preferred.

3.09 binary complex relations

An integer as an argument of a binary complex relation refers to the complex symbol of the node matching the corresponding labeled term of an analysis.

Each of the binary complex relations is defined by a matrix which gives the result of a comparison of two feature specifications. The relation will be true for two complex symbols if and only if it is true for all of their feature specifications. The matrices are given in the Table below.

integer INC1 complex symbol designator

The complex symbol B pointed to by the integer on the left includes-1 the complex symbol A pointed to by the complex symbol designator on the right. That is, every feature specification of A also occurs in B . It is false only if some feature for which A has a specification is absent from B, or if some feature occurs in one with the value + and in the other with the value - .

integer INC2 complex symbol designator

The complex symbol B pointed to by the integer on the left includes-2 the complex symbol A pointed to by the complex symbol designator on the right. Includes-2 differs from includes-1 only in the case of the value * .

Includes-2 will be false if A has * where B has + or -, or if B has * where A has + or - .

integer CSEQ complex symbol designator

The relation CSEQ holds between two complex symbols if and only if their feature specifications are identical.

integer NDST complex symbol designator

Two complex symbols are nondistinct (NDST) unless there is a feature for which one has the value + and the other has and the other has the value - .

TABLE

Matrices defining binary complex relations

EQUALITY

A \ B	+	-	*	abs
+	T	F	F	F
	F	T	F	F
*	F	F	T	F
abs	F	F	F	T

NONDISTINCTNESS

A \ B	+	-	*	abs
+	T	F	T	T
	F	T	T	T
*	T	T	T	T
abs	T	T	T	T

INCLUSION -1

A \ B	+	-	*	abs
+	T	F	T	F
	F	T	T	F
*	T	T	T	F
abs	T	T	T	T

INCLUSION -2

A \ B	+	-	*	abs
+	T	F	F	F
-	F	T	F	F
*	F	F	T	F
abs	T	T	T	T

(a Note follows on next page)

Note: T represents true, F represents false, and abs indicates that the feature is absent altogether. For the test to be true for complex symbols it must be true for all of their feature specifications.

Reference

Restrictions are discussed further in Pollack[34].

4. Complex symbol

Syntax

4.01 complex symbol ::= list[feature specification]

4.02 feature specification ::= value feature

4.03 feature ::= catogory feature or inherent feature or rule feature or contextual feature identifier

4.04 category feature ::= category

4.05 category ::= word

4.06 inherent feature ::= word

4.07 rule feature ::= transformation name

4.08 contextual feature identifier ::= contextual feature or contextual feature label

(See 7.06, 7.07 for contextual feature label)

4.09 contextual feature ::= ⟨ analysis ⟩ opt[restrictions]

4.10 value ::= + or - or *

Constraints

4.02 A feature may appear only once in a complex symbol.

4.03 No more than one category feature may appear in a complex symbol.

4.05, 4.06, 4.07, 4.09 To avoid ambiguity, each word should have only one immediate parse.

4.08 The analysis in a contextual feature is restricted by constraints 3.04 and 3.01 above.

Alternatives

4.10a value ::= + or - or * or value word

4.10b value ::= + or - or value word

Remarks

4.02 The use of "feature specification" to denote a signed feature is introduced in Chomsky [4].

4.10 The value * was suggested in the UCLA Working Papers [20], where it is used to mean "obligatory specification".

4.10b is used by Gross [8].

Interpretation

Complex symbols appear in the lexicon, in base trees during and after the lexical insertion process, and in analyses and restrictions. Their use in lexical insertion will be described in Friedman and Bredt [26]. Their role in analysis is described briefly in section VI below and in more detail in Friedman and Martner [28]. Complex symbols are implicitly expanded by the redundancy rules of the lexicon.

4.01 A complex symbol is an unordered list of properties or feature specifications.

4.08 A complex symbol appearing in a tree may not contain contextual features.

4.10 The value * is defined by its use in the binary complex relations (3.10) and operations (5.10). It can be regarded as $\pm$; it is nondistinct from both + and - .

Examples

4.01 complex symbols

|+ N + HUMAN - COMMON|

|+ V - TRANS + ANIMSUBJ|

4.02 feature specifications

+ HUMAN

+ ANIMSUBJ

4.06 inherent feature

HUMAN

4.07 rule feature

~PASSIVE

4.08 contextual feature identifiers

ANIMSUBJ

S ⟨ # NP ⟨ % N | + ANIMATE | ⟩ VP ⟨ __ % ⟩ # ⟩

5. Structural Change

Syntax

5.01 structural changes ::= SC structural change .

5.02 structural change ::= clist[change instruction]

5.03 change instruction ::= change or conditional change

5.04 conditional change ::= IF ⟨ restriction ⟩ THEN ⟨ structural change ⟩ opt[ELSE ⟨ structural change ⟩]

5.05 change ::= tree designator binary tree operator node designator or complex symbol designator binary complex operator node designator or unary operator node designator or complex symbol designator ternary complex operator node designator node designator;

5.06 node designator ::= integer cword

5.07 complex symbol designator ::= complex symbol or integer

5.08 tree designator ::= (tree) or node designator

5.09 binary tree operator ::= ADLAD or ALADE or ADLADI or
ALADEI or
ADFID or AFIDE or ADRIA or ARIAE or
ADRIS or ARISE or ADRISI or ARISEI or
ADLES or ALESE or ADLESI or ALESEI or
SUBST or SUBSE or SUBSTI or SUBSEI

5.10 binary complex operator ::= ERASEF or MERGEF or SAVEF

5.11 unary operator ::= ERASE or ERASEI

5.12 ternary complex operator ::= MOVEF

Remark

The operators listed here are just examples. They include those used in the MITRE grammars [21] and in the IBM Core Grammar [16]. The list can easily be expanded.

The changes are to be made in the order in which they appear in the structural change.

5.11 unary operator

ERASE n deletes the subtree headed by n, and furthermore erases its ancestors until a node with more than one daughter is encountered.

5.09 binary tree operators

The changes with binary tree operators are adjunctions of the form m ADXXXX m and mean that the subtree headed by the node corresponding to the label m is to be adjoined to the node corresponding to n .

n ADRIS m (n ADLES m) means that a copy of the subtree headed by n is to be adjoined as the rightmost (leftmost) sister of node m .

n ADFID m (n ADLAD m) means that a copy of the subtree headed by n is to be adjoined as the first (last) daughter node m .

These same operators, when the second letter (D) is missing and they are terminated by the letter E, i.e., ARISE, ALESE, AFIDE and ALADE, mean that the original subtree headed by n is to be erased in the course of the operation. That is, n ARISE m is equivalent to n ADRIS m, ERASE n .

n SUBST m, and n SUBSE m, mean that the subtree n is to be substituted for the subtree m .

The operators with names terminated by the letter I, ADLADI, ADRISI, ADLESI and SUBSTI, are the similar, but not identical, operators used in the IBM Core Grammar and defined in[16].

5.10 binary complex operators :-

The binary complex operators modify complex symbols. The complex symbol pointed to by an symbolr is the complex of the node corresponding to the term of the analysis labeled with the integer.

n ERASEF m means that the feature specifications of the complex symbol pointed to by n are to be deleted from the complex symbol pointed to by m .

n MERGEF m means that the feature specifications of n are to be merged into the complex symbol m .

n SAVEF m means that all feature specifications of m are to be deleted except for those of n, which are to be saved. Notice that | *SG *PRO | SAVEF m will leave in m only the specifications for SG and PRO, and will leave their values unchanged.

Table binary complex operators

The matrices show the values in m after the change is performed.

n MERGEF m

n \ n	+	-	*	abs
+	+	+	+	+
	-	-	-	-
*	+	-	*	*
abs	+	-	*	abs

n ERASEF m

n \ m	+	-	*	abs
+	abs	-	-	abs
-	+	abs	+	abs
*	abs	abs	abs	abs
abs	+	-	*	abs

n SAVEF m

n \ m	+	-	*	abs
+	+	abs	+	abs
-	abs	-	-	abs
*	+	-	*	abs
abs	abs	abs	abs	abs

5.12 ternary complex operator

n MOVEF m k is equivalent to first evaluating the result of n SAVEF m and saving it temporarily as r (without changing m) and then merging this result into m by r MERGEF k . Thus | *SG | MOVEF n m will change the complex symbol m so that the value of the feature SG is the same in m as in n.

6. Phrase structure

Syntax

6.01 phrase structure ::= PHRASESTRUCTURE list[phrase structure rule] $END

6.02 phrase structure rule ::= rule left = rule right .

6.03 rule left ::= node

6.04 rule right ::= choicestructure[list[node]]

Constraints

6.03 The rule left of the first phrase structure rule must be the sentence symbol.

6.01 The following ordering constraint is sometimes placed on the phrase structure: The phrase structure rules must be ordered so that, with the exception of the sentence symbol, no node occurs in a rule right below a rule in which it occurs as rule left.

In the computer system [24], this constraint is imposed only if the algorithm for directed random generation of base trees [23] is used.

Interpretation

6.04 The node on the rule left can be expanded to any of the lists of nodes obtained from the rule right choicestructure.

Remark

6.04 The use of the Kleene star to define rule schemata is not included.

Example

```
        "ALFREDI A N PROSE--TRAUGOTT"
PHRASESTRUCTURE
"I"    S = # (PRE) NP VP AUX (ADV) # .
"II"   PRE = (NEG) (Q).
"III"  AUX = (AUX1) T.
"IV"  T = (PRES,PAST).
"V"    AUX1 = (INF M, PP PERF).
"VI"   PERF = (HABB,BE).
"VII" VP = ((ADJ,NP) COP, MV).
"VIII"ADJ = (NP)( INT)AD.
"IX"  MV = (PASS)(NP)((NP,S))V(PRP BE).
"X"    PASS = PREP P.
"XI"  NP = (DET) N (S).
"XII" DET = (QUANT1)(DEM)((QUANT2,NUM))(D)(S).
        $ENDPSG
```

Reference

Further discussion of phrase structure is given in Doran [35].

7. Lexicon

Syntax

7.01 lexicon ::= LEXICON prelexicon lexical entries $END

7.02 prelexicon ::= feature definitions opt[redundancy rules]

7.03 feature definitions ::= category definitions
opt[inherent definitions]
opt[contextual definitions]

7.04 category definitions ::= CATEGORY list[category feature].

7.05 inherent definitions ::= INHERENT list[inherent feature].

7.06 contextual definitions ::= CONTEXTUAL clist[contextual definition].

7.07 contextual definition ::= contextual feature label = contextual feature

7.08 contextual feature label ::= word

7.09 redundancy rules ::= RULES clist[redundancy rule].

7.10 redundancy rule ::= complex symbol => complex symbol

7.11 lexical entries ::= ENTRIES list[lexical entry].

7.12 lexical entry ::= list[vocabulary word] list[complex symbol]

7.13 vocabulary word ::= word

Example

```
    PHRASESTRUCTURE
          S = # NP VP # . VP = V (NP). NP = (DET) N.
  $ENDPSG
  LEXICON
          CATEGORY V N DET   .
          INHERENT ABSTRACT ANIMATE COUNT HUMAN .
          CONTEXTUAL TRANS = <VP<_NP>>,  COMMON = <NP<DET _>>,
          ABSTSUBJ = <S<#NP<%N|+ABSTRACT| > VP<_%>#>>,
          NABSTSUB=<S<#NP<%N|-ABSTRACT|>VP<_%>#>>,
          ANIMSUBJ=<S<#NP<%N|+ANIMATE |>VP<_%>#>>,
          NABSTOBJ= <VP<_NP<% N| -ABSTRACT | >>>,
          NHUMOBJ = <VP<_NP<% N|-HUMAN|>>>,
          ANIMOBJ = <VP<_NP<% N|+ANIMATE|>>>  .
          RULES |+COUNT| => |+COMMON|, |+HUMAN|=>|+ANIMATE|,
          |+ABSTRACT| => |+COMMON -ANIMATE| .
  ENTRIES
          SINCERITY VIRTUE |+N -COUNT +ABSTRACT|
          BOY |+N + COUNT  +COMMON +ANIMATE +HUMAN|
          GEORGE NOAM|+N -COMMON +HUMAN|
          THE |+DET|
          G R O W EAT|+V +TRANS +ANIMSUBJ +NABSTOBJ|
          FRIGHTEN| +V +TRANS +ANIMOBJ|
          ELAPSE OCCUR |+V -TRANS +NANIMSUB|
   ADMIRE READ |+V  +TRANS+ANIMSUBJ|
   BUTTER |+N -COUNT -ABSTRACT|
   BOOK |+N -ANIMATE +COUNT|
   BEE |+N +COUNT +ANIMATE -HUMAN|
   READ WEAR |+V +NHUMOBJ|
   KNOW OWN |+V +TRANS|
   EGYPT  |+N -COMMON -ANIMATE|
   DOG |+N +COMMON -HUMAN +ANIMATE|
   CARROJ |+N +ANIMATE -HUMAN +COUNT|
          RUN|+V -TRANS +ANIMSUBJ|.
   $ENDLEX
```

Interpretation

7.06 The contextual feature label is used in complex symbols as an abbreviation for the contextual feature.

7.09 A redundancy rule (A ⇒ B) has the interpretation that if the left-hand complex symbol A is explicitly included in another complex symbol C, then the right-hand complex symbol B is-implicitly included in the complex symbol C .

Alternative a

4.09a value ::= + or - or * or value word

7.03a feature definitions ::= category definitions

opt[inherent definitions]

opt[analysis definitions]

opt[value definitions]

7.031a value definitions ::= VALUE list[value word]

7.032a value word ::= word

Remark

If 4.09a is chosen as an alternative to 4.09, the additional rules 7.03a, 7.031a and 7.032a would be needed for the lexicon.

Reference

The use of the lexicon in lexical insertion will be described in Friedman and Bredt [26].

8. Transformations

Syntax

8.01 transformations ::= TRANSFORMATIONS list[transformation | control program $END

8.02 transformation ::= identification structural description opt[restrictions] opt[structural changes]

8.03 identification ::= TRANS opt[integer] transformation name opt[list [parameter]] opt[keywords] .

8.04 parameter ::= group number or optionality or cyclicity or embedding

8.05 group number ::= I or II or III or IV or V or VI or VII

8.06 optionality ::= OB or OP

8.07 cyclicity ::= NC or C or CNR

8.08 embedding ::= EMB

8.09 keywords ::= (list[node]

8.10 structural description ::= SD analysis .

(for analysis see 2.01)

8.11 restrictions ::= RES restriction .

(for restriction see 3.01)

8.12 structural changes ::= SC structural change .

(for structural change see 5.01)

Interpretation

8.05 Group numbers are for use in the control program. If no group number is given in the identification, the group number of the previous transformation will be used, or I for the first transformation.

8.07 The cyclicity determines whether and how a transformation is to be retested after a successful application. If NC (non-cyclical), it will not be retested. If C (cyclic), it will be retested. If CNR (cyclical non-recursive), all analyses will be found before any changes are made. See also the discussion of the restriction NONREP.

8.06 If no optionality is given, OP (optional) is assumed, rather than OB (obligatory).

8.07 The null option is NC.

8.09 The keywords are used by the control program to avoid unnecessary analysis. If none of the keywords appear in the tree, the analysis routine is bypassed.

Control Program

As long as the theory of transformational grammar is still changing, grammars are likely to differ in the order of consideration of the transformations. Therefore, to complete the description of a grammar, it is necessary to specify the order in which the transformations are to be considered. Rather than choose a particular order for this system, we have chosen to include a control program as part of the specification of the grammar. However, since this part of the syntax is highly experimental and subject to more radical change than the rest of the syntax, we do not include it in full in this presentation.

9.01 control program ::= CP sclist[opt[label :] control instruction]

9.02 label ::= word

9.03 control instruction ::= transformation name or group number or transformation list or conditional instruction or goto instruction or repeat instruction

9.04 goto instruction ::= GOTO label

9.05 repeat instruction ::= RPT opt[integer] control instruction

9.06 conditional instruction ::= IF transformation name THEN ⟨ list [control instruction]⟩ opt[ELSE ⟨ list[control instruction] ⟩]

References

Transformations are further described in [32]. A full description of the control language and its use will be presented in Friedman and Pollack [34].

ACKNOWLEDGMENT

Thomas H. Bredt, Theodore S. Martner and Bary Pollack have made important contributions to the formalization of the syntax. Their work has been primarily in the areas of complex symbols and lexicon (Bredt), analysis (Martner), and restrictions and control language (Pollack). They have all made many helpful suggestions in other areas as well.

REFERENCES

[1] J. W. Backus, The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. ICIP Paris, June 1959.

[2] Paul Chapin, On the Syntax of Word Derivation in English, M.I.T. Thesis, 1967.

[3] Noam Chomsky, Syntactic Structures, Mouton & Co., The Hague, 1957.

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

[5] Noam Chomsky, Remarks on Nominalization, to appear in Peter S. Rosenbaum and Roderick Jacobs, eds., Readings in Transformational Grammar, Blaisdell Publishing Company.

[6] J. Friedman, SYNN, an experimental analysis program for transformational grammars, WP-229, The MITRE Corporation, 1965.

[7] L. N. Gross, On-line programming system user's manual, MTP-59, The MITRE Corporation, 1967.

[8] L. N. Gross, M.I.T. Rule Tester, 1968.

[9] George Lakoff, On the nature of syntactic irregularity, NSF-16, The Computation Laboratory, Harvard University, 1965.

[10] D. Lieberman, Design of a grammar tester, in [11].

[11] D. Lieberman, ed., Specification and Utilization of a Transformational Grammar, AFCRL-66-270, 1966.

[12] D. L. Londe and W. J. Schoene, TGT: Transformational Grammar Tester, Systems Development Corporation, 1967.

[13] John McCarthy, et. al., LISP 1.5 Programmer's Manual, M.I.T. Press, 1962.

[14] Peter Naur, ed., Revised Report on the Algorithmic Language Algol 60 International Federation for Information Processing, 1962.

[15] Stanley R. Petrick, A Recognition Procedure for Transformation Grammars, M.I.T. Thesis, 1965.

[16] P. Rosenbaum and D. Lochak, The IBM Core Grammar of English, in [11].

[17] John R. Ross, A proposed rule of tree-pruning, paper presented to the Linguistic Society of America, 1965.

[18] John R. Ross, Constraints on Variables in Syntax, M.I.T. Thesis, 1967.

[19] Sanford A. Schane, A schema for sentence coordination, MTP-10, The MITRE Corporation, 1966.

[20] R. Stockwell, P. Schacter, B. Partee, et. al., Working Papers of the English Syntax Project, U.C.L.A., 1967.

[21] A. M. Zwicky, J. Friedman, B. C. Hall, and D. E. Walker, The MITRE Syntactic Analysis Procedure for Transformational Grammars, Fall Joint Computer Conference 1965, 27, 317-326. See also MTP-9, The MITRE Corporation, 1965.

The following references are working papers and reports of the Computational Linguistics Project, Computer Science Department, Stanford University.

[22] Robert W. Doran, 360 O.S. FORTRAN IV Free-field Input/output Subroutine Package, CS - 79, AF - 14, October 1967.

[23] Joyce Friedman, Directed Random Generation of Sentences, CS - 80, AF - 15, October 1967 (submitted for publication).

[24] Joyce Friedman, A Computer System for Transformational Grammar, CS - 84, AF - 21, January 1968 (submitted for publication).

[25] Joyce Friedman, Computer Experiments in Transformational Grammar, AF - 22 and AF - 23, February 1968.

[26] Joyce Friedman and Thomas H. Bredt, Lexical Insertion in Transformational Grammar, AF - , forthcoming.

[27] Joyce Friedman and Bary Pollack, A Control Language for Transformational Grammar, AF - , forthcoming.

[28] Joyce Friedman and Theodore S. Martner, Analysis in Transformational Grammar, AF - , forthcoming.

[29] Joyce Friedman, ed., Users' and Programmers' Guide to a Transformational Grammar System. This document is not yet complete, but the following sections are available as working papers:

[30] J. Friedman, Subroutine structure, AF - 17, November 1967.

[31] J. Friedman, Trees, AF - 1, September 1966.

[32] J. Friedman, Input routine for transformations, AF - 16, October 1967.

[33] J. Friedman, Input routine for structural change, AF - 18, November 1967.

[34] Bary Pollack, Routines for restrictions, AF - 19, December 1967.

[35] R. W. Doran, Section III 14 of PSGINN, AF - 9, May 1967.

[36] Olasope O. Oyelaran, AF Test Grammar, AF - 13, September 1967.

APPENDIX I

The following is an example of a transformational grammar, based on one written by Olasope Oyelaran [36]. It is not intended here to be linguistically correct, but is just an example of the use of formats. Strings within quotation marks are comments, and are ignored by the program.

```
 " AF TEST    OLASOPE OYELARAN , AUG. 22 , 1967   .   "

PHRASESTRUCTURE
S = # NP VP # .
VP = (PRE) V (((NP) (PP) (AGNT), S, AP) ).
V = AUX (VB , COP).
AUX = ((DO, (HAVE EN)(BE ING) )) AUXA.
AUXA = (MOD) (PRES, PAST) (ASP).
ASP = (IMPERF,PERF) .
AP = ((PRE) ADJ (S), S)    .
PP = PRT NP.
NP = (NP S, (D) N NU, S).
NU = (SG,PL).
D = (PRE) (ART(ADJ) (S), (D) ADJ).
PRE = (NEG)   .
ART = (WH) (INDEF, DEF).
$END     "END OF PHRASESTRUCTURE"

LEXICON
CATEGORY  VB COP ADJ N DEF INDEF PRT NOMINALIZER MOD .
INHERENT COUNT PRO ANIMATE HUMAN ABSTRACT MASC SG1 SG2 SG3
          LOC TIME PLACE   .
CONTEXTUAL
      TRANS = <VP/<%_NP %>>,
      ANIMSUBJ = <S< # NP < % N|+ANIMATE| % >VP/<%_%>%>>,
      HUMSUBJ = < S< # NP < % N|+HUMAN| % >VP/<%_%>%>>,
      ABSTOBJ = <VP <% _ NP < % N|+ABSTRACT | % > % > >,
      VPCOMP = <VP < % V < AUX _ % > S % > >,
      SGNOUN = <NP /<% _ SG>>,
      COMMON = <NP<D_NU>>,
      VPADJ = < VP < % V < AUX _ > AP < % ADJ % > % > > .
RULES      |+COUNT| => |+COMMON|,
           |+ABSTRACT| => |+COMMON -COUNT -ANIMATE| ,
           |-ANIMATE| => |-HUMAN| ,
           |+HUMAN| =>|+ANIMATE|  .
ENTRIES
JOHN CHOMSKY RUSS |+N -COMMON + HUMAN +MASC | ,
MARY |+N -COMMON +HUMAN -MASC| ,
GRAMMAR | +N +COMMON -ABSTRACT |
OFFICE |+N +COUNT -ANIMATE| ,
MIGHT |+N +ABSTRACT|,
COME | +VB -TRANS| ,
DEFEND | +VB +TRANS +ANIMSUBJ| ,
WEEP | +VB -TRANS +ANIMSUBJ |,
KNOW | +VB +TRANS +ANIMSUBJ| | +VB +VPCOMP +HUMSUBJ |,
CLAIM |+VB +TRANS +HUMSUBJ|
      | +VB +VPCOMP +HUMSUBJ | ,
WRITE | +VB +ABSTOBJ | ,
BE |+COP +TRANS|  |+COP +VPADJ| ,
BE | +VB -TRANS| ,
RIGHT POOR |+ADJ|,
FOR AT IN |+PRT|,
CAN |+MOD|,
THE |+DEF|, SOME |+INDEF|,
WHEN THAT W H Y |+NOMINALIZER| .
$END   "END OF LEXICON"

TRANSFORMATIONS
"CYCLIC RULES 1."
```

```
TRANS 1 NUAG "NUMBER AGREEMENT"        I  CNR  OB.
 SD % N 1NU 2V % .
 SC 1 ADRIS 2.
TRANS 2 RELNOM "RELATIVE NOMINALIZATION"      OB.
 SD % NP V NP/<S/<# 1NP V %> %> %.
 SC NOMINALIZER ADLES 1.
TRANS 3 PRO1  "PRONOMINALIZATION"  NC EMB       OB.
 SD % D/<% 1N 2NU >% NP<S<# 3N 4NU> V %> % .
 RES 1 EQ 3 & 2 EQ 4 .
 SC |+PRO| MERGEF 3.
TRANS 4 PRO2  "PRONOMINALIZATION"               OP.
 SD % 1D<N NU>  V  NP< S<   2D<3N NU> V % >  % > %.
 RES 1 EQ 2 .
 SC |+PRO| MERGEF 3.
TRANS 5 NEG   "NEGATIVE  PLACEMENT" OB (NEG) .
 SD % 1NEG 2(DO,HAVE EN, BE ING, MOD) VB %.
 SC 1 ADRISE 2.
TRANS 6 WHPLA  "WH PLACEMENT" OB (WH).
 SD % ART S/< % 1NP % S/< % 2NP /<(PRE) WH (DEF,INDEF) N NU > % > % > %.
 SC 2 ADLESE 1.
TRANS 7 RELEX  "RELATIVE  EXTRAPOSITION"      OB EMB C .
 SD % NP V D<ART # 1S/<% (PRE) WH %> %>   N 2NU.
 SC  1 ADRISE 2.
"CYCLIC RULES 2 "   "RULES 8 TO  11 ARE ORDERED"
TRANS 8 NUMCON "NUMBER CONCORD"                 OB.
 SD % N 1NU V<2AUX %> %.
 SC  1 ADRIS 2.
TRANS  9 ASP1  "ASPECT SPECIFICATION"   OB (ING IMPERF).
 SD % V< BE 1ING (PRES, PAST) IMPERF 2%> %.
 SC 1 ADRISE 2.
TRANS  10 ASP2  "ASPECT SPECIFICATION" OB (HAVE EN PERF).
 SD % V<HAVE 1EN (PRES, PAST) PERF 2%>.
 SC  1 ADRISE 2.
TRANS  11 "A" SIMPV "SIMPLE VERB TRANSFORMATION"     OB.
 SD % V<% 1(PRES,PAST) 2NU 3(VB,COP)> %.
 SC  1 ADRISE 3 , 2 ADRISE 3.
TRANS  11 "B" MOD    "COMPOUND VERB" OB (MOD) .
 SD % V<MOD (PRES,PAST) 1NU>%.
 SC ERASE  1.
TRANS  12  BOUNDE   "BOUNDARY ERASURE"         OB.
 SD 1# % 2#.
 SC ERASE  1  , ERASE 2 .
"POST CYCLIC RULES."       "PARTIALLY ORDERED."
TRANS 13 WHREP1 "WH REPLACEMENT"      II     OB (WH DEF).
 SD % NP<(PRE) 1WH 2DEF N NU> %.
 SC WHICH SUBST 2, ERASE  1.
TRANS  14  WHREP2  "WH REPLACEMENT"  OB (WH INDEF) .
 SD % NP<( PRE) 1WH 2INDEF N NU> %.
 SC    WHICHEVER SUBST 2, ERASE 1.
TRANS 16  THAT    "THAT SUBSTITUTION." OP (WHICH)  .
 SD  %  1*<N NU> % NP< 3 WHICH   4*<N NU>> %.
 RES 1 EQ 4.
 SC THAT SUBST 3 , ERASE 4 .
TRANS  17  WHPRO1    "WH PRONOMINAL SUBSTITUTION " OP (WHICH).
 SD % N % NP<  1WHICH 2N|+HUMAN| %> %.
 SC     WHO SUBST  1, ERASE 2.
TRANS  18  WHPRO2    OP (WHICH).
 SD % N % NP<  1WHICH 2N|+ABSTRACT| %> %.
 SC      WHAT SUBST 1, ERASE 2.
TRANS 20  NEGSP "NEGATIVE SPELLING"  OB (NEG).
 SD  %2NEG  %.
```

```
 SC NOT SUBST 2.
"POST CYCLIC V-TRANSFORMATION,  ORDERED."
TRANS 22 V2 OB (BE PRES PL).
 SD % 1BE 2PRES 3PL %.
 SC ARE SUBST 1, ERASE 2, ERASE 3.
TRANS 24   V4 OB (PRES SG).
SD % N 1 (BE, HAVE, DO, VB) 2PRES 3SG %.
 SC   S ADRIS 1, ERASE 2, ERASE 3.
TRANS 25   V5 OB (PRES).
 SD % (HAVE, DO, VB) 1PRES 2NU %.
 SC ERASE 1, ERASE 2.
TRANS 26   V6 OB (BE PAST).
 SD % 1BE 2PAST 3(PL, SG) %.
 SC   WERE SUBST 1, ERASE 2, ERASE 3.
TRANS 27   V7 OB (BE PAST SG).
 SD % 1BE 2PAST 3SG %.
 SC   WAS SUBST 1, ERASE 2, ERASE 3.
TRANS 28 V8 OB (PAST).
 SD % (HAVE, DO, VB) 1PAST 2NU %.
 SC ED SUBST 1, ERASE 2.
TRANS 29 NSP1      "NOUN SPELLING" OB (PL).
 SD % N 1PL %.
 SC S SUBST 1.
TRANS 30 NSP2                                OB.
 SD % N 1NU %.
 SC ERASE 1.
CP I II III.  "CONTROL PROGRAM"
$END   "END OF TRANSFORMATIONS"
  $TEST
```

APPENDIX II

COMPLETE SYNTAX FOR TRANFORMATIONAL GRAMMAR

```
0.01  TRANSFORMATIONAL_GRAMMAR ::= PHRASE_STRUCTURE LEXICON TRANSFORMATIONS $END

1.01  TREE_SPECIFICATION ::= TREE OPT< , CLIST< WORD TREE >> .
1.02  TREE ::= NODE OPT< COMPLEX_SYMBOL > OPT< < LIST< TREE > > >
1.03  NODE ::= WORD  OR  SENTENCE_SYMBOL  OR  BOUNDARY_SYMBOL
1.04  SENTENCE_SYMBOL ::= S
1.05  BOUNDARY_SYMBOL ::= #

2.01  ANALYSIS ::= LIST< OPT< INTEGER > TERM >
2.02  TERM ::= STRUCTURE  OR  SKIP  OR  CHOICE
2.03  STRUCTURE ::= ELEMENT OPT< COMPLEX_SYMBOL > OPT< OPT< ¬ > OPT< / > < ANALYSIS > >
2.04  ELEMENT ::= NODE  OR  *  OR  -
2.05  CHOICE ::= ( CLIST< ANALYSIS > )
2.06  SKIP ::= % OPT< OPT< ¬ > OPT< & > < CLIST< STRUCTURE > > >

3.01  RESTRICTION ::= BOOLEANCOMBINATION< CONDITION >
3.02  CONDITION ::= UNARY_CONDITION  OR  BINARY_CONDITION
3.03  UNARY_CONDITION ::= UNARY_RELATION INTEGER
3.04  BINARY_CONDITION ::= INTEGER BINARY_TREE_RELATION NODE_DESIGNATOR  OR
                           INTEGER BINARY_COMPLEX_RELATION COMPLEX_SYMBOL_DESIGNATOR
3.05  NODE_DESIGNATOR ::= INTEGER  OR  NODE
3.06  COMPLEX_SYMBOL_DESIGNATOR ::= COMPLEX_SYMBOL  OR  INTEGER
3.07  UNARY_RELATION ::= TRM  OR  NTRM  OR  NUL  OR  NNUL  OR  NONREP
3.08  BINARY_TREE_RELATION ::= EQ  OR  NEQ  OR  DOM  OR  NDOM
3.09  BINARY_COMPLEX_RELATION ::= INC1  OR  NINC1  OR  INC2  OR  NINC2  OR  CSEQ  OR  NCSEQ  OR  NDST  OR  NNDST

4.01  COMPLEX_SYMBOL ::= | LIST< FEATURE_SPECIFICATION > |
4.02  FEATURE_SPECIFICATION ::= VALUE FEATURE
4.03  FEATURE ::= CATEGORY_FEATURE  OR  INHERENT_FEATURE  OR  CONTEXTUAL_FEATURE_IDENTIFIER  OR  RULE_FEATURE
4.04  CATEGORY_FEATURE ::= CATEGORY
4.05  CATEGORY ::= WORD
4.06  INHERENT_FEATURE ::= WORD
4.07  RULE_FEATURE ::= TRANSFORMATION_NAME
4.08  CONTEXTUAL_FEATURE_IDENTIFIER ::= CONTEXTUAL_FEATURE  OR  CONTEXTUAL_FEATURE_LABEL
4.09  CONTEXTUAL_FEATURE ::= < ANALYSIS > OPT< RESTRICTIONS >
4.10  VALUE ::= +  OR  -  OR  *

5.01  STRUCTURAL_CHANGES ::= SC STRUCTURAL_CHANGE .
5.02  STRUCTURAL_CHANGE ::= CLIST< CHANGE_INSTRUCTION >
5.03  CHANGE_INSTRUCTION ::= CHANGE  OR  CONDITIONAL_CHANGE
5.04  CONDITIONAL_CHANGE ::= IF < RESTRICTION > THEN < STRUCTURAL_CHANGE >
                             OPT< ELSE < STRUCTURAL_CHANGE > >
5.05  CHANGE ::= UNARY_OPERATOR NODE_DESIGNATOR  OR
        TREE_DESIGNATOR BINARY_TREE_OPERATOR NODE_DESIGNATOR  OR
        COMPLEX_SYMBOL_DESIGNATOR BINARY_COMPLEX_OPERATOR NODE_DESIGNATOR
        OR  COMPLEX_SYMBOL_DESIGNATOR TERNARY_COMPLEX_OPERATOR NODE_DESIGNATOR NODE_DESIGNATOR
5.06  NODE_DESIGNATOR ::= INTEGER  OR  WORD
5.07  COMPLEX_SYMBOL_DESIGNATOR ::= COMPLEX_SYMBOL  OR  INTEGER
5.08  TREE_DESIGNATOR ::= ( TREE )  OR  NODE_DESIGNATOR
5.09  BINARY_TREE_OPERATOR ::=  ADLAD  OR  ALADE  OR  ADLADI  OR  ALADEI  OR  ADFID  OR  AFIDE  OR
                  ADRIS  OR  ARISE  OR  ADRISI  OR  ARISEI  OR  ADLES  OR  ALESE  OR  ADLESI  OR  ALESEI
                  OR  ADRIA  OR  ARIAE  OR  SUBST  OR  SUBSE  OR  SUBSTI  OR  SUBSEI
5.10  BINARY_COMPLEX_OPERATOR ::=  ERASEF  OR  MERGEF  OR  SAVEF
5.11  UNARY_OPERATOR  ::=  ERASE  OR  ERASEI
5.12  TERNARY_COMPLEX_OPERATOR ::=  MOVEF

6.01  PHRASE_STRUCTURE ::= PHRASESTRUCTURE LIST< PHRASE_STRUCTURE_RULE > $END
```

```
6.02   PHRASE_STRUCTURE_RULE ::= RULE_LEFT = RULE_RIGHT .
6.03   RULE_LEFT ::= NODE
6.04   RULE_RIGHT ::= CHOICESTRUCTURE< LIST< NODE >>

7.01   LEXICON ::= LEXICON PRELEXICON LEXICAL_ENTRIES $END
7.02   PRELEXICON ::= FEATURE_DEFINITIONS OPT< REDUNDANCY_RULES >
7.03   FEATURE_DEFINITIONS ::= CATEGORY_DEFINITIONS OPT< INHERENT_DEFINITIONS > OPT< CONTEXTUAL_DEFINITIONS >
7.04   CATEGORY_DEFINITIONS ::= CATEGORY LIST< CATEGORY_FEATURE > .
7.05   INHERENT_DEFINITIONS ::= INHERENT LIST< INHERENT_FEATURE > .
7.06   CONTEXTUAL_DEFINITIONS ::= CONTEXTUAL CLIST< CONTEXTUAL_DEFINITION > .
7.07   CONTEXTUAL_DEFINITION ::= CONTEXTUAL_FEATURE_LABEL = CONTEXTUAL_FEATURE
7.08   CONTEXTUAL_FEATURE_LABEL ::= WORD
7.09   REDUNDANCY_RULES ::= RULES CLIST< REDUNDANCY_RULE > .
7.10   REDUNDANCY_RULE ::= COMPLEX_SYMBOL => COMPLEX_SYMBOL
7.11   LEXICAL_ENTRIES ::= ENTRIES LIST< LEXICAL_ENTRY > .
7.12   LEXICAL_ENTRY ::= LIST< VOCABULARY_WORD > LIST< COMPLEX_SYMBOL >
7.13   VOCABULARY_WORD ::= WORD

8.01   TRANSFORMATIONS ::= TRANSFORMATIONS LIST< TRANSFORMATION > CONTROL_PROGRAM $END
8.02   TRANSFORMATION ::= IDENTIFICATION STRUCTURAL_DESCRIPTION OPT< RESTRICTIONS > OPT< STRUCTURAL_CHANGES >
8.03   IDENTIFICATION ::= TRANS OPT< INTEGER > TRANSFORMATION_NAME OPT< LIST< PARAMETER >> OPT< KEYWORDS > .
8.04   PARAMETER ::= GROUP_NUMBER OR OPTIONALITY OR CYCLICITY OR EMBEDDING
8.05   GROUP_NUMBER ::= I OR II OR III OR IV OR V OR VI OR VII
8.06   OPTIONALITY ::= OB OR OP
8.07   CYCLICITY ::= NC OR C OR CNR
8.08   EMBEDDING ::= EMB
8.09   KEYWORDS ::= ( LIST< NODE > )
8.10   STRUCTURAL_DESCRIPTION ::= SD ANALYSIS .
8.11   RESTRICTIONS ::= RES RESTRICTION .
8.12   STRUCTURAL_CHANGES ::= SC STRUCTURAL_CHANGE .

9.01   CONTROL_PROGRAM ::= CP SCLIST< OPT< LABEL : > CONTROL_INSTRUCTION >
9.02   LABEL ::= WORD
9.03   CONTROL_INSTRUCTION ::= TRANSFORMATION_NAME OR GROUP_NUMBER OR TRANSFORMATION_LIST
                 OR CONDITIONAL_INSTRUCTION OR GOTO_INSTRUCTION OR REPEAT_INSTRUCTION
9.04   GOTO_INSTRUCTION ::= GOTO LABEL
9.05   REPEAT_INSTRUCTION ::= RPT OPT< INTEGER > CONTROL_INSTRUCTION
9.06   CONDITIONAL_INSTRUCTION ::= IF TRANSFORMATION_NAME THEN < LIST< CONTROL_INSTRUCTION > >
                           OPT< ELSE < LIST< CONTROL_INSTRUCTION > > >
```