T H E U N I V E R S I T Y O F M I C H I G A N

DEPARTMENT OF COMPUTER AND COMMUNICATION SCIENCES
Phonetics Laboratory

Natural Language Studies No. 9

PHONOLOGICAL GRAMMAR TESTER: DESCRIPTION

Joyce Friedman
Yves Ch. Morin

September 1971

ACKNOWLEDGMENT

An earlier version of this report was presented to the meeting of the Association for Computational Linguistics, Columbus, Ohio, July 23, 1970. The research reported here was supported in part by the National Science Foundation under Grant GS-2771, and in part by the International Business Machines Corporation.

ABSTRACT

A computer program to test phonological grammars has been constructed as an extension to Friedman's computer system for transformational syntax. The program accepts the phonological component of a transformational grammar in a format derived from The Sound Pattern of English and underlying phonological forms to which it applies the phonological grammar.

In this report, the system is presented descriptively, with emphasis on the notation for a phonological grammar and its interpretation by the program, for the reader who is thinking of using the system. Extended examples show grammars acceptable to the program and the result of computer runs. The underlying theory and its justification by empirical data are given separately in Phonological Grammar Tester: Underlying Theory.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

CHAPTER 1

INTRODUCTION

This report describes an extension of Friedman et al. (1971), Computer Model of Transformational Grammar to include the phonological component of a transformational grammar.

This extension is implemented as a tool to help linguists write the phonological component of a tranformational grammar. The program and outputs a derivation of the underlying form according to this grammar. Hence, the program permits the user to test a phonological grammar by inputing a tentative version and study the derivation of some underlying phonological forms, and, once a phonological grammar is satisfactory, to test the adequacy of underlying forms by comparing the surface forms obtained through derivation with the actual phonetic representations.

This system differs from the two previous computer programs for phonological grammar, Bobrow and Fraser (1968) and Fromkin and Rice (1969), in that it tests a full phonological grammar rather than phonological rules. It also incorporates recent suggestions in linguistics theory. For example, (a) it is possible to modify the whole tree structure during the phonological derivation, and (b) "everywhere rules" can be defined. The user has the ability to define for himself the order of application of the rules; this may be either cyclic or

noncyclic. The program analyses systematic phonemes as nodes on a par with the other nodes of the tree and is thus more general. Regular tree operations apply to the phonemes: metathesis is viewed as the permutation of two nodes, elision as the deletion of a node and assimilation or dissimilation as the modification of feature values in a node.

The theory underlying the computer program is derived from the theory presented by Chomsky and Halle in The Sound Pattern of English (1968); however, we have felt free to make some modifications to this theory. The underlying theory and its justification are reported in Morin and Friedman (1971).

The computer system for phonology is completely compatible with the syntax system; this implies that the features described here can also be of use in syntax, in particular n-ary features (cf. Friedman and Myslenski, 1970). We describe here, however, only the parts of the system which are relevant to the phonological component. This description does not presuppose any knowledge of the syntax component (Friedman et al., 1971); we do, however, refer the reader to that description for some points which are relevant to syntax rather than to phonology. The treatment here parallels, insofar as possible, that in the syntax description.

To help the reader to understand the system as a whole, we give in Chapter 2 an example of phonological grammar containing both the conversion component and the rule component. This example is most meaningful to readers familiar with some notation for phonological

grammars; others may choose to ignore it and return to it as its parts are discussed. In Chapter 3 we introduce the formal metalanguage used throughout this report and for the summary of the syntax given in Appendix B. Chapter 4 presents the basic concept of tree, Chapter 5 the concept of complex symbol. Chapter 6 describes the conversion of complex symbols when the trees are output by the program. Chapter 7 describes the representation of rules and their mechanism of application; we describe in detail the application algorithm as it is essential in the choice of the segments which are modified. Chapters 8 and 9 present the concept of rule ordering and mode of application. In Chapter 10 we give some indication of the use of the program. Finally, in the Appendixes we give a summary of the syntax, and some computer experiments in generative phonology.

CHAPTER 2

OVERVIEW

Before beginning the formal discussion of the computer model, we present in this chapter an example which illustrates both the notation for grammars and the use of the program. This example consists of a small grammar and a sample derivation. The appendix presents a longer version and the full derivation of two words in this grammar.

Figure 1 shows the conversion lexicon. The conversion lexicon associates symbols with sets of features. This lexicon is implicit in most phonological descriptions where the symbols are either defined during the description of the grammar, or assumed to be known. The entry FEATURES lists all features which are used in the grammar. The entry VARIABLE defines symbols which represent (or are the names for) some type of segments. For instance, the symbol V represents a vowel, i.e., a segment containing the two feature specifications +VOC and -CONS; the symbol C represents a consonant, i.e., a segment which is not a vowel (the symbol ¬ is the logical symbol for "not"). The symbols defined in this entry are used for inputs and in the description of the rules. The entry PHONUNIT, like the entry VARIABLE, defines symbols; these symbols are used for output. For example when a string is output which contains the segment

-CONS +VOC -HIGH -BACK +LOW -ANT -ROUND

the symbol A is to be output, and not the symbol V, which explains why

the two lexicons must be kept distinct. The entry DIACRIT defines symbols which are attached as diacritics in the output of strings.

```
PHONLEXICON
    FEATURES
          VOC HIGH BACK LOW ANT ROUND CONS COR VOICED STRID CONT
          TENSE STRESS  .
    VARIABLE
          OO = |-CONS +VOC -HIGH +BACK -LOW -ANT +ROUND|,
          A  = |-CONS +VOC -HIGH -BACK +LOW -ANT -ROUND|,
          E  = |-CONS +VOC -HIGH -BACK -LOW -ANT -ROUND|,
          UH = |-CONS +VOC -HIGH +BACK -LOW -ANT -ROUND|,
          EH = |-CONS +VOC -HIGH -BACK -LOW -ANT -ROUND -TENSE -STRESS
               +RED|,
          R  = |+CONS +VOC -ANT +COR +VOICED -STRID +CONT|,
          S  = |+CONS -VOC +ANT +COR -VOICED +STRID +CONT|,
          K  = |+CONS -VOC -ANT -COR -VOICED -STRID -CONT|,
          J  = |+CONS -VOC -ANT +COR +VOICED +STRID -CONT|,
          V  = |-CONS +VOC|,
          C  = ¬ V .
    PHONUNIT
          OO=OO, A=A, E=E, UH=UH, EH=EH, R=R, S=S, K=K, J=J.
    DIACRITIC
          '  = |1STRESS|.
$ENDLEX
```

Figure 1

Figures 2 and 3 show a set of phonological rules. In Figure 2 they are as they would be represented in traditional phonological description and in Figure 3 as prepared for input to the computer program. The computer form is basically a linearization of the usual form:

a. The context of a rule must be within angular brackets.

b. A single-quote symbol ' precedes each segment in the context.

c. The context must match the entire string under derivation. If it need not match the initial segment, the variable symbol % must begin the context; if it need not match the final segment, the variable symbol % must end the context.

(0) Redundancy

$$V \rightarrow \begin{bmatrix} -\text{tense} \\ -\text{stress} \end{bmatrix}$$

(1) Main stress

$$V \rightarrow [1\ \text{stress}] / \left[X \_ C_o \left(\begin{bmatrix} -\text{cons} \\ +\text{voc} \\ -\text{tense} \end{bmatrix} \right) (C)) \left(\begin{bmatrix} -\text{cons} \\ +\text{voc} \\ -\text{tense} \end{bmatrix} C_o \right) \right]_N$$

(2) Rounding adjustment

$$\begin{bmatrix} +\text{voc} \\ -\text{cons} \\ \alpha\text{round} \\ +\text{back} \end{bmatrix} \rightarrow [-\ \text{round}] / \left\{ \begin{bmatrix} \overline{\quad\quad} \\ -\text{tense} \end{bmatrix} \right\}$$
$$\left\{ \begin{bmatrix} \overline{\quad\quad} \\ \beta\text{low} \\ \beta\text{round} \\ +\text{tense} \end{bmatrix} \right\}$$
$$\_\ V$$

(3) e-elision

$$\begin{bmatrix} -\text{back} \\ -\text{high} \\ -\text{low} \\ -\text{cons} \end{bmatrix} \rightarrow \emptyset \ / \ - \left\{ \begin{matrix} \# \\ + \end{matrix} \right\}$$

(4) Vowel reduction

$$\begin{bmatrix} -\text{cons} \\ +\text{voc} \\ -\text{stress} \\ -\text{tense} \end{bmatrix} \rightarrow \text{ə}$$

Figure 2

d. The square bracket notation is replaced by the angular bracket notation, more specifically, the notation N < . . . > replaces the notation [...]$_N$.

e. Curly brackets are represented by parentheses; a comma separates the alternate elements.

f. The dash appears before a complex symbol rather than inside the complex symbol.

g. The indices appear before, rather than after, the segments: 'O'C replaces C_o.

The control program (CP) and the identification of the rules define the order of application of the rules of the grammar. In this example, the rules apply in their order of appearance in the grammar.

Figures 4 and 5 show the derivation of the word [kʌ́rəj] from the underlying form /koræje/: Figure 4 is the traditional presentation and Figure 5 shows part of the derivation of the computer program.

The output of the program contains a list of the rules which have applied, the input string before and after application of the rules, and a list of the rules which have applied. A history of the derivation indicates which rules have been invoked (ANTEST CALLED ...) and whether they have applied (ANTEST RETURNS ...). The changes are shown as they take place; MERGEF IN n indicates that new feature specifications have been introduced in the complex symbol with number n (the segments are numbered during the input); ERASE O n indicates that the segment with number n has been erased.

```
TRANSFORMATIONS
RULE REDVOWL AACC. V => |-STRESS -TENSE|.
RULE MSTR .  V => |1STRESS| /< N <% _ 'O'C ('|-CONS +VOC -TENSE| ('C))
                                ('|-CONS +VOC -TENSE| 'O'C)# > >.
RULE RADJ.        |+VOC -CONS (ALPHA)ROUND +BACK| => |(-ALPHA)ROUND|
          /<% ( _|-TENSE|, _|(BETA)LOW (BETA)ROUND +TENSE|, _'V) %>.
RULE EELI.          |-CONS -HIGH -BACK -LOW| => * /<% _ (+,#) %>.
RULE EHRED.         |-CONS +VOC -STRESS -TENSE| => EH.
CP@(N) I.
 $END $MAIN FTRIN TRAN.
 N< # 'K 'OO 'R 'A 'J 'E # >.
```

Figure 3

	# k o r æ j e #
main stress placement	ó
rounding adjustment	ʌ́
e-elision	ø
vowel reduction	ə
	# k ʌ́ r ə j #

Figure 4

```
 1 N          2 #
              3 K
              4 OO
              5 R
              6 A
              7 J
              8 E
              9 #

    # K OO R A J E #

****      TRANSFORMATIONS      ****
SCAN CALLED AT    4      @
SCAN CALLED AT    5      I
ANTEST CALLED FOR    1"REDVOWL "(AACC)
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR MERGEF  IN        4
CHANGE. HAVE CSEXCH FOR MERGEF  IN        6
CHANGE. HAVE CSEXCH FOR MERGEF  IN        8
ANTEST CALLED FOR    2"MSTR    "(AC   )
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR MERGEF  IN        4
ANTEST CALLED FOR    3"RADJ    "(AC   )
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR MERGEF  IN        4
ANTEST CALLED FOR    4"EELI    "(AC   )
ANTEST RETURNS ** 1**
CHANGE. HAVE ELEMOP FOR ERASE   IN    0   8
ANTEST CALLED FOR    5"EHRED   "(AC   )
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR MERGEF  IN        6
SCAN CALLED AT    6      .

TRANSFORMATIONS WHICH HAVE APPLIED ARE
              1          1 REDVOW
              2          2 MSTR
              3          3 RADJ
              4          4 EELI
              5          5 EHRED

 1 N          2 #
              3 K
              4 UH'
              5 R
              6 EH
              7 J
              9 #

    # K UH' R EH J #

 4 UH'   |-TENSE|
```

Figure 5

CHAPTER 3

METALANGUAGE

The metalanguage used to specify the syntax of phonological grammars is a modification of the Backus-Naur Form (BNF) used in the definition of the programming language Algol (Backus, 1959; Naur, 1962) and described in J. Friedman et al. (1971, §3).

In this chapter, we shall mention only the main characteristics of this metalanguage. A complete definition of the form of phonological grammar is included as Appendix A to this report. This syntax is designed for (human) readers and not for syntax-directed translation by computer as is frequently the case in the definition of programming languages.

3.1 CONVENTIONS OF THE METALANGUAGE

The metalanguage is a formalism for writing context-free rules or "meta-rules" which define a phonological grammar. A meta-rule is of the form:

RHS ::= LHS

which reads as "the element RHS consists of the elements in LHS."

3.1.1 Primitive Formats

Basic to the metalanguage are the three primitive formats word, symbol, and integer. A word is a contiguous string of uppercase

letters and digits beginning with a letter. Examples of words are A, A/B, VERB. A symbol is a contiguous string of uppercase letters, digits and special characters (with the exception of = (equal), " (double quote), and $ (dollar sign)). The total length of a symbol may not exceed 8 characters. Example of symbols are A, 3, *, N/1/*. An integer is a contiguous string of digits, not beginning with zero. Examples of integers are 312, 5, and 6000.

3.1.2 Nonterminal Symbols

The nonterminal symbols or format names are denoted by the name of the linguistic construct in italicized type or, in typescript, by the underlined name of the construct.

3.1.3 Symbols in the Language

Symbols which are not in the metalanguage (cf. §§ 3.1.4 and 3.1.5), i.e., symbols other than ::=, |, [, and] are used to denote themselves. This includes digits and upper case letters.

3.1.4 Choice

The symbol | is used to indicate a choice and reads as "or."

3.1.5 Operators

Five operators are used in the metalanguage to simplify metarules. They are list, clist, sclist, option, and Boolean combination. The operators are given in lower case letters; the operand is given within square brackets following the operator.

the list operator: list[ ]

A list contains items separated only by spaces. It may contain only one item, but must not be empty.

the clist operator: clist[ ]

A comma list, or clist, contains items separated by commas. There is no terminal comma. It may contain only one item, but must not be empty.

the sclist operator: sclist[ ]

A semicolon list, or sclist, is like a clist, but has semicolons in places of commas.

the opt operator: opt[ ]

The optional operator indicates an optional element.

the booleancombination operator: booleancombination[ ]

The booleancombination operator expresses a Boolean combination. The logical operators are ¬ (not), & (and), | (or); their precedence is: ¬ before & and & before |. Parentheses override this precedence order.

3.2 EXAMPLE

1.01 tree ::= complex-node opt[< list[tree] >]

Rule 1.01 reads as follows: "a tree consists of a complex-node optionally followed by the following sequence: the character <, a list of trees, i.e., a sequence of at least one tree separated by spaces, and finally the character >."

4.08 sign ::= + [] -

Rule 4.08 reads as follows: "a sign consists of either the character + or the character -."

3.3 COMMENTS IN A GRAMMAR

Expressions within quotation marks (") are not formally part of the grammar. This permits their insertion at any point in the grammar as clarifying remarks or comments for readers of the grammar.

CHAPTER 4

TREES

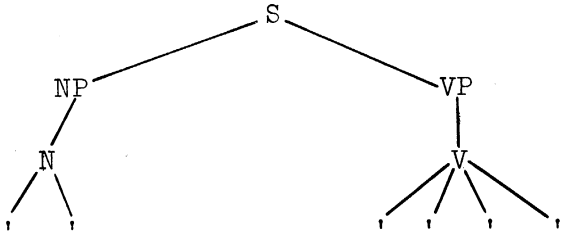
Trees play an essential role in transformational grammars. The input of the phonological component is a tree which contains both labelled and unlabelled nodes. The computer program uses two forms for trees: (1) a linear form for the trees which appear in the description of the grammar, and (2) a tabular form for output. Input trees may have either form; the linear form, however, is more convenient for the input of trees used in the phonological component and we shall restrict the description of input trees to this form.[1] The linear form is defined in the formal metalanguage (definition 1.01 in Appendix B).

4.1 TERMINOLOGY FOR TREES

We define here some of the terms that refer to the relationship between the nodes of a tree and that appear frequently in this report.

The following is an archetype of tree used in the phonological component of a transformational grammar and which we will further define later in this chapter.

(1)



[1] The reader is referred to Friedman et al. (1971, §4), for an alternate input using the tabular form.

This tree contains both labelled and unlabelled nodes. The position of the labelled nodes is represented by their labels: S, NP, N, VP, V; we may refer to a node with the label NP as a node NP. The position of the unlabelled nodes is represented by the character ' (single quote); we may refer to unlabelled nodes as dummy nodes. The root of the tree is the node at the top of the tree, in tree (1) this is the node S. The branches of the tree are lines which connect some of the nodes.

daughter: A node X is a daughter of node Y if a branch connects X and Y and X is immediately below Y. In tree (1) N is a daughter of the node NP.

parent: if X is a daughter of Y, then Y is the parent of X.

immediately dominates: Y immediately dominates X, if Y is the parent of X.

dominates: X simply dominates Y, if there exists a sequence of nodes X_o, X_1, X_2,..., X_{n-1}, X_n such that each immediately dominates its follower, i.e. X_o immediately dominates X_1, X_1 immediately dominates X_2, ..., X_{n-1} immediately dominates X_n and such that X equals X_o and Y equals X_n.

X dominates Y if in the sequence X_o, X_1, X_2, ..., X_{n-1}, X_n none of the nodes has the label S, except, possibly, the extreme nodes X_o and X_n.

sister: X is a sister of Y if X and Y have the same parent.

right sister: X is the right sister of Y, if X is a sister of Y and is immediately to the right of Y in the right-to-left list of daughters of the common parent. In the tree (1) VP is the right sister of node NP.

first daughter: X is the first daughter of Y if X is a daughter of Y and is the left-most daughter in the right-to-left list of daughters of Y.

left sister, last daughter: same definition as previously changing right for left and first for last, mutis mutandis.

subtree: the subtree headed by X consists of all the nodes it simply dominates and the branches connecting these nodes.

terminal: a terminal node is a node without daughters.

4.2 LINEAR FORM FOR TREES

If we use angular brackets to represent the notion "immediately dominates the list," the subtree headed by the node N in tree (1) can be represented as (2) and tree (1) can be itself represented as (3)

(2) N < ' ' >

(3) S < NP < N < ' ' > > VP < V < ' ' ' ' > > >

In this representation, the daughters of a node appear to the right of the node, and are surrounded by angular brackets. The linear representation does not change the right-to-left order of daughters.

The following definitions in the metalanguage define the linear form of trees without complex symbols:

1.01 tree ::= complex-node opt[< list[tree] >]

1.02' complex-node ::= node ▯ dummy-node

1.03 node ::= word ▯ sentence-symbol ▯ boundary-symbol

1.04 sentence-symbol ::= S

1.05 boundary-symbol ::= # ▯ + ▯ =

1.06 dummy-node ::= '

The optional list of trees following node in the definition 1.01 is the list of daughter subtrees of this node. The semantic interpretation of rule 1.01 is that a tree consists of a node, referred to in the rule as complex-node, which may either immediately dominate a list of trees; or be a terminal node.

Rule 1.02' distinguishes two types of nodes: labelled nodes (node) and unlabelled nodes (dummy-node). (Rule 1.02' is a modified version of rule 1.02 to follow.)

Rule 1.03 indicates that the label of a labelled node may be either a word, the label S, or one of the boundary symbols #, + or =. The symbol S is distinguished by its appearance in rule 1.04. It represents the sentence-symbol and may not be used as a word.

4.3 COMPLEX SYMBOLS IN LINEAR TREES

We describe the form and function of complex symbols later in Chapter 6. In this section, we discuss the position of the structure complex-symbol in the structure tree.

In a tree, a complex-symbol may optionally follow any node; a complex-symbol-reference must necessarily follow a dummy-node. A complex-symbol-reference is either a complex-symbol or a complex-symbol-name. The definitions previously given for tree extend as follows:

1.02 complex-node ::= node opt[complex-symbol]

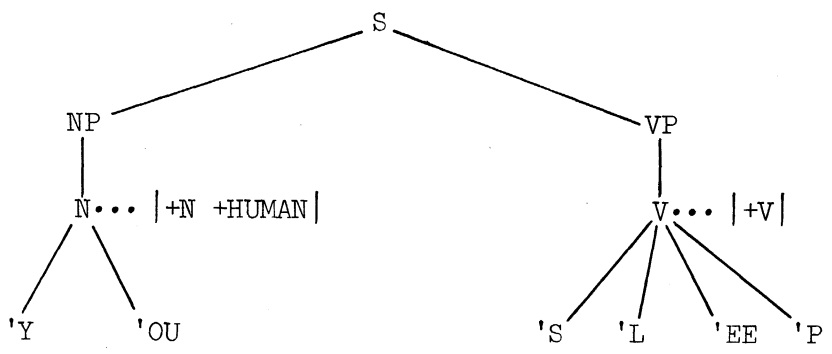
[] dummy-node complex-symbol-reference

1.07 complex-symbol-reference ::=

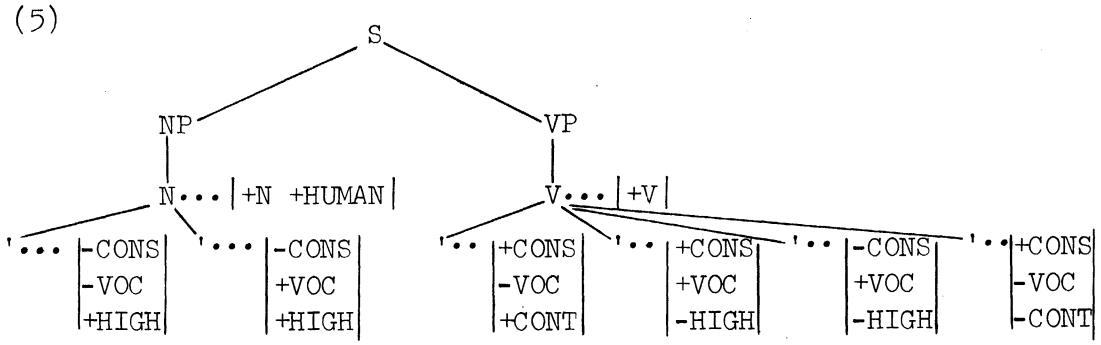
complex-symbol [] complex-symbol-name

Tree (1) was not complete since the dummy nodes are not followed by complex-symbol-references. Trees (4) and (5) are examples of full trees with complex symbols: Tree (4) contains all complex-symbol names.

(4)



(5)



The linear form of trees (4) and (5) is (6) and (7), respectively.

(6) S<NP<N|+N +HUMAN| < 'Y 'OU>> <VP<V|+V|< 'S 'L 'EE 'P>>>>

(7) S < NP < N|+N +HUMAN| < '|-CONS -VOC +HIGH|
'|-CONS +VOC -HIGH| >> < VP < V|+V| < '|+CONS -VOC -HIGH|
'|+CONS +VOC -HIGH| '|-CONS +VOC -HIGH|
'|+CONS -VOC -CONT| >>>>

4.4 SUBSTITUTION FEATURE FOR LINEAR TREES

A substitution facility, defined by the format tree-specification, permits the user to decompose a tree into several subparts, for easier reading of long and intricate trees. This facility is available only for the input of trees. Trees must be input as tree-specifications: that is, they must be terminated by a period.

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

A tree-specification is a tree followed by word tree pairs. A period ends the tree-specification. The semantic interpretation of a tree-specification is best described by using an example:

(8) S < A B < E SUBS1 > C >,

SUBS1 F < S < G H >>.

In (8) the first tree contains the node SUBS1 which is identical to the word of the following word tree pair; this is an indication that the tree F < S < G H >> of this word tree pair replaces the node SUBS1 of the first tree. Thus, the tree specified by (8) is the tree (9):

(9) S < A B < E F < S < G H >>> C >

A tree-specification defines a tree only if the word of each word tree pair occurs once and only once as the node of a previous tree. The word tree pairs in the clist are processed from left to right. The interpretation of each $word_i$ $tree_i$ pair is that in the tree so far constructed, the subtree $tree_i$ replaces the node labelled $word_i$.

A last example of this substitution feature is the specification (10) for tree (4)

(10) S < NP < N1 > VP < V1 > >,

N1 N|+N +HUMAN| < 'Y 'OU >,

V1 V|+V| < 'S 'L 'EE 'P >.

4.5 TABULAR TREES

In the tabular output form tree (5) appears as

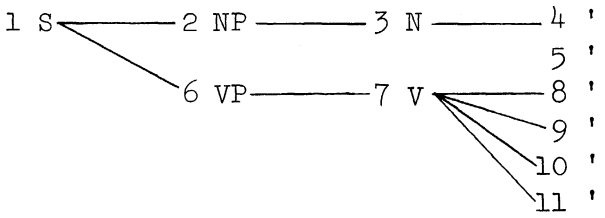
```
TREE READ BY FTRIN
  1 S        2 NP        3 N        4 '
                                     5 '
             6 VP        7 V        8 '
                                     9 '
                                    10 '
                                    11 '

NODE   3 N|+N +HUMAN|
NODE   4 '|-CONS -VOC +HIGH|
NODE   5 '|-CONS +VOC +HIGH|
NODE   7 V|+V|
NODE   8 '|+CONS -VOC +CONT|
NODE   9 '|+CONS +VOC -HIGH|
NODE  10 '|-CONS +VOC -HIGH|
NODE  11 '|+CONS -VOC -CONT|
          ' ' ' ' ' '
```

A tabular tree contains four parts: (1) a title, here TREE READ BY FTRIN, which indicates that the input of the tree was linear; (2) a representation of the tree in which each node has an index; (3) a list of the complex symbols and of the nodes to which they are attached; and (4) a list of the terminal nodes separated by blanks.[1]

In the tabular representation of the tree, each node of its index appear at the beginning of a field of twelve character width. The tabular representation is a mirror reflection of the representation we described earlier, followed by a deformation which brings the left-most daughter of a node on the same line and in the field immediately

[1] In the syntax program (Friedman et al., 1971) the terminal nodes are separated by blanks unless they are special characters. In the phonology program the treatment of all terminal symbols is uniform.

to the right of the node, and the daughters to the right below the left-most daughter. In the following figure we reintroduce the branches which are not expressed in the tabular output (because they are redundant in a fixed field representation) to show the relation between the tabular representation and the representation defined previously:



4.6 SUBSTITUTION FEATURE FOR TABULAR TREES

If the depth of a tree exceeds the maximum number of fields allowed for output a substitution feature analogous to the previous feature defined for linear trees permits the representation of the whole tree.

```
1 S        2 NP        3 SUBO
           6 VP        7 SUB1

REPLACEMENT FOR SUBO
3 N        4 '
           5 '

REPLACEMENT FOR SUB1
7 V        8 '
           9 '
          10 '
          11 '
```

This representation indicates that the node SUBO in the first tree stands for the subtree listed under "REPLACEMENT FOR SUBO," the same remark applies to SUB1.

CHAPTER 5

COMPLEX SYMBOLS

Complex symbols play an essential role in the phonological grammar program because in our interpretation a phonological segment is the complex symbol of a terminal dummy node. We distinguish two types of complex symbols: (a) tree complex symbols; and (b) operator complex symbols. The input trees contain only tree complex symbols and the rules of the grammar preserve their character, i.e., they transform tree complex symbols into tree complex symbols. The operator complex symbols appear in the description of rules and participate in complex symbol operations, viz., comparisons and changes.

5.1 DESCRIPTION OF COMPLEX SYMBOLS

A complex symbol is an unordered set of features[1] which can take different values. The set of features which can appear in a complex symbol is defined in the lexicon.

6.01' conversion-lexicon ::= PHONLEX feature-lexicon $ENDPHON

6.02 feature-lexicon ::= FEATURES list[feature]

4.09 inherent-feature ::= word

We allow features to appear with or without a preceding value. A value

[1]These features correspond to the inherent-features of the syntactic program; we may sometimes refer to them as inherent-features.

can be a binary value, i.e., one of the two signs + (plus) or - (minus), or a n-ary value, i.e., an integer 0, 1, 2, 3,..., or else one of three operator values ¬, prefix and *.

4.01 complex-symbol ::= | opt[list[feature-specification]] |

An example of a complex-symbol is |+CONS -VOC -CONT|. Note that the definition allows a complex symbol to be empty.

4.02 feature-specification ::= opt[value] feature

In the example above, the feature CONS has the value + and VOC has the value -; an example of a complex-symbol with a feature-specification without a value is |+CONS NASAL|

4.03 value ::= sign ▯ integer ▯ - ▯ (prefix) ▯ *

4.07 sign ::= + ▯ -

5.1.1 Tree Complex-Symbols

Tree complex-symbols are complex-symbols in which feature-specifications have either no value or else a binary or n-ary value. If a tree complex-symbol does not contain a feature, it is said to be unspecified for this feature; if it contains a feature without a value, it is said to be marked for this feature.

5.1.2 Operator Complex-Symbols

In addition to the feature-specifications allowed for tree

complex-symbols, an operator complex-symbol may also contain feature-specifications with one of the three operator values. An operator complex-symbol is said to be unspecified and marked for certain features in the same conditions as tree complex-symbols. It is said to be unmarked for a feature if it contains a feature-specification with this feature preceded by the symbol ¬. Prefixes in operator complex symbols correspond to Greek letter variable in phonology, i.e., they are variables over binary and n-ary values; we distinguish sign-prefixes, where the variable ranges over binary values and integer-prefixes where the variable ranges over integer values.

4.04 prefix ::= sign-prefix [] integer-prefix

4.05 sign-prefix ::= opt[-] variable

4.06 integer-prefix ::= variable opt[sign integer]

4.08 variable ::= word

Examples of complex-symbols containing a sign-prefix are:

|(ALPHA) CONS|

|(-BETA) NASAL|

Examples of complex-symbols containing an integer-prefix are

|(ALPHA) STRESS|

|(ALPHA+1) STRESS|

The name for variables can be any word; the same name may be used in

different rules without interference.

5.1.3 Conventions

As we shall observe later unmarked features in operator complex-symbols correspond to unspecified features in tree complex-symbols; for this reason, the terminology "unmarked feature" can be extended to tree complex-symbols to mean "unspecified feature" (this convention is possible because tree complex-symbols do not contain feature specifications with the value $-$).

5.2 COMPLEX SYMBOL OPERATIONS

There are two types of operations for complex-symbols: comparisons and changes. The first argument of an operation is always an operator complex-symbol and the second a tree complex-symbol. A tree complex symbol can also be interpreted as an operator complex-symbol and used as the first argument of any operation.

5.2.1 Complex Symbol Comparisons

Testing structural descriptions and restrictions in a transformation, as well as testing the applicability of a redundancy rule requires comparison of complex symbols. The basic comparisons in the program are equality, nondistinctness and inclusion.

The comparison of two complex symbols is defined in terms of individual feature comparisons. The comparison of two complex symbols succeeds for each feature. For instance, two complex symbols are equal if

for each feature in the lexicon both complex symbols are either unspecified for that feature, or else have the same value, or else are marked for that feature. A special case must be made, however, for the three values which appear only in operator complex symbols. The general interpretation of ¬ and * is that the feature in the corresponding tree complex symbol must be, respectively, unspecified and specified for some value (binary or n-ary). A prefix is either assigned or unassigned; a prefix is assigned either a binary or an n-ary value, thus the two complex symbols |(ALPHA) CONS| and |+CONS| are equal if ALPHA is assigned the value +, and unequal, if ALPHA is assigned the value - or 3. Prefixes are assigned values during comparisons. If a prefix remains unassigned at the end of a comparison, the comparison fails. A comparison can be formally defined by a matrix which describes whether the result of the comparison is true for each of the feature specifications in the operator complex symbol and the tree complex symbol.

operator complex symbol \ tree complex symbol	unspf'd	+	-	1	2	3	⋮	marked
¬								
unspf'd								
+								
-	F	T						
1								
2								
3								
...								
marked								
*								
prefix								

In the table above, the comparison of two specifications for a feature succeeds if the operator complex symbol has the value - for this feature and the tree complex symbol has the value + for the same feature; the comparison fails, if the tree complex symbol is unmarked for the feature.

5.2.2 Value Assignment

A prefix is said to be assigned when its value has been computed. We first describe how the value of a prefix depends on the value assigned to its variable.

Given an integer-prefix P and an integer i, the value of P for the assignment i is the value obtained by substituting i for the variable

of P and evaluating the resulting arithmetic expression. For example, prefix (ALPHA+1) with integer assignment 3 has the value 4, and prefix (ALPHA-2) with assignment 2 has the value 0. If the substitution results in a negative value, the value is taken by definition to be 0. For instance, the prefix (ALPHA-2) with assignment 1 has the value 0.

Given a sign-prefix P and a sign s, the value of P for the assignment s is obtained by substituting s for the variable in P and evaluating the resulting logical expression. For example, sign-prefix (ALPHA) with assignment + has the value +, and sign-prefix (-ALPHA) with the assignment - has the value +.

The value assigned to the variable must be of the appropriate type; it is meaningless to assign an integer value to a prefix unless it is an integer-prefix, or conversely, a sign value unless it is a sign-prefix.

Whenever a previously unassigned prefix P is compared to a value v, the variable of P is assigned a value which makes the value of P equal to v, if this is possible. For example, a comparison of |(ALPHA)VOC| with |+VOC| assigns + to ALPHA, if it is otherwise unassigned. If there is no assignment which makes P equal to v, the variable and the prefix remain unassigned. For example, ALPHA remains unassigned in the comparison of |(ALPHA+2)STRESS| with each of the tree complex-symbols |+VOC -CONS|, |-STRESS|, and |1STRESS|.

5.2.3 Equality

Two complex symbols are equal if for each feature in the lexicon, both complex symbols are (a) unspf'd for that feature, or (b) marked for that feature, or (c) have the same value (after assignment). The values * and ¬ are not meaningful for equality. The following matrix formally defines equality of complex symbols (* and ¬ are included for completeness of the definition).

		tree complex symbol				
	Equality	unspf'd	+	-	integer I_2	marked
operator complex symbol	¬	T	F	F	F	F
	unspf'd	T	F	F	F	F
	+	F	T	F	F	F
	-	F	F	T	F	F
	integer I_1	F	F	F	$\Delta_{1,2}$	F
	marked	F	F	F	F	T
	*	F	F	F	F	F
	prefix	F	X	X	X	F

$\Delta_{1,2}$ is T if integer I_1 equals integer I_2, F otherwise. X behaves like the corresponding binary or n-ary value if it is assigned a value. It is equal to F, if it cannot be assigned a value.

5.2.4 Nondistinctness

An operator complex symbol is nondistinct from a tree complex symbol, if for each feature in the lexicon,

(a) the feature is unspf'd in the operator complex symbol.

(b) the feature has the value ¬ in the operator complex symbol and is not marked in the tree complex symbol.

(c) the feature has a binary or an n-ary value (after assignment) in the operator complex symbol and is either unmarked or has the same value in the tree complex symbol.

(d) the feature is marked in both complex symbols.

(e) the feature has the value * in the operator complex symbol and the tree complex symbol is not marked for this feature (in this case, therefore, the value * corresponds to the notion of "any binary or n-ary value").

Informally, this means that two complex symbols are nondistinct if they have the same values for the corresponding features, or one is unmarked for that feature. The following matrix explicitly defines nondistinctness.

operator complex symbol	tree complex symbol: Nondistinctness	unspf'd	+	-	integer I_2	marked
	¬	T	T	T	T	F
	unspf'd	T	T	T	T	T
	+	T	T	F	F	F
	-	T	F	T	F	F
	integer I_1	T	F	F	$\Delta_{1,2}$	F
	marked	F	F	F	F	T
	*	T	T	T	T	F
	prefix	T	X	X	X	F

$\Delta_{1,2}$ and X have the same meaning as previously.

5.2.5 Inclusion

An operator complex symbol is included in a tree complex symbol, if for each feature in the lexicon,

(a) the feature is unspf'd in the operator complex symbol.

(b) the feature has the value ¬ in the operator complex symbol and it is unmarked in the tree complex symbol.

(c) the feature has a binary or n-ary value (after assignment) in the operator complex symbol and has the same value in the tree complex symbol.

(d) the feature is marked in both complex symbols.

(e) the feature has the value * in the operator complex symbol and the tree complex symbol is neither marked

nor unmarked for this feature (in this case, the value * corresponds to the notion of "any binary or n-ary value").

Informally, this means that an operator complex symbol is included in a tree complex symbol if both have the same values for those features which are not unmarked in the operator complex symbol. The following matrix explicitly defines inclusion.

operator complex symbol \ tree complex symbol	unspf'd	*	-	integer I_2	marked
¬	T	F	F	F	F
unspf'd	T	T	T	T	T
+	F	T	F	F	F
-	F	F	T	F	F
integer I_1	F	F	F	$\Delta_{1,2}$	F
marked	F	F	F	F	T
*	F	T	T	T	F
prefix	F	X	X	X	F

where $\Delta_{1,2}$ and X are defined as previously.

5.2.6 Complex Symbol Changes

An operator complex symbol containing prefixes successfully participates in a change only if its prefixes have been assigned. The basic changes are merging, erasing, and saving. As in the case of comparisons, each change applies to one feature at a time.

5.2.7 Merging

Each of the features which are specified in the operator complex symbol,

(a) is marked in the tree complex symbol, if it is marked in the operator complex symbol.

(b) receives the same value as in the operator complex symbol if the operator complex symbol has a binary or n-ary value for this feature.

(c) is erased from the tree complex symbol, if it has the value ¬ in the operator complex symbol.

The value * in the operator complex symbol is not meaningful.

Informally, the change merges the feature specification of the operator complex symbol into the tree complex symbol. The following matrix explicitly defines merging.

		tree complex symbol				
	Merge	unspf'd	+	-	integer I_2	marked
operator complex symbol	¬	U	U	U	U	U
	unspf'd	U	+	-	I_2	U
	+	+	+	+	+	+
	-	-	-	-	-	-
	integer I_1	I_1	I_1	I_1	I_1	I_1
	marked	M	M	M	M	M
	*	U	+	-	I_2	U
	prefix	X	X	X	X	X

The value U means unspf'd

M means marked

X means that the prefix behaves like its value if it is assigned a value, and that the operation is not defined otherwise.

5.2.8 Erasing

Each of the features which are not unspf'd in the operator complex symbol,

(a) becomes unmarked in the tree complex symbol, it if has the value * in the operator complex symbol.

(b) becomes unmarked in the tree complex symbol, if the two complex symbols have the same binary or n-ary values or both marked for this feature.

The operation is meaningless for the value ¬.

Informally, the change deletes the feature specifications designated by the operator complex symbol from the tree complex symbol (the value * here means "any binary or n-ary value, or marked"). The following matrix explicitly defines erasing.

operator complex symbol \ tree complex symbol	unspf'd	+	-	integer I_2	marked
¬	U	+	-	I_2	M
unspf'd	U	+	-	I_2	M
+	U	U	-	I_2	M
-	U	+	U	I_2	M
integer I_1	U	+	-	$U_{1,2}$	M
marked	U	+	-	I_2	U
*	U	U	U	U	U
prefix	X	X	X	X	X

The value U means unspf'd

M means marked

X means that the prefix behaves like its value if it is assigned, and that the operation is not defined otherwise

$U_{1,2}$ means U if I_1 equals I_2, means I_2 otherwise

5.2.9 Saving

Each of the features which are unspf'd in the operator complex symbol and the features which have different values in the two complex symbols, unless one of these values is * in the operator complex symbol, become unspf'd in the tree complex symbol. The value ¬ is meaningless for this operation.

Informally, the change deletes all the feature specifications of

the tree complex symbol, unless they are specified in the operator complex symbol (the value * here means "any binary or n-ary value, or marked"). The following matrix explicitly defines saving.

		tree complex symbol				
	Save	unspf'd	+	-	integer I_2	marked
operator complex symbol	¬	U	U	U	U	U
	unspf'd	U	U	U	U	U
	+	U	+	U	U	U
	-	U	U	-	U	U
	integer I_1	U	U	U	$U_{1,2}$	U
	marked	U	U	U	U	M
	*	U	+	-	I_2	M
	prefix	X	X	X	X	X

The value U means unspf'd

M means marked

X means that the prefix behaves like its value if it is assigned a value, and that the operation is not defined otherwise

$U_{1,2}$ means U if I_1 is different from I_2, means I_1 otherwise.

5.2.10 Moving

Moving is a composite change obtained by composition of the changes Saving and Merging. The change n MOVEF m k moves the feature specifications common to the operator complex symbol n and the tree

complex symbol m into the tree complex symbol k. It is equivalent to n SAVEF m, m MERGEF k, except that m is not affected during the change.

5.3 REDUNDANCY RULES

The use of redundancy rules permits (a) the input of minimally specified trees, and (b) the reduction of the specifications of rules. For instance, all vowels of English are voiced and therefore need not be specified as such in the input trees and also if a rule creates a new vowel, it is unnecessary to specify that this new segment is voiced. For that reason, the redundancy rules specified in the lexicon apply at any time in the entire process of the derivation when a complex symbol is modified. This is what Stanley (1967) refers to as "everywhere rules."

The metarules defining redundancy rules are:

7.01 everywhere-rules ::= RULES clist[redundancy-rule]

7.02 redundancy-rule ::= complex-symbol-reference =>
complex-symbol-reference

Example of redundancy rules are:

|+VOC| => |+SON|

|+VOC -CONS| => |-ANT -STRID +CONT +VOICE -LATERAL|

|-LOW (ALPHA)BACK| => |(ALPHA)ROUND|

The redundancy rules apply after the input of the tree and

everytime a complex symbol is modified in the derivation. Any complex symbol which includes the complex symbol on the left-hand side of the redundancy rule is expanded to contain the complex symbol on the right-hand side of the redundancy rule; i.e., the complex-symbol on the right-hand side of the redundancy rule merges into any complex symbol which includes the left-hand side complex symbol of the redundancy rule.

The redundancy rules of the grammar are not ordered. The process of expansion of the complex symbols terminates only when no further application is possible.

5.4 MARKEDNESS CONVENTIONS

The program distinguishes two special prefixes, the prefix U and the prefix M which are used for the representation of the marking convention rules.

Unlike other prefixes which match a tree complex symbol when the feature associated with the prefix has a sign or an integer value, these two prefixes match only tree complex symbols which are either unmarked or marked for the feature associated with the prefix. The prefix U is assigned the value + if the complex symbol is unmarked for the associated feature and the value - if it is marked. The prefix M is assigned the value + if the complex symbol is marked for the associated feature and the value - if it is unmarked.

In our implementation, there can be only one occurrence of either

of these marking prefixes in a complex symbol, as it appears that markedness convention rules require only one marking prefix.

5.4.1 Example

The complex symbol |+SEG (U)VOC| matches the complex symbol |+SEG| and U is assigned the value +; it matches also the complex symbol |+SEG VOC| and U is assigned the value -.

The tables describing the comparisons can be further specified for the "marking" prefixes as follows:

	unspf'd	+	-	integer	marked
prefix U or M	T	F	F	F	T

CHAPTER 6

CONVERSION OF DUMMY NODES

A dummy node is completely characterized by its complex-symbol. Complex-symbols are difficult to write and even to read; this suggests giving a name to the most frequent of these complex-symbols and referring to them by name, thus simplifying the description of trees and of rules. On input each name must be converted into its associated complex-symbol, and on output a name must must be found for each complex-symbol whenever this is possible. We thus distinguish two types of conversions for names of complex-symbols and of complex-symbol combinations: input conversion and output conversion.

6.01 conversion-lexicon ::= PHONLEX feature-lexicon
opt[input-name-lexicon]
opt[output-name-lexicon]
opt[diacritic-lexicon]
$ENDCON

This rule of the metalanguage defines the conversion-lexicon as the set of feature-lexicon and three optional lexicons: input-name-lexicon, output-name-lexicon and diacritic-lexicon. The diacritic-lexicon is used in the output conversion.

6.1 INPUT CONVERSION

The input-name-lexicon contains the list of the input-name-definitions:

6.03 input-name-lexicon ::= VARIABLE clist[input-name-definition].

An input-name-definition associates a name with a given complex symbol.

6.06' input-name-definition ::=

complex-symbol-name = complex-symbol

6.09 complex-symbol-name ::= word

Once a name has been given to a complex symbol in the conversion lexicon, this name can be used instead of the complex-symbol itself in most parts of the grammar. This is why the term complex-symbol-reference in the metalanguage represents either a complex-symbol or a complex-symbol-name.

1.07 complex-symbol-reference ::=

complex-symbol ▯ complex-symbol-name

Examples of input-name-definitions are

V = |-CONS +VOC|

L = |+CONS +VOC|

G = |-CONS -VOC|

CONS = |+CONS -VOC|

GR = |(ALPHA)CONS (ALPHA)VOC -ANT|

Phonological rules are susceptible of further simplification if boolean-combinations of complex symbols also have names. For instance, a consonant is a segment which is not a vowel. Definition 6.06 extends 6.06' to permit such naming.

6.06 input-name-definition ::=

complex-symbol-name = complex-symbol

[] input-name = booleancombination[complex-symbol-reference]

Examples of input-name-definitions with input-names are

C = ¬V & |+SEG|

C = |+CONS| | |-CONS -VOC|

C = L | G | CONS

The first example defines a consonant C as a segment |+SEG| which is not a vowel V. The last example defines a consonant C as being either a liquid L or a glide G or a "restricted consonant" CONS.

A complex-symbol-name may appear in the definition of an input name only if it has been previously defined.

6.2 OUTPUT CONVERSION

Output conversion takes place just prior to tree output. It does

not modify the structure of the tree; it only gives a printed version of the tree which is easier to read. It operates in two steps, first it finds the principal symbol corresponding to the dummy node and complex-symbol which are to be converted, and then finds the diacritics modifying the principal symbol.

6.2.1 Principal Symbol

The output-name-lexicon contains the list of the output-name-definitions which define the principal symbols of the output conversion.

6.04 output-name-lexicon ::=

PHONUNIT clist[output-name-definition].

An output-name associates a name to a given complex-symbol-reference.

6.07 output-name-definition ::=

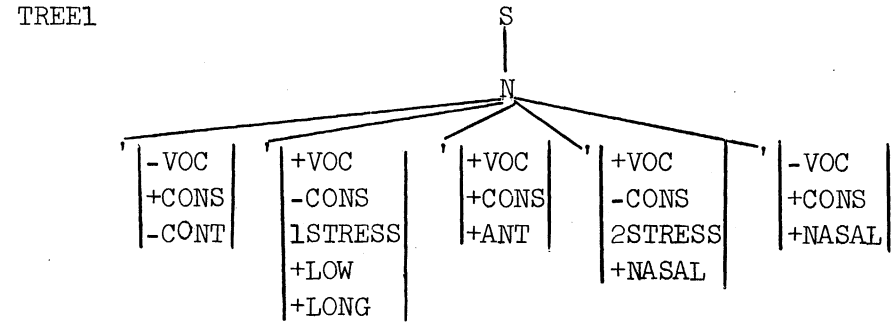
output-name = complex-symbol-reference

6.10 output-name ::= symbol

Example of output-name-lexicon

```
PHONUNIT    A  =  |+VOC -CONS +LOW|,
            E  =  V,
            N  =  |+CONS -VOC +NASAL|,
            B  =  |-VOC +CONS -CONT|,
            L  =  L.
```

The complex symbol corresponding to the output-name must be a tree complex symbol with binary or n-ary valued features. Before a tree is output, each of the complex symbols associated to its dummy nodes is compared with the complex symbol associated with the output-names. If the complex symbol of an output name definition is included in the complex symbol of a dummy node, this output name replaces the dummy node in the tree. The features of the complex symbol in the tree which are not specified in the complex symbol of the output name are listed after the terminal string. For example let us analyze the conversion of the tree TREE1.



Without a conversion lexicon, this tree has the following output:

```
TREE1
  1  S       2N        3'
                       4'
                       5'
                       6'
                       7'
NODE 3 '
          |-VOC +CONS -CONT|
NODE 4 '
          |+VOC -CONS 1STRESS +LOW +LONG|
NODE 5 '
          |+VOC +CONS +ANT|
```

```
NODE 6 '
          |+VOC -CONS 2STRESS +NASAL|
NODE 7 '
          |-VOC +CONS +NASAL|
```

The same tree, but this time using the output-name-lexicon above, has the following output:

```
TREE1
  1 S        2 N        3 B
                        4 A
                        5 L
                        6 E
                        7 N
          B A L E N

  4 A |1STRESS +LONG|
  5 L |+ANT|
  6 E |2STRESS +NASAL|
```

It is very important that the definitions in the output-name-lexicon be properly ordered, since it is the first output whose complex symbol is included in the complex symbol of the dummy node which is taken as the symbol for output. In the previous example, if the definitions of A and E are inverted, the terminal string is BELEN, since E is defined as any vowel and is compared first.

6.2.2 Diacritics

The features which are left after the principal conversion can be specified as diacritics after the principal output symbol. The diacritic-lexicon contains the list of the diacritic-definitions.

6.05 diacritic-lexicon ::= DIACRIT

clist[diacritic-definition]

A diacritic-definition associates a name to a given complex-symbol-reference.

6.08 diacritic-definition ::=

diacritic = complex-symbol-definition

6.11 diacritic ::= symbol

Example of diacritic-lexicon

```
DIACRIT  >   = |+NASAL|,
         :   = |+LONG|,
         /1/ = |1STRESS|,
         /2/ = |2STRESS|.
```

The complex symbol in the definition of a diacritic must be a tree complex symbol with binary of n-ary values features. The complex symbol of each diacritic is compared for inclusion in the complex symbol of the dummy nodes in the tree; if some feature specifications were specified in the complex symbol of the principal symbol and of previous diacritics, they are not available and are assumed to be absent from the complex symbol being converted at this stage. If the complex symbol of a diacritic is included in the complex-symbol of the tree, the diacritic is written immediately after the last character of the mode. In the previous example of conversion, if we add the diacritic lexicon, the printed output of TREE1 becomes

TREE1

1 S 2 N 3 B
4 A:/1/
5 L
6 E>/2/
7 N

B A:/1/ L E>/2/ N

5 L |+ANT|

In the example node 7 is converted to N; the diacritic > for nasality does not appear after the principal symbol because the feature +NASAL was part of the definition of N. The diacritics appear in the order of their definition in the lexicon. If the definitions of the diacritics: and /1/ are inverted in the lexicon, the terminal string becomes B A/1/: L E>/2/ N.

CHAPTER 7

ANALYSIS, RESTRICTION, AND CHANGE[1]

A phonological rule typically specifies modification, deletion, or introduction of segments in a given environment. In the tree format of the program the representation of a segment is a dummy node followed by its complex symbol; the interpretation of a phonological rule becomes the modification of a complex symbol, the deletion of a node, or the introduction of a node, i.e., a tree operation. We refer to tree operations as structural-changes. The description of the environment in which the change takes place is the structural-description of the rule. The structural description is aimed at singling out specific nodes in the tree to which the changes apply. Consider, for instance the rule of penultimate stress placement, which places a stress on the penultimate vowel of a word. It may be stated as a structural description, which singles out the penultimate dummy node with a complex symbol with the features |+VOC -CONS|, and a structural change, which consists of merging the feature |1STRESS| into the complex symbol attached to the node singled out in the structural description. Actually, a structural description contains two parts, a structural analysis and, optionally, a restriction. The metarule corresponding to this decomposition is

[1] Except for bounded-skips and node-names, the analysis of phonological rules is essentially the same as in Friedman et al. (1971,§8), to which the reader is referred for a complete description of transformations.

2.01 structural-description ::=

structural analysis opt[, WHERE restriction]

In this chapter, we treat structural analyses, restrictions, and structural changes.

7.1 STRUCTURAL ANALYSIS

Before we describe the format for structural-analysis and its interpretation, we give three examples of structural analyses which we use as examples throughout this section. Analyses (1) is associated with stress placement, analyses (2) and (3) with diphthongization.

(1) N < % 1 'V ('O'C 'V) 'O'C # >

(2) % 1'|-CONS +VOC +TENSE (ALPHA)BACK| %

(3) % 1'A ('R'C, 'R #, 'L'M #, #) %

7.1.1 Formal Description

2.02 structural-analysis ::= list[term]

2.03 term ::= opt[integer] structure

▯ opt[integer] choice ▯ skip

2.04 structure ::= complex-element opt[opt[¬] opt[/]

< structural-analysis >]

2.05 complex-element ::= element opt[complex-symbol]

▯ dummy-node node-name

2.06 element ::= node ▯ * ▯ _

2.07 node-name ::= complex-symbol [] complex-symbol-name

[] input-name

2.08 choice ::= (clist[structural analysis])

2.09 skip ::= % [] bounded-skip

2.10 bounded-skip ::=

' integer opt[, integer] dummy-node node-name

Terms

The structural analysis (1) is a single term, it is also a structure. The substring (4) of the structural-analysis (1) is a structural-analysis.

(4) % 1 'V ('O'C 'V) 'O'C #

It contains five terms: %, 1'V, ('O'C 'V), 'O'C and #. The first and the fourth of these terms are skips, the fourth being a bounded-skip. The third term is a choice containing the two structural-analyses. 'O'C and 'V which are also, respectively, a bounded-skip and a complex-element. The second term of (4) is of the form integer structure; this structure is a dummy-node followed by the node-name V.

Numbering of terms

The definition of terms allows them to contain integers. These integers are indices which single out some nodes in the tree. These indices appear in the definition of changes and in restrictions. Skips

are terms, but are not numbered. To insure proper indexing of the nodes in a tree, two further constraints restrict the use of integers in terms: first, the same integer may precede two terms, only if one of those terms must be null in any analysis, and second, two different integers may not govern the same term as in 2(3'C,4'V) or 1(2'C).

Structures

The structural-analysis (1) is a structure. The complex-element N is an element and also a node, the string (4) is the structural-analysis of this structure.

Skips

There are two types of skips: % and bounded-skips. The skip % replaces the variables used in conventional notations. Bounded-skips replace sequential variables of the type C_i^j which represent strings of at least i and most j consecutive segments C:

'0'C stands for C_o

'0,3'C stands for C_o^3

Skips need not correspond to single subtrees and connot be numbered; this prevents their indexation and their use in structural-changes and restrictions.

Two other constraints restrict the use of skips: first, in a structural-description two adjacent terms of a structural analysis may

not be both skips and second, each structural-analysis in the clist [structural-analysis] of a choice must contain at least one term which is not a skip. The first constraint is partially justified by the following equivalences:

% 'O'C, 'O'C %, and % 'O'C % are all equivalent to %.

% '2,3'C is equivalent to % 'C 'C.

'2,3'C % is equivalent to 'C 'C %.

Complex-elements

The most common complex-element is a dummy-node followed by a node-name. A node-name can be a complex-symbol, a complex-symbol-name or an input-name; the last two are also words. Therefore any word following a dummy-node is interpreted as a complex-symbol-name or an input-name (this applies also to trees). It is for this reason that a dummy-node to which no complex-symbol is attached must be followed by an empty complex-symbol; it prevents the next word from being analyzed as a node-name. Another type of complex-element is an element optionally followed by a complex-symbol.

Elements

The most common element is a node. The element * is an unspecified single node, i.e., a variable over a single node. The element _ (underline) may appear only within the structural-description of a simple-rule where it indicates the position at which a change takes

place. We study these simple rules later and ignore the underline in our discussion until then. There may be several underlines in a structural-analysis, it must be the case, however, that exactly one underline be nonnull in any analysis.

Choice

There are two types of choices: options and collections. In an option, the clist of the structural-analysis is limited to one element; for example ('C) or ('O'C 'V). It corresponds to the use of optional parentheses in phonological descriptions. In a collection, the clist contains at least two elements; for example ('F'C, 'R #, 'L'M #, #). It corresponds to grouping by curly brackets in phonological descriptions:

$$\left\{\begin{array}{l} \text{rC} \\ \text{r\#} \\ \text{lm\#} \\ \text{\#} \end{array}\right\} .$$

7.1.2 Analyzability

Analysis is a mechanism which compares a tree and a structural-analysis and, if the comparison succeeds, indexes some nodes in the tree with the integers appearing in the terms of the structural-analysis. We describe here the analyzability of a tree with respect to a structural-description, that is the conditions under which a match between a tree and a structural-description is possible; we limit this description to structural-descriptions without restrictions and without

prefixes in complex-symbols or in the definition of their complex-symbol-name and input name. These cases are studied later.

The conditions of analyzability not only describe whether a tree is analyzable with respect to a structural-description, but also indicate how many possible analyses of the tree there are; they do not, however, indicate any relation between these analyses, these relations are described later in the description of scans.

The indexing of nodes is straightforward, once a match has been found. The integer of a term by definition governs this term; if an integer governs a term, and this term contains a structure, the integer also governs the element or the dummy-node of this structure; if the term contains a choice, the integer also governs the first term of each of the structural-analyses in the clist of the choice. By recursive application of the relation "governs," an integer governs a list of elements and/or dummy-nodes. For instance the integer 3 governs the dummy-node of the complex-element 3'C, the dummy-node of the choice 3('C) and all the dummy-nodes of the choice # 3('R, 'L, 'N). In an analysis, the integer of a term indexes the node of the tree which matches one element or dummy-node of the structural-description governed by this integer. For example the integer 3 indexes the unlabelled node in the tree which matches the dummy-node of 3'C, 3('C) or one of the dummy-nodes of the collection 3('R, 'L, 'N).

We describe analyzability for structural-analyses which are more and more complex, until the description is general.

(a) The terms of the structural-analysis are all elements. The structural-analysis is simply a list of elements, for example (5).

(5) * N VP

The comparison between a tree and the structural description succeeds if there is a one-to-one match of the tree nodes with all the structural-description elements and such that:

1.a The element * matches exactly one node of the tree, whether this node is labelled or not, and independently of its label if it is labelled.

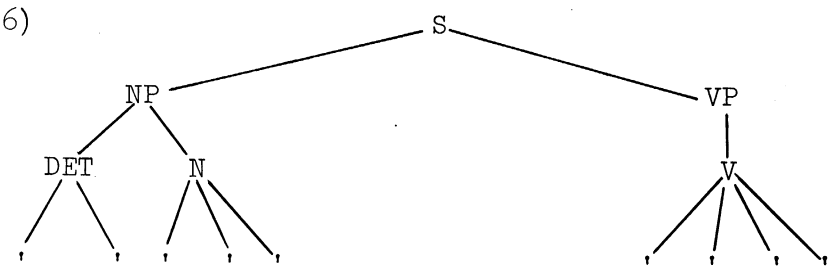
1.b An element, which is a node, matches exactly one labelled node in the tree whose label is the same as the node.

2. Each terminal node in the tree matches an element or a dummy-node of the structural description or is dominated by a node which does.

3. The right-to-left order of the elements and dummy-nodes in the structural-description and the right-to-left order of the nodes in the tree which they match is identical.

The structural description (5) matches the tree (6).

(6)



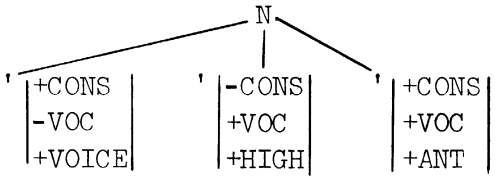
In this match, the element * matches the node labelled DET in the tree, the element N matches the node labelled N in the tree and the element VP matches the node labelled VP in the tree.

(b) The terms of the structural analysis are all complex-elements. The procedure is identical to the preceding, but if an element is followed by a complex-symbol in the structural-analysis, the tree node it matches must have a complex symbol which includes the complex-symbol of the structural-analysis. If the complex-element is the sequence dummy-node node-name, it matches an unlabelled node of the tree which has a complex symbol which meets the conditions of the node-name. When the node-name is a complex-symbol or a complex-symbol-name this means that the complex symbol attached to the unlabelled node includes this complex-symbol or the complex symbol corresponding to the complex-symbol-name. When the name-node is an input-name, each of the complex symbols corresponding to the complex-symbol-references appearing in the booleancombination defining the input-name are tested for inclusion in the complex symbol of the unlabelled node; this complex symbol meets the conditions of the input-name if the booleancombination of the values of the test for inclusion is true.

The structural analysis (7) matches the tree (8).

(7) 'C 'V '|+CONS +VOC|

(8)



where V is the complex-symbol-name defined by

$$V = |-CONS\ +VOC|$$

and C the input-name defined by

$$C = \neg V$$

(c) The terms of the structural analysis are complex symbols or skips. The skip % matches not only a single node, but any string of adjacent nodes, including a string of zero nodes, in which case the skip is said to be null. A bounded-skip is of the form 'n,m' node-name or 'n'node-name. In these two forms n is the lower-bound of the bounded-skip, in the first form, m is the upper bound of the bounded-skip. A bounded-skip matches a string of adjacent unlabelled nodes such that the complex symbol of each of these unlabelled nodes meets the conditions of the node-name and such that the number of nodes matched is at least equal to its lower-bound, and, if the bounded-skip has an upper bound, at most equal to its upper bound. Note that 'n,n' node-name, not 'n'node-name, expresses "precisely n."

(d) Terms of the structural analysis are complex-segments, choices and skips. If the choice is an option, i.e., the clist of the choice contains only one structural-analysis, then the structural-analysis

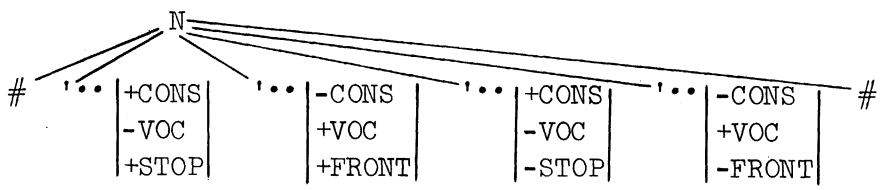
containing this option matches any tree that would be matched by a similar structural description where the choice is absent or replaced by its own structural-analysis.

If the choice is a collection, i.e., the clist of the choice contains several structural-analyses, the structural-analysis containing this option matches any tree that would be matched by a similar structural description with one of the structural-analyses of the choice in place of the choice.

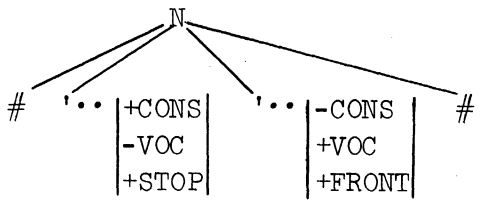
The structural-analysis (9) matches trees (10) and (11).

(9) % 'V ('O'C 'V) 'O'C #

(10)

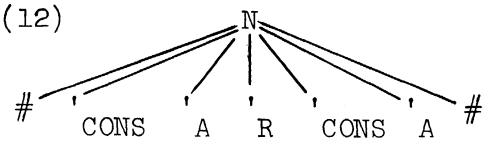


(11)

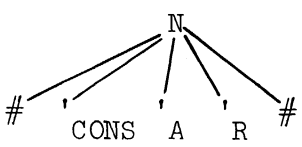


The structural-description (3) matches trees (12), (13), (14), and (15). (In these trees the complex symbols have been written as complex-symbol-names.)

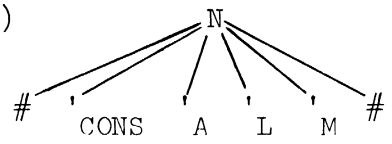
(12)



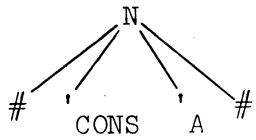
(13)



(14)



(15)



(e) General case. In the general case a complex-element may be followed by a bracketed structural-analysis. The four types of brackets are: (1) < ... > , (2) / < ... > , (3) ¬ < ... >, and (4) ¬ / < ... >. A bracketed structural-analysis is a "sub-analysis" of the complex-element it follows. This means that the complex-element not only matches a node and its complex symbol, but also that the bracketed structural-analysis "matches" the subtree headed by this node. The subanalysis differs slightly from an ordinary analysis of a tree. The top node of the subtree, which has already been matched to the complex-element, is not allowed to match any term within the subanalysis. Otherwise, subanalysis is primarily a recursive application of the definition of analysis. The exact requirement of the subanalysis depends on the type of bracketing. For bracketing (2), the analysis is made in the usual sense; for bracketing (1), the analysis is subject to the further requirement that any element or dummy-node in the bracketed structural-analysis must match a tree node which is immediately dominated by the top node of the subtree. Brackets (3) and (4) correspond to brackets (1) and (2) respectively, but in this case, the analysis is successful if the subanalysis cannot match the subtree.

For example structural-analysis (1) matches tree (10), but does not match the same tree if the label of the top-node is replaced by the label V.

7.1.3 Assignment of Variables

(a) Variables appearing in the complex-symbol of a complex-element. We defined the conditions of analyzability when the complex-symbols in the structural-description have no prefixed features. If an assignment of the variables can be found such that the tree is analyzable with respect to the assigned structural-description, i.e., the structural-description where all the prefixes have been replaced by their assigned value, then the tree is analyzable with respect to this structural-description.

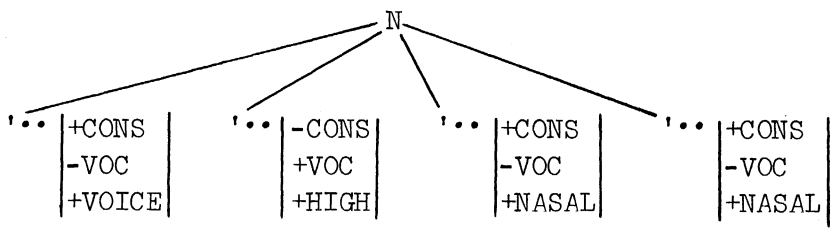
For example, let us consider the structural-descriptions (16), (17), (18) and the tree (19).

(16) % '|-CONS +VOC| '|+CONS -VOC (ALPHA)NASAL| %

(17) % '|-CONS +VOC| '|+CONS -VOC (ALPHA)NASAL|
'|+CONS -VOC (ALPHA)NASAL| %

(18) % '|-CONS +VOC| '|+CONS -VOC (ALPHA)NASAL|
'|+CONS -VOC (-ALPHA)NASAL| %

(19)



Structural-description (16) matches tree (19) for the assignment of + to ALPHA

Structural-description (17) matches the tree (19) for the same assignment

Structural-description (18) does not match tree (19), because no assignment of ALPHA allows a match.

The variable occurring in the complex-symbol of a complex-element is assigned after analysis if the complex-element matches a node in the tree. In the previous example, ALPHA is assigned after comparison of the tree and the structural descriptions (16) and (17); on the other hand, if tree (19) is analyzed with respect to structural description (20),

(20) % '|-CONS +VOC| ('|+CONS -VOC (ALPHA)NASAL|,
'|-CONS (BETA)NASAL|) %

the complex-element '|-CONS (BETA)NASAL| does not match any node and since BETA does not appear in the complex-symbol of any other matching complex-element, BETA remains unassigned after analysis.

There are as many assignments for the set of variables as there are analyses of the tree.

(b) Variables appearing in the complex-symbol of a skip or in the definition of complex names and input-names. The assignment of variables appearing in the definition of a complex-symbol-name, or an input-name, or in a bounded-skip is independent in each occurrence of that complex-symbol and need not be uniform throughout the structural-

description, i.e., in an assigned structural description these variables may have different assignments; the assignment is restricted to the matching of only one node, or to the testing of one complex symbol in case of skips.

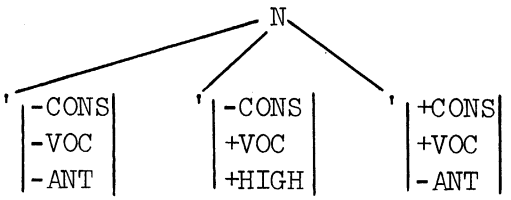
For example, if the complex-symbol-name GR is defined as (21), then structural description (22) matches tree (24), but structural description (23) does not.

(21) GR = |(ALPHA)CONS (ALPHA)VOC -ANT|

(22) % 'GR '|-CONS +VOC| 'GR %

(23) % '|(ALPHA)CONS (ALPHA)VOC -ANT| '|-CONS +VOC|
'|(ALPHA)CONS (ALPHA)VOC -ANT| %

(24)



Structural description (22) is equivalent to structural description (25) where all the complex-symbols are specified in the structural-description.

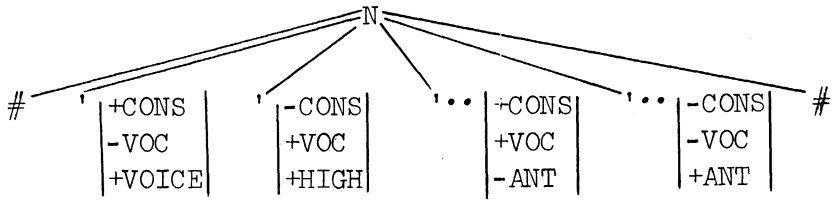
(25) % '|(ALPHA)CONS (ALPHA)VOC -ANT| '|-CONS +VOC|
"|(BETA)CONS (BETA)VOC -ANT| %

In the case of skips, complex-symbols and complex-symbol-names play the same role. For example, structural descriptions (26) and (27) match tree (28).

(26) % '|-CONS +VOC| 'O'GR #

(27) % '|-CONS +VOC| 'O' |(ALPHA) CONS (ALPHA)VOC -ANT| #

(28)



The variables appearing in these contexts are not assigned after an analysis. The difference in conventions between on the one hand complex-symbols appearing in complex-elements and on the other hand complex symbols appearing in skips and definitions correspond to the fact that in the complex-symbols of complex-elements, the variables represent true relations between different nodes in the tree, where as in the other cases, the variables characterize only types of segment and do not represent a relation between segments.

To avoid interference between the two interpretations, the same variable should not be used in the complex-symbol of a complex-element and also in a skip or in the definition of a complex-symbol-name or of an input-name. For instance, if GR is defined as above, the following structural-description might not behave as expected:

(29) % '|+CONS -VOC (ALPHA)NASAL| 'GR
'|-CONS +VOC (ALPHA)NASAL| %

The variable appearing in the definition of a name represents a contraint which typifies a complex symbol, and a definition such as

(30) ZZ = |+CONS -VOC (ALPHA)NASAL|

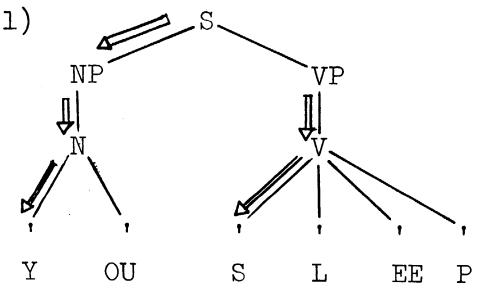
makes little sense, since ALPHA does not indicate any constraint; ZZ matches all tree complex-symbols that includes |+CONS -VOC| and that are neither marked nor unmarked for the feature NASAL.

7.1.4 Scan

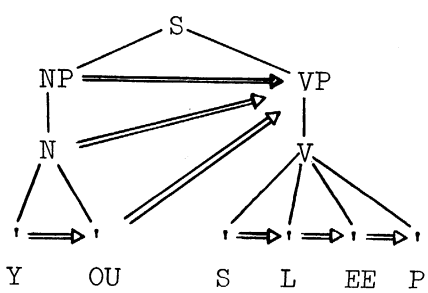
The scan defines an order among the possible analyses of a tree with respect to a structural description. One scan only is defined in the program. It is a left-to-right scan. We define the notion of first analysis and the notion of next analysis, thus a complete ordering of the analysis.

First we must introduce a few definitions relative to the nodes of a tree and the element of a structural description. Node A is next below node B, if A is the first daughter of B. Node A is next right to node B, if either B has a right sister and A is B's right sister or B does not have a right sister and A is the right sister of the lowest node which both dominates B and has a right sister. In the example below, the arrows indicate in (31) the relation next below and in (32) the relation next right.

(31)



(32)



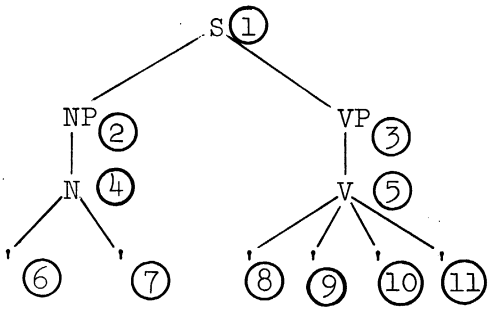
In a structural description elements and dummy-nodes play the same role and are referred to as descriptor-nodes, or simple, as descriptors. Descriptor A is next right to descriptor B in a structural description in the obvious sense, i.e., A is to the right of B and there is no intermediate descriptor between A and B. Node A is below node B in a tree if there is a chain of nodes X_o, X_1, ... X_n where X_o=A and X_n=B, such that each node of the chain, but the last, is next below the next node. A cut is a chain of nodes such that each node, except the first, is next right to the previous or below a node which is next to the previous node of the chain. The first node is called the beginning node, the last, the terminating node of the cut. In the tree (32), for example, the node NP and the last four unlabelled nodes form a cut.

An unlabelled cut is a cut, in which all the nodes are unlabelled nodes and such that each node, except the first, is next right to the previous or the first unlabelled node below the node which is next right to the previous node of the cut.

A cut C joins the nodes A and B if the chain A C B is a cut. A cut joins the node A and the right edge of the tree if A C is a cut and the terminating node of the cut C does not have a next right successor. A cut C joins the left edge of the tree and the node B if C B is a cut and the beginning node of the cut does not have a predecessor, i.e., a node such that the beginning node of the cut is its next right successor. In case of unlabelled cuts, the joining cut must be such that there are no unlabelled nodes between the node next right to A and the

beginning node of the cut or, if the cut joins the left edge of the tree, between the top of the tree and the beginning node of the cut.

(33) Example



In tree (33) the nodes are numbered only for reference during the discussion.

In the tree the cut (2, 8, 9, 10, 11) joins the two edges of the tree; the cut (8, 9, 10, 11) is an unlabelled cut which joins the node (2) to the right edge of the tree and the unlabelled cut (7, 8, 9, 10) joins node (6) to node (11).

There are two types of matching in the algorithm we define in the next section: a matching of nodes with descriptors and a matching of cuts with skips. We shall use the term matching for the first type and the term s-matching for the second (this formal difference makes the description easier, a matched node refers to a node which has been matched during the comparison of a descriptor and a node, not during the comparison of a skip and a cut.)

A descriptor * matches any single node, a descriptor which is a node in the structural description matches a tree node with the same label and a dummy-node ' matches an unlabelled tree node.

A skip % s-matches any cut, including an empty cut (i.e., a cut with no nodes). A bounded-skip s-matches an unlabelled cut of length at least equal to the lower bound and at most equal to the upper bound if the bounded skip has an upper bound; each of the nodes of the cut must satisfy the conditions of the node name specified in the skip, i.e., if the node-name is a complex symbol or a complex symbol name, the complex-symbol attached to each node must include this node name and if the node name is an input name, the complex symbol attached to each node must give a true value to the boolean combination of the input name. A bounded skip whose lower bound is zero may s-match an empty cut.

A. First analysis. The description of the analysis requires the use of two markers, a tree marker pointing to a node in the tree and a structural description marker pointing to a descriptor in the structural analysis. Initially, the tree marker points to the top-node of the tree and the structural description to the left-most descriptor.

After a successful match of the node pointed to by the tree marker and the descriptor pointed to by the structural description marker an attempt to move the tree pointer to the node next right to the matched node and the structural description marker to the descriptor next right to the matched descriptor is undertaken. If the attempt is successful, both pointers are moved. This stage is the "ready for a next match" state, this is the state assumed at the beginning of an analysis. A new match is attempted every time this state is reached. If after a

successful match, the state "ready for a next match" cannot be reached, the next step is the back-up procedure (which we describe later) unless the analysis is successful. The analysis is successful, if the structural description marker cannot move to a next right position and either the last matched descriptor is not followed by a skip in the structural analysis and the tree marker cannot move to a next right position (i.e., it is on the right edge of the tree) or the skip following the last matched descriptor in the structural analysis s-matches any cut joining the last matched node and the right edge of the tree.

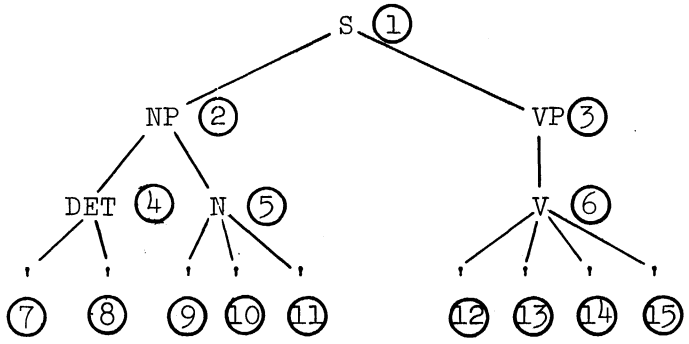
When a state of "ready for a next match" is reached, or initially, a match is attempted between the node pointed to by the tree marker and the descriptor pointed to by the structural description marker.

(34) Example

Structural description

DET	N	*
1	2	3

Tree



Initially, The tree marker points to the node S.
The s.d. marker points to the descriptor DET.

1st match S does not match DET.
The tree marker moves to the node NP, next below S.

NP does not match DET.
The tree marker moves to the node DET, next below NP.
DET matches DET.
The match is successful, the pointers move to their next right position, the tree marker to N and the s.d. marker to N.

2nd match N matches N
The match is successful, the pointers move to their next right position, the tree marker to VP and the s.d. marker to *.

3rd match VP matches *.
The match is successful, none of the pointers has a next right position: the analysis is successful.

If a match is found and a complex-symbol follows the descriptor in the structural description, this complex-symbol is compared for inclusion with the complex symbol attached to the node matched by the descriptor. If a complex-symbol-name follows a dummy-symbol, the test is again inclusion, but this time with the complex symbol corresponding to the complex-symbol-name. If an input name follows a dummy-symbol, each of the complex symbols mentioned in the definition of the input name is tested for inclusion and the corresponding booleancombination is computed. If the comparison fails or the booleancombination is not met, the analysis proceeds as though the tree node and the descriptor do not match, otherwise, the match is valid.

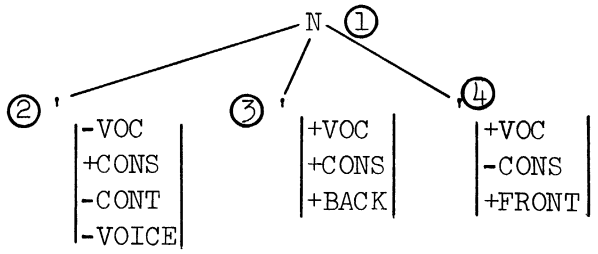
(35) Example

Structural description * |+CONS| 'C 'V
1 2 3

C and V are defined as V = |+VOC -CONS|

C = ¬ V

Tree



Initially The tree marker points to the node N.
The s.d. marker points to the descriptor *.

1st match N matches the descriptor *.
The complex symbol |+CONS| is tested for inclusion in the complex symbol attached to N, which is empty.
The test fails, the tree marker moves to the node (2) next below.
The unlabelled node (2) matches the template *.
The complex symbol |+CONS| is included in the complex symbol attached to the dummy node (2); the match is successful.
The tree marker moves to point to the unlabelled node (3) in the tree and the s.d. marker moves to point to the dummy node (2) in the structural description.

2nd match The unlabelled node (3) matches the descriptor '(2).
The complex symbol |+VOC -CONS| mentioned in the input-name C by its name V is tested for inclusion in the complex symbol |+VOC +CONS +BACK| attached to the unlabelled node (3).
The booleancombination ¬ V, (i.e., not V) is true, the match is valid.
The pointers move to their next right positions.
The tree markers points to the unlabelled node (4) and the s.d. marker to the dummy-node (3).

3rd match, etc.

If a complex symbol contains an unassigned variable, a successful inclusion test assigns a value to this variable. The variables are all unassigned at the beginning of an analysis. At the end of a successful analysis the variables which appear in the complex symbols of the structural description and which follow matched descriptors are all assigned.

This does not apply to the complex symbols of complex-symbol-names or input-names, which are all unassigned after any match.

If an integer precedes the descriptor, or a complex symbol follows the descriptor, the restrictions must be checked (we see later that restrictions mention the integers preceding a template, or the variables appearing in a complex symbol). During the process of an analysis, the value of a restriction may depend on information not yet available. For instance, in the following structural description, when the first dummy node is matched, the value BETA has not yet been defined.

(36) SD '|(ALPHA)LOW| '|(BETA)LOW|, WHERE BETA VEQ ALPHA.

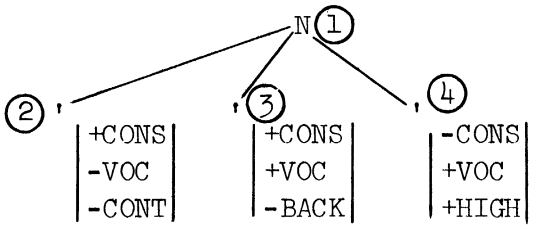
For this reason a three-valued logic is used. The value of a condition is "undefined" until the analysis has proceeded far enough to determine a value of "true" or "false" for the whole restriction. Again, failure of the restriction causes the analysis to proceed as though the node on which the restriction received the value "false" had not matched the descriptor. Everytime a descriptor is preceded by a skip in the structural analysis and the descriptor successfully matches a tree node, the preceding skip must s-match the cuts which join the previously matched node, or the left edge of the tree if there was no previous match, to the newly matched node. If the s-matching fails, and in that case the skip is a bounded skip since % s-matches any cut, the next step depends on whether or not the skip is s-matchable. The skip is s-matchable if there is an unlabelled cut which joins the previously

matched node, or the left edge of the tree if there was no previous match to the newly matched node and which meets all the conditions for s-matching the skip, except that the number of its nodes is smaller than the lower bound of the skip. The analysis, then, proceeds as though the template which is preceded by this skip had not matched the node. If the skip is not s-matchable, the back-up procedure is entered.

(37) Example

Structural description '1'C 1'C 'V %
1 2 3 4

Tree



Initially The tree marker points to node (1).
The s.d. marker points to the descriptor (2).

1st match N does not match the dummy-node (2).
The tree marker moves next below to node (2).
Node (2) matches descriptor (2).
The complex symbol of node (2) satisfies the condition of the input-name C.
Descriptor (2) is preceded by the skip (1), there is no unlabelled cut joining the left edge of the tree and node (2) which contains at least one node; in this case the unlabelled cut is too short.
The analysis proceeds as if node (2) and descriptor (2) do not match.

If in an attempt to match a tree node and a descriptor, the tree marker has moved from one tree node to the next below, until a terminal node has been reached without success, there are two options

depending upon whether or not a skip precedes the descriptor in the structural description. If there is a skip, the tree marker moves to the node next right to the terminal node which has been reached, and a match is attempted again. If in this process, the marker reaches the right most terminal node of the tree without any match, the back-up procedure is entered.

(38) Example continuation of example (37).

1st match (cont'd)
At the end of the description of example (37), the tree marker points to node (2) and the s.d. marker points to descriptor (2) and the analysis proceeds as if they do not match. The tree marker has reached a terminal node and there is no matching. The descriptor (2) is preceded by the skip (1) in the structural analysis: the tree marker moves to node (3) which is next right to node (2).
The unlabelled node (3) matches the descriptor (2). The complex symbol of node (3) satisfies the conditions of the input-name C.
Descriptor (2) is preceded by the skip (1) which s-matches the unlabelled cut consisting of the single node (2). The match is successful. The s.d. marker move to the next right descriptor (3) and the tree marker moves to the next right node (4).

2nd match, etc.

If the structural description marker is pointing to the first descriptor of an option, i.e., a choice containing only one structural-analysis, the analysis proceeds as if the parentheses around the option were absent; the parentheses of an option only affect the backup procedure.

If the structural description marker is pointing to the first descriptor of a collection (i.e., a choice with several structural-analyses) the analysis proceeds as if the parentheses were absent. However, the instruction "move to the next right template" to the structural description marker has a different meaning if the descriptor currently pointed to by the structural description marker is the last descriptor before one of the commas of a collection. In that case, the structural description marker moves to the next descriptor following the last parenthesis of the collection. This hopping of the structural description marker before the comma of a collection applies at any level of embedding of choices, for instance in the structural analysis A (B, (C,D), E) F, if the structural description marker must move from descriptor C to the next descriptor it moves to descriptor F. The backup procedure allows the structural description marker to move inside the different structural-analyses of a collection.

If a structural-analysis within angle brackets follows a descriptor that has been satisfactorily matched, a record is made of relevant information about the current status of the analysis, and analysis begins again, this time using the angle-bracketed structural-analysis and the subtree headed by the node matched to the template. If no / precedes, the tree marker is only allowed to point to immediate daughters of the top node during this analysis, instead of looking all the way down to terminal nodes. If a ¬ precedes and the subtree is not analyzable, or if no ¬ precedes and the subtree is analyzable,

analysis continues following the angle-bracketed structural-analysis; otherwise, analysis proceeds as if the descriptor followed by this angle-bracketed structural-analysis had not matched its tree node.

Backup procedure

(a) If a backup procedure is entered when the structural description marker points to the left-most descriptor, the analysis is unsuccessful.

(b) If the backup procedure is entered when the structural description marker points to the first template of an option, the structural description marker hops to the descriptor next right to the last descriptor of the option; the tree marker does not move.

(c) If the backup procedure is entered when the structural description marker points to the first descriptor of the structural-analysis of a collection, the tree marker does not move, the structural description marker hops to the first descriptor of the next structural-analysis of the collection if there is one; if there is none, it follows the general case (d) of backup.

(d) In the general case, the structural description marker backs up to the previous descriptor which matched and the tree node to the previous tree node which matched. The matching resumes from these new pointer positions as though the node

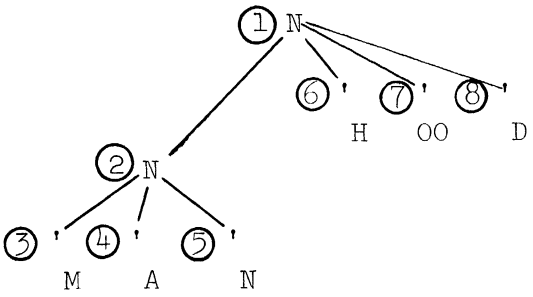
and the descriptor do not match. If during a previous match of a node on which the tree marker backs up a variable was assigned or a condition was defined, they are respectively unassigned and undefined.

(39) Example

Structural description

N	'C	%
1	2	3

Tree



Initially The tree marker points to the top-node N (1).
The s.d. marker points to the descriptor N (1).

1st match N matches the descriptor N.
The tree pointer cannot move on the right, and the s.d. marker can move to the right; the back-up procedure is entered.
The markers move back to the last successful match.
The tree marker points to the top-most node N.
The node N does not match the descriptor N (because of the back-up procedure convention).
The tree marker moves to the next node (2) N below.
The new node (2) N matches the descriptor N.
The tree marker moves to the next right node, the unlabelled node (6).
The s.d. marker moves to the next right descriptor '(2).

2nd match The unlabelled node (6) matches the description (2)' and the booleancombination corresponding to the input name C is true.
The descriptor (2) does not have a next right successor and is followed by a skip (3) which matches the last two unlabelled nodes: the analysis is successful.

B. Next analysis. After an analysis is found, the next analysis is obtained by entering the back-up procedure on the last descriptor that matched in that analysis. The values assigned to variables are saved before a next analysis is started.

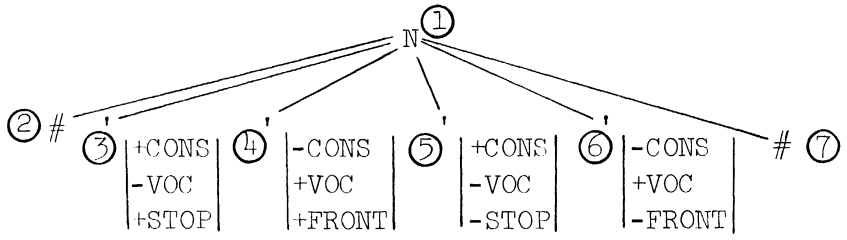
Examples of ordering

(46) Example

Structural description

%	1'V	('O'C	'V)	'O'C	#
①	②	③	④	⑤	⑥

Tree



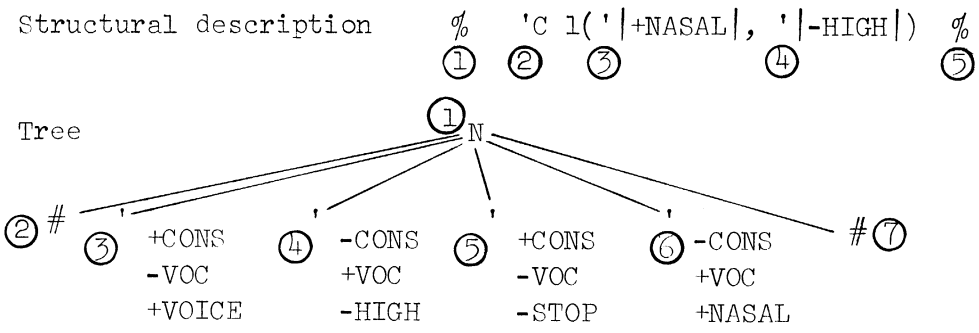
1st analysis Skip 1 s-matches the cut (2 - 3).
Descriptor 2 matches the node (3).
Descriptor 4 matches the node (5).
Skip 5 s-matches the cut (6).
Descriptor 6 matches the node (7).

We can abbreviate this analysis as 1(2 - 3), 2(3), 3(4), 4(5), 5(6), 6(7).

2nd analysis 1(2 - 3 - 4 - 5), 2(6), 3(ø), 4(ø), 5(ø), 6(7).

This can be paraphrased as: if a structural description contains an option, the analysis where the option is present precedes the analysis where the option is absent.

(41) Example



1st analysis 1(2), 2(3), 3(∅), 4(4), 5(5 - 6 - 7)

2nd analysis 1(2 - 3 - 4), 2(5), 3(6), 4(∅), 5(7)

This example shows one of the characteristics of left-to-right scan, namely, if there exist several analyses of a tree with respect to a structural description, they are ordered from left-to-right. An analysis is to the right of another analysis if its first matched node is below, or right of, the first matched node of this other analysis.

7.2 RESTRICTIONS

Restrictions add conditions to a structural description, and are necessary to state requirements for analyzability which cannot otherwise be expressed in the notation for structural descriptions. For example, the requirement that the value of one feature be smaller than the value of another feature necessitates a formulation with a restriction. The conditions of analyzability when a structural description when a structural description contains a restriction remain the same as previously stated, except that the analysis of the tree must satisfy the restriction.

A restriction is a Booleancombination of conditions, the logical connectives being ¬ (not), & (and), | (or). The metarules defining the restrictions are:

3.01 restriction ::= booleancombination[condition]

3.02 condition ::= unary-condition ▯ binary-condition

3.03 unary-condition ::= unary-relation integer

3.04 binary-condition ::=

integer binary-tree-relation node-designer

▯ integer binary-complex-relation complex-symbol-designator

▯ p-value value-relation p-value

3.05 node-designator ::= integer ▯ node ▯ dummy-node

3.06 complex-symbol-designator ::= complex-symbol ▯ integer

3.07 p-value ::= sign ▯ integer ▯ variable

If a node-designator is an integer, it refers to the tree node indexed by that integer in the analysis. For example, in the structural-description

(42) % 1'C 2'C %, WHERE 1 EQ 2

The binary-condition assures that in the tree matches by this structural description, the subtrees headed by the nodes indexed 1 and 2 are identical, namely here that they have the same complex symbol. If a complex-symbol-designator is an integer, it refers to the complex-symbol attached to the tree node indexed by that integer.

(43) 1* < % 1'V ('O'C 'V) 'O'C # > , WHERE 1 INCl|+N|
|1 INCl|+N|

Structural description (43) assures that the node dominating the unlabelled nodes contains either the feature specification |+N| or the feature specification |+V|, that is it is either a noun or a verb. If p-value is a variable, the variable must be defined somewhere in the analysis, i.e., it must appear in the complex-symbol of a complex-element which matches a node.

The various relations are defined in the following metarules.

3.08 unary-relation ::= TRM ▯ NTRM ▯ NUL ▯ NNUL

3.09 binary-tree-relation ::= EQ ▯ NEQ ▯ DOM ▯ NDOM
▯ DOMBY ▯ NDOMBY

3.10 binary-complex-relation ::= INCl ▯ NINCL
▯ CSEQ ▯ NCSEQ ▯ NDST ▯ NNDST

3.11 value-relation ::= VLT ▯ VLE ▯ VEQ ▯ VGE
▯ VGT ▯ VNE

The relations 3.08 - 3.10 are in pairs of the form XXX and NXXX, where NXXX is the negation of XXX; for the value-relations, this convention has not been followed because the negation of a relation has another conventional name.

Unary-relations

TRM : the corresponding node is a terminal node of the tree

NUL : the descriptor governed by the integer does not match a node in the tree; this relation is meaningful only if the integer governs a descriptor in an option or a collection.

Binary-tree-relation[1]

EQ : The subtrees dominated by the two nodes are identical, including equality of complex symbols. For this relation the node-designator must be an integer.

DOM : The node on the left-hand side of the relation dominates the node on the right-hand side without an intervening S. For this relation, the node-designator must be a node.

DOMBY : The node on the left-hand side of the relation is dominated by the node on the right-hand side. For this relation the node-designator must be a node.

Binary-complex-relation[1]

These relations are defined in Chapter 6; they are INCl (include), CSEQ (equal) and NDST (nondistinct). If a complex-symbol contains a variable, this variable is assigned a value during the comparison if it was not already defined. In the case of a negative complex relation, the value assigned to a variable is not necessarily unique.

---

[1] Other relations defined in syntax (cf. Friedman et al.; §8) are available, but do not seem to be required in phonology.

For instance, if the condition

(44) 2 NCSEQ |(ALPHA)CONS (-ALPHA)VOC|

is tested for a node whose complex symbol is |+CONS +VOC|, the condition will be met, furthermore, if ALPHA was unassigned, ALPHA is assigned the value + when the feature CONS is checked first and the value - when the feature VOC is checked first. Binary-complex-relations as a rule should be used to check conditions, and not to assign values. Another example is the restriction (45):

(45) 3 NCSEQ |(ALPHA)NASAL|

If ALPHA is not defined at the time of comparison, ALPHA is assigned the value of the feature NASAL in the tree complex-symbol, and the comparison returns false whatever this value of NASAL, unless the tree complex symbol was marked or unmarked for NASAL, in which case ALPHA cannot be assigned and since the positive condition 3 CSEQ |(ALPHA) NASAL| in this case is false, the negative restriction 3 NCSEQ |(ALPHA)NASAL| is true.

Value-relation

VLT : less than

VLE : less than or equal to

VEQ : equal to

VGE : greater than or equal to

VGT : greater than

VNE : non-equal to

The p-values on both sides of the relation must be both integers or both signs; otherwise, the relation is assumed to be false. In the Sound Pattern of English (SPE), the interpretation of value-relation is that the sign - is greater than any integer. Therefore, the relation alpha $\geq$ 3 of SPE should be written in our conventions (ALPHA VGT 3 | ALPHA VEQ -).

The variables appearing in a value-relation must be assigned a value at some point in the derivation; otherwise the relation is taken to be true. This excludes the use of variables which appear in skips, in complex-symbol-names or input-names in restrictions, since they are unassigned after each match.

Restrictions are restricted to nodes matched by descriptors because a relation requires the use of integers, which can only govern descriptors, or of a variables which appears only in the complex symbols of descriptors. In SPE, some rules use restrictions on skips, for instance

(46) ## % 1'|+VOC -CONS 1STRESS| X ##,

WHERE X is an unlabelled skip which does not contain a segment including the feature specification |1STRESS|.

This restriction should be interpreted in the program as a bounded-skip using an input name describing the restriction.

(47) NSTRESS = ¬ |1STRESS|

SD ## % 1'|+VOC -CONS 1STRESS| '0'NSTRESS ##.

7.3 STRUCTURAL CHANGE

The effect of the structural change of a rule is to modify nodes which have been indexed during the analysis and trees headed by such nodes. For example a rule with the structural-description % 1'|+CONS -STOP -VOC| # and the structural-change ERASE 1, has the effect of deleting the subtree headed by the node which matches the term 1'|+CONS -VOC -STOP| of the structural description.

The metarule for a rule is

8.03 rule ::= SD structural-description.

opt[SC structural-change .]

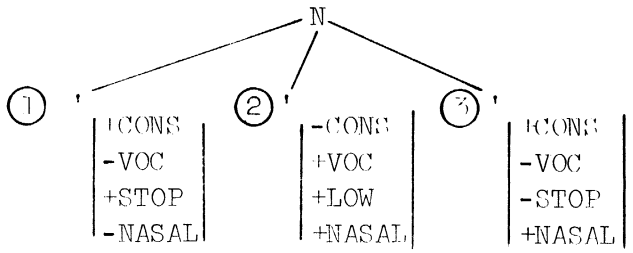
A more extended example of rule is the rule (48) of reduplicative stem formation, which reduplicates some monosyllabic stems.

(48) SD 1'C 2'V 3('C).

SC 1 ADLES 1, 2 ADLES 1, 3 ADLES 1.

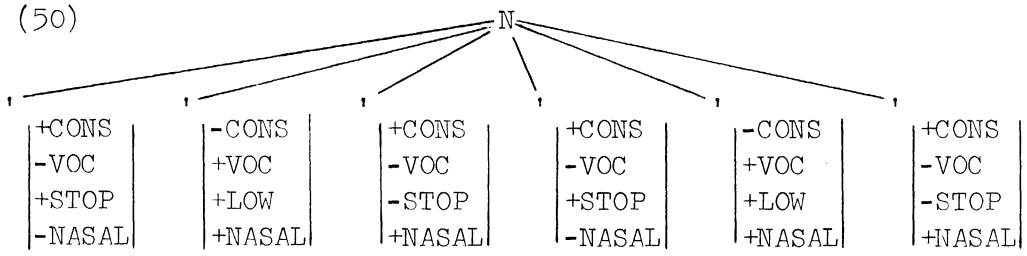
This rule applies to the tree (49)

(49)



The indices 1, 2, and 3 of the structural-description are represented in the tree. The three changes are instruction to adjoin the three nodes indexed 1, 2, and 3 as left sisters of the node 1 in this order. The tree, after application of the rule, contains six unlabelled nodes:

(50)



The order of the changes is important. Any other ordering of the changes in the previous structural-change does not produce an output tree where the two syllables are identical.

The metarule for structural changes is

5.01 structural-change ::= clist[change-instruction]

5.02 change-instruction ::= change [] conditional-change

In the reduplicative rule, all the change-instructions are simple changes rather than conditional-changes. We discuss these two types of change-instructions in this order.

7.3.1 Simple changes

A change consists of an operator and one, two, or three arguments.

5.04 change ::= unary-operator integer

▯ tree-designator binary-tree-operator integer

▯ complex-symbol-designator
binary-complex-operator integer

▯ complex-symbol-designator
ternary-complex-operator integer integer

The change operators are either operators which change tree structures or operators which modify complex symbols.

Tree operators[1]

5.06 unary-operator ::= ERASE

5.07 binary-tree-operator ::=

ADLAD ▯ ADFID ▯ ADRIS ▯ ADLES ▯ SUBST ▯ ADCHR ▯ ADCHL

ALADE ▯ AFIDE ▯ ARISE ▯ ALESE ▯ SUBSE ▯ ACHRE ▯ ACHLE

The unary-operator ERASE deletes the subtree indexed by the number, and then chains upward, i.e., it erases all the ancestors of the subtree until a node with at least two daughters is encountered.

The binary-operators are different types of adjunctions: sister adjunction, daughter adjunction and Chomsky adjunction. They are divided into two groups: adjunction with erasure and adjunction without erasure.

---

[1] Other operators are available, which are described in the syntax program (§8), but which do not seem to be required for phonology.

ADLAD	adjoin as last daughter	ALADE, idem with erasure
ADFID	adjoin as first daughter	AFIDE, idem with erasure
ADRIS	adjoin as right sister	ARISE, idem with erasure
ADLES	adjoin as left sister	ALESE, idem with erasure
ADCHR	Chomsky-adjoin to the right	ACHRE, idem with erasure
ADCHL	Chomsky-adjoin to the left	ACHLE, idem with erasure

An operator with erasure is equivalent to the corresponding operator without erasure, followed by an erasing of the first argument of the operator; hence i ALADE j has the same effect as i ADLAD j, ERASE i.

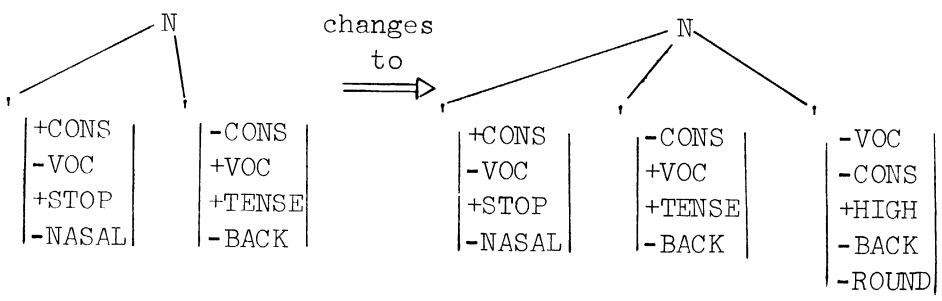
The second argument of a binary-tree-operator must be an integer, i.e., it must refer to a node or sub-tree which has been specified in the structural description, the first argument of the tree-binary-operator may also refer to a node or subtree defined in the structural-description as it is the case in the rule of reduplicative stem formation, but it can also refer to nodes or trees directly.

5.05 tree-designator ::= (tree) ▯ node ▯ integer

An example of transformation which adds a subtree is the rule of diphthongization.

(51) SD % 1'|-CONS +VOC +TENSE (ALPHA)BACK| %.

SC ('|-VOC -CONS +HIGH (ALPHA)BACK (ALPHA)ROUND|) ARISE 1.



Complex-symbol operators

5.08 binary-complex-operator ::= ERASEF [] MERGEF [] SAVEF

5.09 ternary-complex-operator ::= MOVEF

These changes are defined in Chapter 6. An example using a binary-complex-operator is the rule of penultimate stress placement.

(52) SD % 1'V ('O'C 'V) 'O'C #.
SC |1STRESS| MERGEF 1.

The feature specification 1STRESS is added to the complex symbol of the penultimate vowel if the word has two syllables, or to the only vowel of the word, if it is monosyllabic. An example of a rule using a ternary-complex-operator is the rule of voice assimilation

(53) SD % 1'C 2'C %.
SC |*VOICE| MOVEF 2 1.

The feature specification of the second consonant for the feature VOICE is added, or substituted, in the complex symbol of the first consonant. Note that (53) is equivalent to formulation (54):

(54) SD % 1 C '|+CONS -VOC (ALPHA)VOICE| %.
SC |(ALPHA)VOICE| MERGEF 1.

This second formulation, however, is less efficient than the first. If a complex symbol on the left-hand side of an operator contains a

variable, this variable must have been assigned in the analysis or in evaluating the restriction of a conditional change.

7.3.2 Conditional changes

A conditional change is a structural change which applies only if certain conditions are met, more specifically:

5.03 conditional-change ::= IF < restriction >
THEN < structural-change > opt[ELSE < structural-change >]

The structural-change following THEN is executed if the restriction is met; otherwise, the structural-change following ELSE is executed, if there is one. The conditional change makes it possible to specify that a change takes place only after another has taken place. For instance the rule of velar softening:

(55) SD % 1'|-ANT -CONT +DER| '|-CONS -BACK -LOW| %.
SC |+COR +STRID| MERGEF 1,
IF < 1 INC1 |-VIOCE| > THEN < |+ANT| MERGEF 1 >.

This rule states that segments which are marked |-ANT -CONT +DER +COR +STRID -VOICE| become |+ANT| in front of segments marked |-CONS -BACK -LOW| only if they have undergone the rule:

(56) SD % 1'|-ANT -CONT +DER| '|-CONS -BACK -LOW| %
SC |+COR +STRID| MERGEF 1.

7.4 SIMPLE RULES

Simple rules are abbreviations for a series of recurrent rule types. These rules allow modification or introduction of only one node. The only changes they allow are ERASE, MERGEF and sister or daughter adjunctions.

8.04 simple-rule ::= operand => operator

opt[/ < structural-description >].

8.05 operand ::= complex-symbol-reference [] input-name [] *

8.06 operator ::= complex-symbol-reference [] *

The structural description of a simple rule must contain at least one underline, and be such that exactly one underline is nonnull in any analysis. The optional expression in a simple rule is called the context of the rule. If a rule does not have a context, it is assumed to be /< % _ % >.

ERASE. This change is characterized by the operator *. During analysis, a dummy node followed by the operand replaces the underline of the structural-description. The change erases the node which matches this newly introduced dummy-node. The following simple rules (57) and (58) are equivalent.

(57) C => * /< % _ # >.

(58) SD % 1'C #.

SC ERASE 1.

MERGEF. This change takes place when the operand is a complex-symbol-reference or an input-name and the operator is a complex-symbol-reference. During analysis, a dummy node followed by the operand replaces the underline. The change consists of merging the operator into the complex symbol attached to the node which matches this newly introduced dummy node. The following simple-rules (59) and (60) are equivalent.

(59) C ⇒ |(ALPHA)VOICE| / < % _ '|+CONS -VOC (ALPHA)VOICE|%>.

(60) SD % 1'C '|+CONS -VOC (ALPHA)VOICE| %.

SC |(ALPHA)VOICE|MERGEF 1.

ADJUNCTION. These operations are characterized by the operand *. The node which is adjoined is an unlabelled node to which is attached the complex symbol of the operator. The type of adjunction depends on the position of the underline in the structural description. The first applicable one of the following rules applies:

(a) There exists an element or dummy-node N on the left of, and contiguous to, the underline; the new node is adjoined as a left sister of N.

(b) There exists an element or dummy node N on the right of, and contiguous to, the underline; the new node is adjoined as a right sister of N.

(c) The underline is the last element of an angle-bracketed structural-analysis; the new node is adjoined as the last

daughter of the element preceding the bracketed structural-analysis.

(d) The underline is the first element of an angle-bracketed structural-analysis; the new node is adjoined as the first daughter of the element preceding the bracketed structural-analysis.

(e) The simple rule is not well defined.

The two following rules are equivalent.

(61) * ⇒ |-VOC -CONS +HIGH (ALPHA)BACK (ALPHA)ROUND|
/ < % ' |-CONS +VOC +TENSE (ALPHA)BACK|_% >

(62) SD % 1'|-CONS +VOC +TENSE (ALPHA)BACK| %
SC |-VOC -CONS +HIGH (ALPHA)BACK (ALPHA)ROUND|
MERGEF 1.

CHAPTER 8

TYPES OF RULES

The program allows phonological rules to be input in either of two formats: the transformational format, or the simple-rule format, the second one being only a shorthand for the first. Each rule may be further specified according to its type, the group to which it belongs, the optionality and its mode of application. These specifications belong to the identification of the transformation.

8.02 transformation ::= TRANS identification . rule

[] RULE identification . simple-rule

8.07 identification ::=

opt[integer] transformation-name

opt[list[parameters]]

8.08 transformation-name ::= word

The identification of a rule must contain the name of the rule; for instance the rule for stress placement could be named NSTRESS. The integer preceding the name is optional and may be used for reference; it is ignored by the program. The parameters define the group, the optionality and the mode of application. If the group is not specified in the identification, it is assumed to be the group of the previous transformation, or the group specified in the implicit statement if there is no previous transformation, or group I if there is no implicit

statement. If the optionality or the mode of application is not specified in the identification, it is assumed to be identical to the optionality and mode of application specified in the implicit statement; if there is no implicit statement they take the value OB for optionality and the value AC for repetition.

8.01 transformations ::= TRANSFORMATIONS
opt[IMPLICIT parameters]
list[transformation] CP control-program $END

8.09 parameter ::= group-number ▯ optionality ▯ repetition

8.10 group-number ::= I ▯ II ▯ III ▯ IV ▯ V ▯ VI ▯ VII

8.11 optionality ::= OB ▯ OP

8.12 repetition ::= AC ▯ ACAC ▯ AACC

The group-numbers and the control-program define the order of application of the rules and are discussed in the next chapter; in this chapter, we describe the repetitions or modes of application, and the optionality.

8.1 MODES OF APPLICATION

The modes of application are defined here for obligatory rules. In the next section, we describe how they are modified in case of optional application.

Several modes of application have been described[1] for phonological

[1] Cf. SPE. Johnson (1970, J31-J118); Anderson (1968); Harms (1966, 608); McCawley (1968, 20-22).

rules. In the program, we have implemented three modes which seem adequate for the description of all phonological rules. This does not preclude that other modes of application might be more justified to capture linguistically significant generalizations[1]; present evidence, however, does not show conclusively that this is the case.

8.12 repetition ::= AC ▯ ACAC ▯ AACC

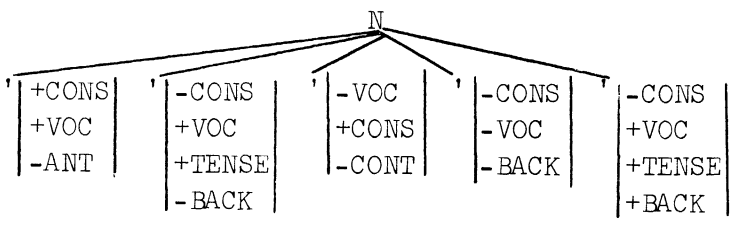
8.1.1 Single Application AC

If the repetition is AC, the analysis algorithm attempts to find the first analysis of the tree with respect to the structural description; if it finds one, the structural change applies to the nodes indexed in the analysis.

The rule of antepenultimate stress placement (1) applied to the word /r̄ed̄jō/, where /r,d,j/ are C's and /e,o/ are V's, and attaches a stress to the first vowel, because the option ('0'C 'V) is present in the first analysis which matches the tree (2).

(1) RULE ANTSTRESS AC.
V => |1STRESS| /<% _ ('0'C 'V) '0'C #>.

(2)



[1] Cf. Friedman et al. (1971, 102-105), define another mode AAC for syntax.

The rule of diphthongization (3) applying to the same word adds only one glide after the first vowel, giving /rējdjō/

(3) RULE DIPHT AC.

* => |-VOC -CONS +HIGH (ALPHA)BACK|

/ <% |-CONS +VOC +TENSE (ALPHA)BACK| _ %>.

8.1.2 Iterative Application ACAC

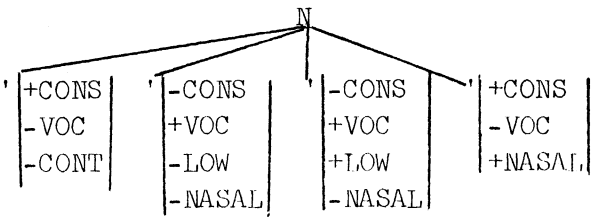
The ACAC mode specifies iterative application. The rule is re-iterated after each application, until no analysis of the tree with respect to the structural description is possible.

If the following rule of nasalization is iteratively applied to the word /poan/ where /o,a/ are V's and /n/ is +NASAL, the vowel /a/ is nasalized on the first application, and the vowel /o/ on the second application, since it is now followed by a segment specified +NASAL.

(4) RULE NASALIZ ACAC.

V => |+NASAL| / <% _ '|+NASAL| %>.

(5)



However, after the second application, if the process is applied again, the vowel /õ/ is still followed by a segment specified |+NASAL|, the tree is analyzable with respect to the structural description, and the rule applies an infinite number of times. Rule (5) should be modified as (7) to prevent this infinite iteration.

(7) RULE NASALIZ ACAC.

|+VOC -CONS -NASAL| => |+NASAL| /<% _ '|+NASAL| %>.

8.1.3 Simultaneous Application AACC

For the repetition AACC, analysis algorithm finds all possible analyses of the tree with respect to the structural description; the structural changes apply to the nodes indexed in the analysis.

If rule (8) of antepenultimate stress placement applies to the word /rēdyō/ simultaneously, both vowels /ē/ and /ō/ receive stress because there are two possible analyses of the tree, one corresponding to the presence of the option ('O'C 'V) and the other to its absence.

If rule (9) of diphthongization is changed to apply simultaneously, in the word /rēdyō/, both vowels are diphthongized, giving /rēydyōw/.

(8) RULE ANTSTRESS AACC.

V = |1STRESS| /<% _ ('O'C 'V) 'O'C # >.

(9) RULE DIPHT AACC.

* => |-VOC -CONS +HIGH (ALPHA)BACK|

/<% '|-CONS +VOC +TENSE (ALPHA)BACK| _ %>.

8.2 OPTIONALITY

In contrast to obligatory application, where the rule applies whenever one or more analyses have been found with respect to the structural description, application of a rule is optional if a choice is made to decide whether or not a rule applies even when some analyses

of the tree have been found. There appears to be many possible definitions for optionality depending on the mode of application. We have implemented one type of optionality for each of the modes of application. The optionality in the identification of a rule indicates whether a rule is optional (OP) or obligatory (OB).

8.12 optionality ::= OB ‖ OP

8.2.1 Single Application

In the case AC there is only one choice: whether or not the rule applies when the tree has an analysis with respect to the structural description.

For instance, if the word /rēdyō/ undergoes a single, optional application of rule (1) of antepenultimate stress placement, the output of the rule can be either /rēdyō/ if the rule applies or /rēdyō/ if it does not apply.

8.2.2 Iterative Application

In the case ACAC, the number of iterations of the rule is the object of the choice; therefore an optional iterative rule may apply zero times, one time, two times, etc.; the maximum number of iterations is the number of iterations in the corresponding obligatory transformation.

For instance, if the word /poan/ undergoes an iterative optional application of the nasalization rule (7), the iteration may stop at

its initial stage, leaving the word unchanged /poan/, or after the first application giving the phonetic form /poãn/, or after the second application (which in this case is the last possible application) giving the phonetic form /põãn/.

8.2.3 Simultaneous Application

In the case AACC, the choice between applying or not applying the changes to any one of the analyses of the tree with respect to the structural description is carried individually for each of the analyses. For example, suppose that the first line of the diphthongization rule (3) is changed to

RULE DIPHT AACC OP.

so that it calls for optional simultaneous application. If applied to tree (2), two analyses are found: the change corresponding to the first one would add a glide after the vowel /ē/, and the second after the vowel /ō/. The two choices between adding and not adding the glide are made independently, so that there are four possible outcomes: /rēdjō/ (no change), /rējdjō/ and /rēdjōw/ (one change), and /rējdjōw/ (two changes).

8.3 FORMAL DEFINITION OF THE MODES OF APPLICATION

8.3.1 Invocation and Application of Rules

Given a subtree and a rule, we say that the subtree is tested

for this rule, or alternatively, that the rule is invoked for the subtree if the procedure which determines the analyzability of the subtree with respect to the structural description is entered. After one or several analyses of the subtree have been found, the rule is said to have applied to the subtree if the procedure which applies the structural changes of the rule is entered at least once.

8.3.2 Formal Definition

In the formal definitions of the modes AC, ACAC, and AACC the step 0 applies only when the rule is optional and must be ignored when the rule is obligatory:

AC
0. decide whether to continue or to stop.
1. invoke the rule for the first analysis.
2. apply if the analysis is successful.

ACAC
0. decide whether to continue or to stop.
1. invoke the rule for the first analysis.
2. apply if the analysis is successful.
3. reiterate steps 0,1, and 2 until the analysis is unsuccessful.

AACC
0. decide whether to do step 3 or go directly to step 4.
1. invoke the rule for all possible analyses.
2. take the first analysis, if any, as the current analysis.
3. apply to current analysis, if any.
4. take the next analysis, if any, as the current analysis.
5. reiterate steps 0,3, and 4 until all analyses have been taken as current analysis.

CHAPTER 9

RULE ORDERING

A phonological grammar consists primarily of a set of rules. It must also indicate the subtrees to which these rules apply and in which order they apply.

There is no general consensus on how the rules should be ordered. For this reason we have introduced a notation for describing different types of ordering. The ordering, thus, becomes part of the phonological grammar. We do not maintain that a universal ordering may not someday be discovered. Our position is only that the linguist should be able to experiment with different types of ordering and to define his own ordering in the absence of a recognized universal scheme.

The ordering is formally described in the control-program in which the user may:

a. group rules into ordered sets and apply rules either individually or by rule sets.

b. specify the subtrees to which a rule or a rule set is to apply.

c. specify the order in which the rules and rule sets are considered.

d. allow the order of application to depend on which rules have previously applied.

The control language uses the mechanism already defined for the appli-

cation of rules. For instance, the fact that a transformation indexes nodes in the tree and therefore defines subtrees allows the use of transformations in the control program to define the subtrees to which rules and rule sets are to apply.

9.1 SUBTREES

Rules do not necessarily apply to the whole tree under derivation. The control-program can specify the subtrees to which the rules apply by listing in a directing-node-definition the nodes which may head these permissible subtrees.

9.02 directing-node-definition ::=

@ (clist[complex-element])

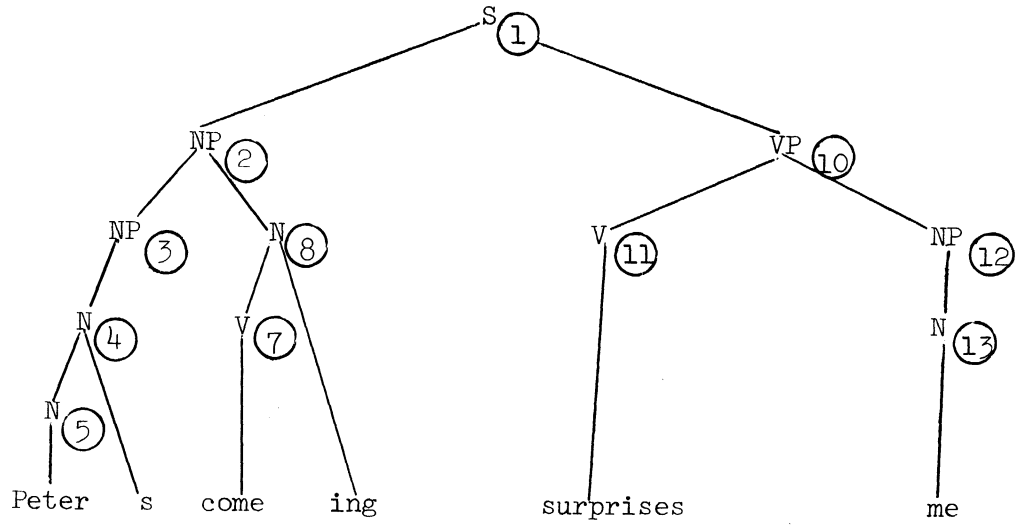
For instance, control program (1)[1] specifies that the transformation with name T1 applies to all subtrees which are headed by nodes labelled either N or V.

(1) CP @(N, V) T1.

The basic order of application of a rule is from bottom-to-top: Any subtree headed by a node which matches one of the complex-elements of the directing-node-definition is a permissible subtree. If there

[1] Control-programs are represented here preceded by the prefix CP and followed by a stop as they appear in the program. It must be understood that the control-program does not formally contain a prefix, nor a final stop (which are part of transformations).

is more than one permissible subtree in the tree under derivation, the subtrees are tested, beginning with the least embedded subtrees. A permissible subtree, therefore, is tested for a rule only if it does not contain another permissible subtree, or if all the permissible subtrees it contains have already been tested. For example, control-program (1), in a grammar which contains only the transformation T1, specifies that the subtrees of tree (2) headed by the nodes (13), (11), (7), (8), (5), and (4) are tested in this order for transformation T1.



9.2 INSTRUCTIONS

A control-program is a series of instructions to the program which specifies the transformations for which the subtrees defined in the directing-node-definitions are tested.

9.01 control-program ::=

sclist[opt[directing-node-definition] instruction]

9.02 instruction ::= transformation-element

▯ control-element ▯ sclist[instruction]

In the next sections we describe the control-elements which define other types of ordering and trace the evolution of a tree in a derivation.

9.05 control-element ::= IN-instruction ▯ IF-instruction

▯ RPT-instruction ▯ STOP-instruction

▯ TRACE-instruction

In this section we describe the basic ordering, i.e., the ordering without control-elements. The rules may be individually ordered, in which case the control program specifies the transformation-name of the rule, or in groups, in which case the control program specifies the group-number of the group.

9.04 transformation-element ::= transformation-name

▯ group-number

A rule may belong to only one group.

In the first case we consider all the transformation-elements are transformation-names and are preceded by a directing-node-definition. In this case each rule applies bottom-to-top to the subtrees defined by the directing nodes. The rules are linearly ordered as they appear in the control program. For example, consider control program (3) and tree (2).

(3) CP@(N,V)T1; @(N)T2; @(N,V,NP,VP)T3; @(N,V)T4.

T1 applies bottom-to-top to the tree (2), i.e., the subtrees (13), , (7), (8), (5), and (4) are tested in this order for transforma- T1. After this, rule T2 applies bottom-to-top to tree (2), i.e., subtrees (13), (8), (5), and (4) are tested in this order for sformation T2, and then for rule T3 and finally for rule T4.

If no directing-node-definition precedes an instruction, the di- ing nodes for this instruction are taken to be the directing nodes he previous instruction, or the directing-node-statement @(S) if e are no previous instructions. For example, control-programs (4) (5) are equivalent.

(4) CP@(N,V)T1; T2; T3; T4.

(5) CP@(N,V)T1; @(N,V)T2; @(N,V)T3; @(N,V)T4.

convention applies also to instructions which are control-ele- s. That is, if a control-element is not preceded by a directing- -definition it is assumed to have the directing nodes of the pre- s instruction. However, if an instruction follows a right angular ket and is not preceded by a directing-node-definition, the di- ing nodes of this instruction are assumed to be, not those of the ious instruction, but rather the directing nodes of the instruc- preceding the corresponding left angular brackets. For instance control-programs (6) and (7) are equivalent:

(6) CP @(S,NP,VP,N,V) T1; T2; < @(N,V)T3; T4 >; T5; T6.

(7) CP @(S,NP,VP,N,V) T1; T2; @(N,V)T3; T4; @(S,NP,VP,N,V)T5; T6.

If an instruction is a group-number, it is equivalent to the sclist of the members of the group. For example, if the group of transformations T1, T2, T3, T4, and T5 has the number II, the control program (8) is equivalent to control program (9).

(8) CP @(N,V) II.

(9) CP @(N,V) T1; T2; T3; T4; T5.

In particular, the directing nodes are the same for all the members of the group.

9.2.1 IN-instruction

The IN-instruction restricts application of rules so that they apply to a single subtree, instead of applying bottom-to-top throughout a tree.

IN-instruction ::= IN transformation-name (integer)
DO < control-program >

The directing transformation of an IN-instruction is the transformation whose name follows the prefix IN of the IN-instruction. It specifies the single subtree to which the rules appearing in the DO-range, i.e., between the angular brackets following the prefix DO, are restricted. The directing transformation is invoked first; if it does not apply,

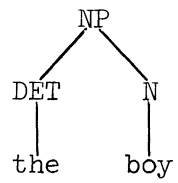
the instructions in the DO-range are skipped; if it applies, it indexes some nodes in the tree. The subtree to which the rules in the DO-range are restricted is the subtree which is headed by the node indexed by the integer following the transformation-name in the IN-instruction. For example, let us consider the transformation (10) and the control-program (11).

(10) TRANS DIR. SD (NP < % 2* >, VP < % 2* >).

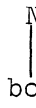
(11) CP @(NP,VP) IN DIR(2) DO<@(N)T1; @(V)T2; @(N,V)T3>.

If the tree for which the directing transformation DIR is invoked is tree (12), the last daughter of the node NP is indexed with the integer 2 and the subtree (13) is the only subtree to which rules T1, T2, and T3 may apply. Actually, rule T2 is not invoked since subtree (13) is not headed by a node labelled V.

(12)



(13)



The directing transformation of an IN-instruction is invoked as an ordinary transformation: if it is not within the DO-range of another IN-instruction, it applies bottom-to-top and if it is within the DO-range of another IN-instruction, it may apply only to the subtree of this instruction.

Example 1

In this example, there are two types of rules: stress rules T1, T2, T3, T5, T6, T7 which apply from bottom-to-top to all subtrees headed by one of the labelled nodes N, V, NP and VP, and word level rules T4, T8, T9, T10 which apply only to subtrees that are marked |+WORD|.

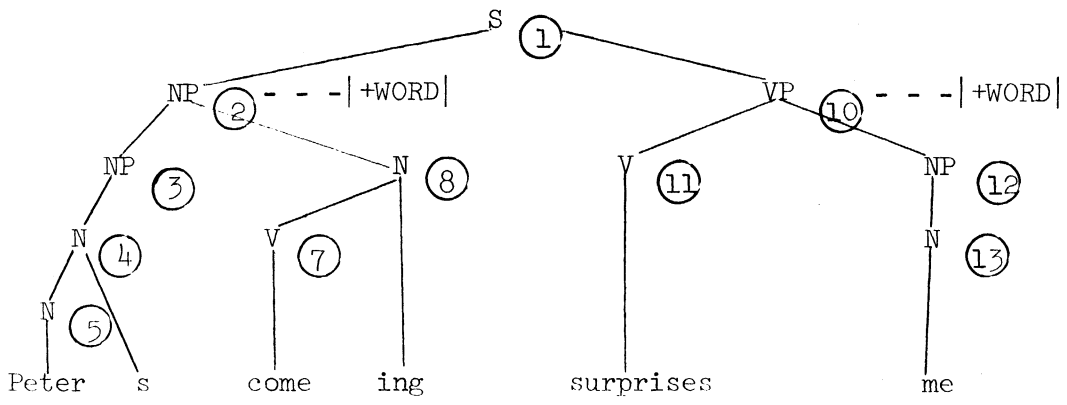
Directing transformation

```
TRANS  CYCLE  AC.  SD  1*.
```

Control program

```
CP @(N,V,NP,VP)  IN CYCLE(1)  DO
< T1; T2; T3; < @(*|+WORD|)T4 >; T5; T6; T7; @(*|+WORD|)
  T8; T9; T10 >.
```

Tree



The transformation CYCLE matches the head node of each subtree for which it is invoked. Therefore the subtree defined by CYCLE is always the tree it is invoked for.

subtree (13) rules T1, T2, T3, T5, T6 and T7 are invoked.

subtree (12) idem

subtree (11) idem

subtree (10) rules T1, T2, T3, T4, T5, T6, T7, T8, T9, T10 are invoked since the head node of (10) matches the node (VP) of the directing-node-definition @(N,V,NP,VP) and the node of the directing definition @(*|+WORD|)

etc....

Example 2

The same ordering is defined in this example, but this time using boundary markers instead of node markers to define a "word."

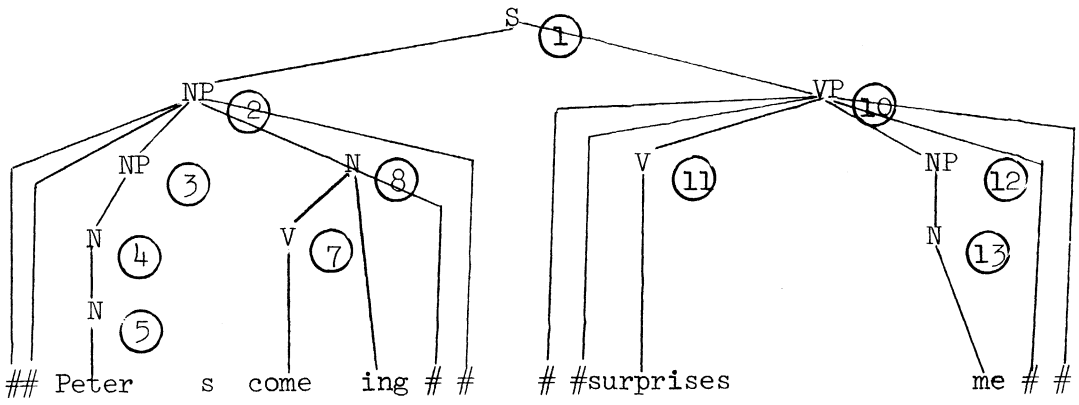
Directing transformations

```
TRANS  CYCLE  AC.  SD  1*.

TRANS  WORD   AC.  SD  1* < ## % ## >.
```

Control program

```
CP @(N,V,NP,VP)  IN CYCLE(1)DO <

   < T1; T2; T3;  IN WORD(1)DO < T4 >; T5; T6; T7;

     IN WORD(1)  DO < T8; T9; T10 >>.
```

Tree



The mechanism of application is similar to the previous. Transformation CYCLE matches the head of each subtree for which it is invoked, thus defining the next lowest permissible subtree. Transformation WORD applies only to subtrees which are bounded by two boundary markers ## on each side.

subtree (13) rules T1, T2, T3 are invoked.
transformation WORD is invoked but does not apply
and therefore the rule T4 is skipped.
rules T5, T6, and T7 are invoked
transformation WORD is invoked and does not apply,
rules T8, T9 and T10 are skipped.

subtree (12) idem

subtree (11) idem

subtree (10) this time when the transformation WORD is invoked it applies. It defines the subtree it was invoked for. Rules T1 to T10 apply in this order to the subtree (10).

etc....

9.2.2 IF-instruction

IF-instruction allows conditional invocation.

9.07 IF-instruction ::= IF instruction THEN instruction
opt[ELSE instruction]

The instruction following the prefix IF is carried out first. If one of the rules involved in this instruction has applied, then the instruction following the prefix THEN is carried out and the instruction following the prefix ELSE, if any is skipped; otherwise the instruction following the prefix THEN is skipped and the instruction following the prefix ELSE, if any, is carried out.

Example 1

Control program

@(N,V) IF T1 THEN T2.

Tree (2)

Transformation T1 is invoked for each of the subtrees (13), (11), (7), (8), (5) and (4). If one of these invocations is successful, transformation T2 is invoked for each of the same subtrees.

Example 2

In this example, the rule STRESSPLACEMENT introduces a main stress |+STRESS| in one of the segments. Everytime this rule applies, all other segments which are marked for stress must increase their stress

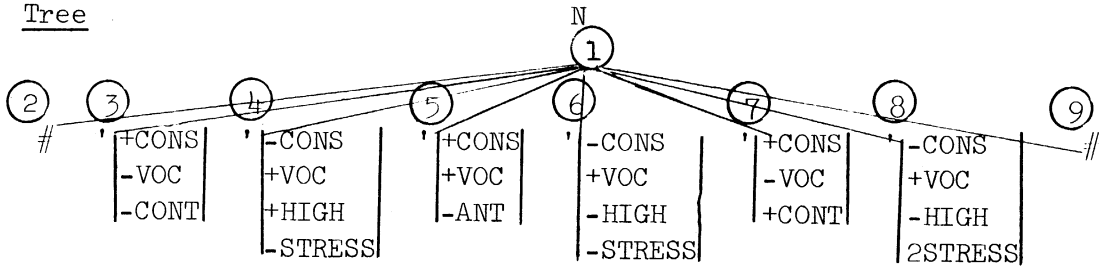
by one unit, that is, the transformation STRESSADJ must be invoked.

```
RULE  STRESSPLACEMENT  AC.

V  ⇒  |+STRESS|  /<% _ ('O'C 'V) 'O'C #>.

TRANS STRESSADJ AACC.

SD  1'|(ALPHA)STRESS|,  WHERE  ALPHA VNE -.

SC  IF < ALPHA VEQ + >  THEN <|1STRESS|  MERGEF  1 >
                        ELSE <|(ALPHA+1)STRESS|  MERGEF  1 >.
```

Control program

```
CP @(N,V,NP,VP)  IN CYCLE(1)  DO < IF  STRESSPLACEMENT
                                   THEN  STRESSADJUST >.
```

Tree



The transformation CYCLE defines the tree on which the transformations STRESSPLACEMENT and STRESSADJ will be invoked. The transformation STRESSPLACEMENT is invoked and inserts the feature specification |+STRESS| in the complex-symbol of node (6). The transformation STRESSADJ is then invoked and changes the feature specification |+STRESS| to |1STRESS| in node (6) and the feature specification |2STRESS| to |3STRESS| in node (8).

9.2.3 RPT-instruction

RPT-instruction allows repetitive invocation of a transformation, a group of transformations, or a control-program. There are two forms for the RPT-instruction: the first form does not have any upper bound on the number of repetitions, the second specifies an upperbound by an integer following the prefix RPT. The repeat instruction indicates that the instructions in the RPT range are to be iterated until either the upper bound, if any, is reached or until none of the transformations in the RPT range applies. For example the control program

CP RPT 5 < T1; T2; III >

repeats the sequence: invoke transformation T1, invoke transformation T2, invoke every transformation in group III, until either none of them apply or five iterations of the sequence have occurred.

9.08 RPT-instruction ::= RPT opt[integer] control-program

9.2.4 STOP-instruction

9.09 STOP-instruction ::= STOP

The STOP-instruction terminates the execution of a control program, and may appear at any point in the control program. A STOP-instruction is assumed before the terminal period of the control program. Therefore, a STOP-instruction need not appear within a control program. The STOP-instruction forces an output of the final tree and

lists the transformations which have applied in order of application.

9.2.5 TRACE-instruction

9.10 TRACE-instruction ::= TREE

Everytime the TRACE-instruction TREE appears in the control program, it causes the tree to output. For example, the control program (14) causes a tree to be output after every invocation of rule T7.

```
(14)  CP @(N,NP,V,VP)  IN CYCLE(1)  DO
          < T1; T2; T3; < @(*|+WORD|)T4 >; T5; T6; T7; TREE;
            @(*|+WORD|) T8; T9; T10 >.
```

A warning message is issued when a strong possibility of error exists. It has the same general form as the error message where WARNING replaces ERROR. For example:

WARNING. NUMNAM. FEATURE HIGH ADDED AS INHERENT

10.3 PROGRAM STRUCTURE

The program is built up of subroutines. Figure 1 is a simplified schematic description of the different subroutines and gives a basic representation of the program structure. Arrows point from calling subroutine to called subroutine. Table I is a brief description of each of the subroutines shown in Figure 1. These descriptions are incomplete and serve only to identify the subroutine mentioned in Figure 1.

This schematic representation shows that MAIN is the controlling program. MAIN consists almost entirely of subroutine calls. A run begins with a call to the input subroutine for grammars (GRAMIN). GRAMIN in turn calls subroutines INIT, which initializes the program, PHONIN, which reads the conversion lexicon, RULEIN, which reads the redundancy rules, and finally TRANIN, which reads the transformations. Control then returns to MAIN which calls FTRIN for tree input, and then calls CONTRL. The subroutine CONTRL first interprets the user's control program and applies the transformations: ANTEST tests the analyzability of the subtree and CHANGE carries the structural change if the analysis is successful. The subroutine TROUT prints the final tree. The process repeats with new inputs.

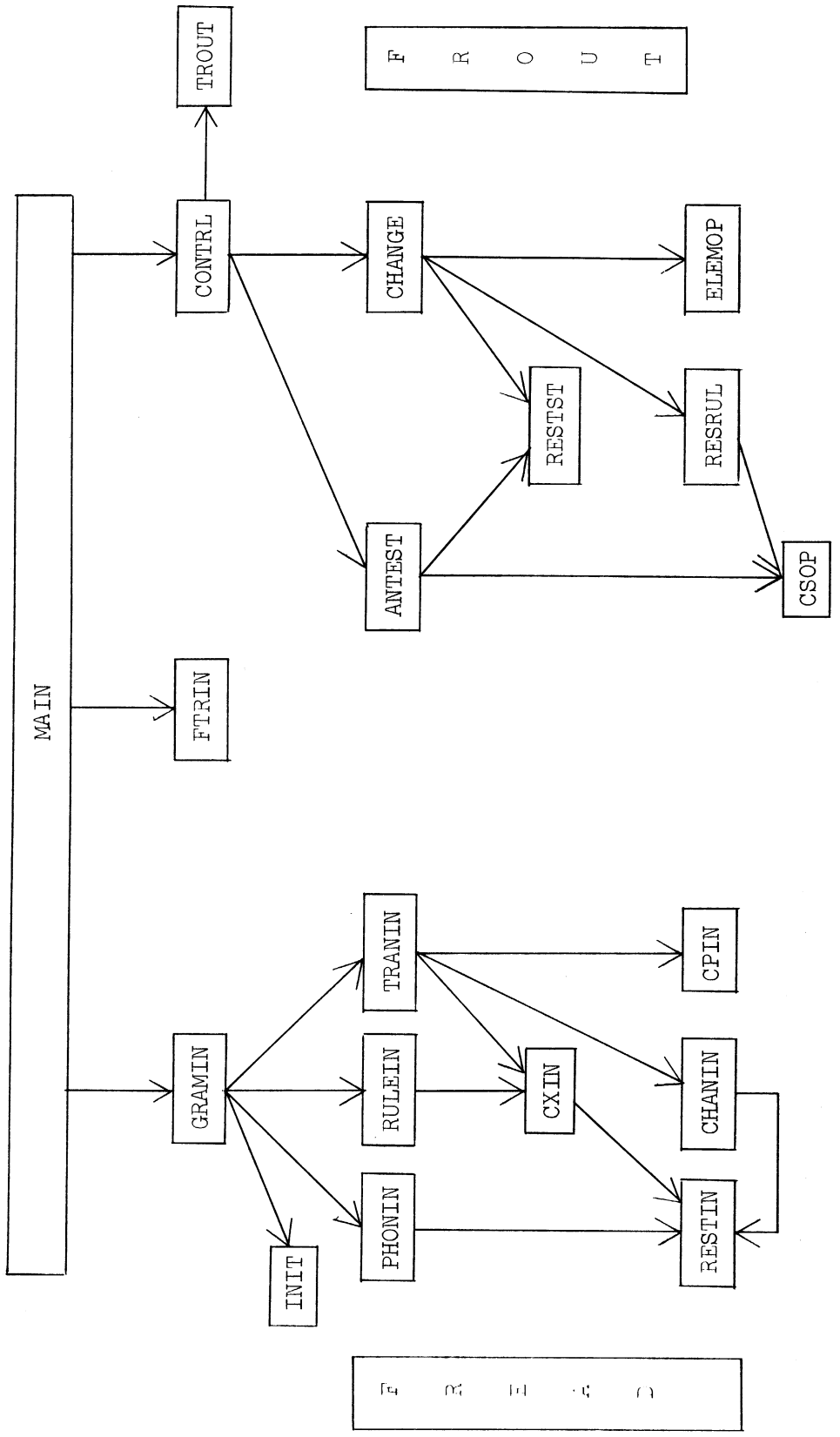


Figure 1. Schematic program structure.

TABLE I

Note: The actual structure of the program has been greatly simplified by the omission of subroutines called from one routine only. Also omitted are output programs, which exist for almost every part of a grammar.

Routine	Role
1.1 Main program	
MAIN	Reads the user's directions for the current run and carries them out.
1.2 Free-field input-output	
FREAD	Free-field read. Returns a word or special character for each call.
FROUT	Free-field write. Outputs a designated area of storage.
1.3 Trees	
TROUT	Outputs a tabular tree.
FTRIN	Inputs a linear tree specification.
1.4 Grammar input	
INIT	Initializes everything.
GRAMIN	Reads in a transformational grammar.
PHONIN	Reads in a conversion lexicon.
RULEIN	Reads in redundancy rules.
CPIN	Reads in the control program.
CHANIN	Reads in a structural change.
CXIN	Reads in a structural description or a complex symbol.
RESTIN	Reads in a restriction.
1.5 Analysis	
ANTEST	Evaluates a structural description against a subtree.
1.6 Restrictions	
RESTST	Test a restriction.

TABLE I (Concluded)

Routine	Role
1.7 Structural change	
CHANGE	Performs the structural change.
ELEMOP	Does the operations.
1.8 Complex symbol operations	
REDRUL	Applies the redundancy rules to expand a complex symbol.
CSOP	Tests complex symbols, performs complex symbol operations.
1.9 Control program	
CONTRL	Interprets the control program.

APPENDIX A

TWO EXAMPLES OF GRAMMAR AND DERIVATION

EXAMPLE 1: PHONOLOGICAL RULES

For this example, we have transcribed some of the phonological rules of The Sound Pattern of English that appear in the derivation of the words courage and courageous.

```
PHONLEXICON

    FEATURES
          CONS VOC HIGH BACK LOW ANT ROUND TENSE STRESS RED COR VOICED
          STRID CONT DER    A N WORD.
    VARIABLE
          OO = |-CONS +VOC -HIGH +BACK -LOW -ANT +ROUND|,
          O  = |-CONS +VOC -HIGH +BACK +LOW -ANT +ROUND|,
          A  = |-CONS +VOC -HIGH -BACK +LOW -ANT -ROUND|,
          E  = |-CONS +VOC -HIGH -BACK -LOW -ANT -ROUND|,
          EH = |-CONS +VOC -HIGH -BACK -LOW -ANT -ROUND -TENSE -STRESS
               +RED|,
          R  = |+CONS +VOC -ANT +COR +VOICED -STRID +CONT|,
          S  = |+CONS -VOC +ANT +COR -VOICED +STRID +CONT|,
          K  = |+CONS -VOC -ANT -COR -VOICED -STRID -CONT|,
          G  = |+CONS -VOC -ANT -COR +VOICED -STRID -CONT|,
          GD = |+CONS -VOC -ANT -COR +VOICED -STRID -CONT +DER|,
          V  = |-CONS +VOC|,
          C  = ¬ V .

    PHONUNIT
          UH = |-CONS +VOC -HIGH +BACK -LOW -ANT -ROUND|,
          OO=OO, O=O, A=A, EH=EH, E=E,
          Y  = |-CONS -VOC +HIGH -BACK -ROUND|,
          W  = |-CONS -VOC +HIGH +BACK +ROUND|,
          J  = |+CONS -VOC -ANT +COR +VOICED +STRID -CONT|,
          R=R, S=S, K=K, G=G.

    DIACRITIC
          :  = |+TENSE|,
          '  = |1STRESS|,
          *  = |2STRESS|,
          *  = |3STRESS|.
$ENDLEX
TRANSFORMATIONS
IMPLICIT AACC.
"    PHONOLOGICAL RULES                                                    "
"    THE RULES ARE REFERENCED BY THEIR INSET AND PAGE IN SPE               "
"    FOR INSTANCE, (36)77 REFERS TO RULE (36) ON PAGE 77 OF SPE.           "


   "REDUNDANCY RULES"
RULE REDVOWL !V.   V  => |-STRESS -TENSE|.

   "VELAR SOFTENING "
TRANS VELSOFT 1.   SD % 1'|-ANT -CONT +DER| '|-CONS -BACK -LOW| %.
                   SC |+COR +STRID| MERGEF 1 ,
                      IF <1 INC1 |-VOICED|> THEN <|+ANT| MERGEF 1>.

   "MAIN STRESS -VERSION(36)77, MODIFIED (45)82 "
RULE MSTR AC.  V => |+STRESS| /<2*/<% _  '0'C ('|-CONS +VOC -TENSE| ('C))
                          3( 4( + '0'C ) '|-CONS +VOC -TENSE|  '0'C )>,
    WHERE (NUL 3 | 2 INC1 |+A| | 2 INC1 |+N|) & (NUL 4 | 2 INC1 |+A|)>.
```

```
   "AUXILIARY REDUCTION - SIMPLIFIED FORM (20-I)240 "
RULE AUXRED AC.     V  => |-STRESS|
                   /<% _|(ALPHA)STRESS| ('C) '|(BETA)STRESS| %,
                    WHERE (BETA VLT 4) & (ALPHA VGT BETA)>.

   "TENSING, PART (B) OF RULE (23-IV)242 "
RULE TENSING II.    |-CONS +VOC -HIGH| => |+TENSE|
                   /<% _ 'C ('|(ALPHA)VOC (ALPHA)CONS -ANT|)
                    '|-CONS -BACK -LOW -STRESS| (+) 'V %>.

   "DIPHTONGIZATION - (31)243 "
RULE DIPHT.         * => |-CONS -VOC +HIGH (ALPHA)BACK (ALPHA)ROUND|
                   /<% '|-CONS +VOC (ALPHA)BACK +TENSE|  _ %>.

   "VOWEL SHIFT (34)187 "
TRANS VOSHIFT .
 SD % 1'|-CONS +VOC +TENSE| % .
 SC  IF<1 INC1 |(ALPHA)HIGH -LOW|> THEN<|(-ALPHA)HIGH| MERGEF 1> ,
     IF<1 INC1 |-HIGH (BETA)LOW|>  THEN<|(-BETA)LOW| MERGEF 1> .

   "ROUNDING ADJUSTMENT - MODIFIED FROM (34)244 "
RULE RADJ.          |+VOC -CONS (ALPHA)ROUND +BACK| => |(-ALPHA)ROUND|
                   /<% ( _|-TENSE|, _|(BETA)LOW (BETA)ROUND +TENSE|,
                    _ 'V) ('C %)>.

   " E- ELISION"
RULE EELI.          |-CONS -HIGH -BACK -LOW| => * /<% _ (+ %)>.

   "VOWEL REDUCTION"
RULE EHRED.         |-CONS +VOC -STRESS -TENSE| => EH.


   "TRANSFORMATION DEFINING THE CYCLE"
TRANS CYCLE AC III.SD 1* .
   "TRANSFORMATION ADJUSTING THE STRESS AFTER A MAIN STRESS RULE"
TRANS  ADST.        SD % 1'|(ALPHA)STRESS| %, WHERE ALPHA VNE - .
                   SC IF<ALPHA VEQ +> THEN< |1STRESS| MERGEF 1>
                                        ELSE< |(ALPHA+1)STRESS| MERGEF 1>.

   "CONTROL PROGRAM DETERMINING THE ORDERING"
CP @(*|+WORD|) IV ; @(N,A) IN CYCLE(1) DO
   <<@(*|+WORD|)VELSOFT>; IF MSTR THEN ADST; TREE; AUXRED;
     @(*|+WORD|) II>.
$END $MAIN FTRIN TRAN .

"  THE UNDERLYING FORM OF THE WORDS COURAGE AND COURAGEOUS ARE INPUT.  "

 N|+N +WORD| <'K 'OO 'R 'A 'GD 'E >.
 A|+A +WORD|<N|+N|< 'K 'OO 'R 'A 'GD 'E > + 'O 'S>.
```

```
 TREE READ BY FTRIN
  1 N          2 K
               3 OO
               4 R
               5 A
               6 G
               7 E
NODE    1 N
          |+N +WORD|

          K OO R A G E


          |+DER|
  6 G

*****      TRANSFORMATIONS      *****
SCAN CALLED AT      4      @
SCAN CALLED AT      5      IV
ANTEST CALLED FOR     1"REDVOWL "(AACC)     ,SD=  2. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       5
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       7
SCAN CALLED AT      6      ;
SCAN CALLED AT     12      @
SCAN CALLED AT     13      IN
SCAN CALLED AT     14      CYCLE
SCAN CALLED AT     15      (
SCAN CALLED AT     16
SCAN CALLED AT     17      )
SCAN CALLED AT     18      DO
SCAN CALLED AT     19      <
ANTEST CALLED FOR    11"CYCLE    "(AC  )     ,SD= 12. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 1**
SCAN CALLED AT     20      <
SCAN CALLED AT     24      @
SCAN CALLED AT     25      VELSOFT
ANTEST CALLED FOR     2"VELSOFT "(AACC)     ,SD=  3. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
SCAN CALLED AT     26      >
SCAN CALLED AT     12      @
SCAN CALLED AT     27      ;
SCAN CALLED AT     28      IF
SCAN CALLED AT     29      MSTR
ANTEST CALLED FOR     3"MSTR     "(AC  )     ,SD=  4. RESTRICTION=  3. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
SCAN CALLED AT     30      THEN
SCAN CALLED AT     31      ADST
ANTEST CALLED FOR    12"ADST     "(AACC)     ,SD= 13. RESTRICTION=  8. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
SCAN CALLED AT     32      ;
SCAN CALLED AT     33      TREE
```

```
 TREE READ BY FTRIN
  1 N          2 K
               3 OO'
               4 R
               5 A
               6 J
               7 E
NODE   1 N
         |+N +WORD|

         K OO' R A J E


         |-TENSE|
  3 OO'
         |-TENSE -STRESS|
  5 A
         |+DER|
  6 J
         |-TENSE -STRESS|
  7 E
SCAN CALLED AT     34      ;
SCAN CALLED AT     35      AUXRED
ANTEST CALLED FOR     4"AUXRED  "(AC  )     ,SD=  5. RESTRICTION=  4. TOP=    1:N
SCAN CALLED AT     36      ;
SCAN CALLED AT     40      @
SCAN CALLED AT     41      ||
ANTEST CALLED FOR     5"TENSING "(AACC)     ,SD=  6. RESTRICTION=  0. TOP=    1:N
ANTEST CALLED FOR     6"DIPHT   "(AACC)     ,SD=  7. RESTRICTION=  0. TOP=    1:N
ANTEST CALLED FOR     7"VOSHIFT "(AACC)     ,SD=  8. RESTRICTION=  0. TOP=    1:N
ANTEST CALLED FOR     8"RADJ    "(AACC)     ,SD=  9. RESTRICTION=  7. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
ANTEST CALLED FOR     9"EELI    "(AACC)     ,SD= 10. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. CALL ELEMOP FOR  ERASE         0    7
ANTEST CALLED FOR    10"EHRED   "(AACC)     ,SD= 11. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       5
SCAN CALLED AT     42      >
SCAN CALLED AT     12      @
SCAN CALLED AT     43      .
```

```
TRANSFORMATIONS WHICH HAVE APPLIED ARE
                1          1   REDVOWL
                2         11   CYCLE
                3          2   VELSOFT
                4          3   MSTR
                5         12   ADST
                6          8   RADJ
                7          9   EELI
                8         10   EHRED


 TREE READ BY FTRIN
  1 N           2 K
                3 UH'
                4 R
                5 EH
                6 J
NODE    1 N
          |+N +WORD|

          K UH' R EH J


          |-TENSE|
  3 UH'
          |+DER|
  6 J
```

```
 TREE READ BY FTRIN
  1 A          2 N          3 K
                            4 OO
                            5 R
                            6 A
                            7 G
                            8 E
               9 +
              10 O
              11 S
NODE    1 A
          |+A +WORD|
NODE    2 N
          |+N|

          K OO R A G E + O S


          |+DER|
  7 G

*****      TRANSFORMATIONS      *****
SCAN CALLED AT      4      @
SCAN CALLED AT      5      IV
ANTEST CALLED FOR     1"REDVOWL "(AACC)     ,SD=  2. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 4**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       8
CHANGE. HAVE CSEXCH FOR  MERGEF    IN      10
SCAN CALLED AT      6      ;
SCAN CALLED AT     12      @
SCAN CALLED AT     13      IN
SCAN CALLED AT     14      CYCLE
SCAN CALLED AT     15      (
SCAN CALLED AT     16
SCAN CALLED AT     17      )
SCAN CALLED AT     18      DO
SCAN CALLED AT     19      <
ANTEST CALLED FOR    11"CYCLE    "(AC  )     ,SD= 12. RESTRICTION=  0. TOP=    2:N
ANTEST RETURNS ** 1**
SCAN CALLED AT     20      <
SCAN CALLED AT     24      @
SCAN CALLED AT     25      VELSOFT
SCAN CALLED AT     26      >
SCAN CALLED AT     12      @
SCAN CALLED AT     27      ;
SCAN CALLED AT     28      IF
SCAN CALLED AT     29      MSTR
ANTEST CALLED FOR     3"MSTR     "(AC  )     ,SD=  4. RESTRICTION=  3. TOP=    2:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
SCAN CALLED AT     30      THEN
SCAN CALLED AT     31      ADST
ANTEST CALLED FOR    12"ADST     "(AACC)     ,SD= 13. RESTRICTION=  8. TOP=    2:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
SCAN CALLED AT     32      ;
SCAN CALLED AT     33      TREE
```

```
 TREE READ BY FTRIN
  1 A          2 N          3 K
                            4 OO'
                            5 R
                            6 A
                            7 G
                            8 E
               9 +
              10 O
              11 S
NODE    1 A
          |+A +WORD|
NODE    2 N
          |+N|

          K OO' R A G E + O S


          |-TENSE|
  4 OO'
          |-TENSE -STRESS|
  6 A
          |+DER|
  7 G
          |-TENSE -STRESS|
  8 E
          |-TENSE -STRESS|
 10 O
SCAN CALLED AT     34      ;
SCAN CALLED AT     35      AUXRED
ANTEST CALLED FOR     4"AUXRED  "(AC  )     ,SD=  5. RESTRICTION=  4. TOP=    2:N
SCAN CALLED AT     36      ;
SCAN CALLED AT     40      @
SCAN CALLED AT     41      ||
SCAN CALLED AT     42      >
SCAN CALLED AT     12      @
ANTEST CALLED FOR    11"CYCLE   "(AC  )     ,SD= 12. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
SCAN CALLED AT     20      <
SCAN CALLED AT     24      @
SCAN CALLED AT     25      VELSOFT
ANTEST CALLED FOR     2"VELSOFT "(AACC)     ,SD=  3. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       7
SCAN CALLED AT     26      >
SCAN CALLED AT     12      @
SCAN CALLED AT     27      ;
SCAN CALLED AT     28      IF
SCAN CALLED AT     29      MSTR
ANTEST CALLED FOR     3"MSTR    "(AC  )     ,SD=  4. RESTRICTION=  3. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
SCAN CALLED AT     30      THEN
SCAN CALLED AT     31      ADST
ANTEST CALLED FOR    12"ADST    "(AACC)     ,SD= 13. RESTRICTION=  8. TOP=    1:A
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
SCAN CALLED AT     32      ;
SCAN CALLED AT     33      TREE
```

```
 TREE READ BY FTRIN
  1 A          2 N          3 K
                            4 OO*
                            5 R
                            6 A'
                            7 J
                            8 E
               9 +
              10 O
              11 S
NODE    1 A
          |+A +WORD|
NODE    2 N
          |+N|

          K OO* R A' J E + O S


          |-TENSE|
   4 OO*
          |-TENSE|
   6 A'
          |+DER|
   7 J
          |-TENSE -STRESS|
   8 E
          |-TENSE -STRESS|
  10 O
SCAN CALLED AT     34      ;
SCAN CALLED AT     35      AUXRED
ANTEST CALLED FOR     4"AUXRED  "(AC  )     ,SD=  5. RESTRICTION=  4. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
SCAN CALLED AT     36      ;
SCAN CALLED AT     40      @
SCAN CALLED AT     41      ||
ANTEST CALLED FOR     5"TENSING "(AACC)     ,SD=  6. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
ANTEST CALLED FOR     6"DIPHT   "(AACC)     ,SD=  7. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. CALL ELEMOP FOR  ARISE       12     6
ANTEST CALLED FOR     7"VOSHIFT "(AACC)     ,SD=  8. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
ANTEST CALLED FOR     8"RADJ    "(AACC)     ,SD=  9. RESTRICTION=  7. TOP=    1:A
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
CHANGE. HAVE CSEXCH FOR  MERGEF    IN      10
ANTEST CALLED FOR     9"EELI    "(AACC)     ,SD= 10. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 1**
CHANGE. CALL ELEMOP FOR  ERASE        0     8
ANTEST CALLED FOR    10"EHRED   "(AACC)     ,SD= 11. RESTRICTION=  0. TOP=    1:A
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
CHANGE. HAVE CSEXCH FOR  MERGEF    IN      10
SCAN CALLED AT     42      >
SCAN CALLED AT     12      @
SCAN CALLED AT     43      .
```

```
TRANSFORMATIONS WHICH HAVE APPLIED ARE
                1          1  REDVOWL
                2         11  CYCLE
                3          3  MSTR
                4         12  ADST
                5         11  CYCLE
                6          2  VELSOFT
                7          3  MSTR
                8         12  ADST
                9          4  AUXRED
               10          5  TENSING
               11          6  DIPHT
               12          7  VOSHIFT
               13          8  RADJ
               14          9  EELI
               15         10  EHRED


 TREE READ BY FTRIN
  1 A          2 N          3 K
                            4 EH
                            5 R
                            6 E:'
                           12 Y
                            7 J
               9 +
              10 EH
              11 S
NODE    1 A
          |+A +WORD|
NODE    2 N
          |+N|

          K EH R E:' Y J + EH S


          |+DER|
  7 J
```

EXAMPLE 2: MARKEDNESS CONVENTIONS

For this example, we have transcribed the complete set of marking conventions relative to vowels and consonants described in The Sound Pattern of English (pp. 404-407).

```
"                    EXAMPLE                                                    "

"     MARKING CONVENTION RULES                                                  "
"     THE RULES PRESENTED HERE ARE TRANSCRIPTIONS OF THE MARKING                "
"     RULES DESCRIBED IN SPE, PAGES 404 TO 407.                                 "
"     THE RULES HAVE BEEN SIMPLIFIED WHEN THEY WERE SIMPLIFIABLE.               "
"     SOME OF THE RULES HAVE BEEN DECOMPOSED INTO TWO SUBRULES.                 "
PHONLEX
FEATURES
  CONS VOC SEG LOW HIGH BACK ROUND SON LATERAL TENSE
  NASAL VOICE ANT COR CONT DELRE STRID .
VARIABLE
 V = |-CONS +VOC|,
 C = |+CONS|  |  |-VOC|.
PHONUNIT
 #=|-SEG|,
 U=|+SEG -CONS +VOC -LOW +HIGH +BACK +ROUND |,
 N=|+SEG +CONS -VOC +NASAL -LOW -HIGH -BACK +VOICE +ANT +COR -CONT
    -DELRE -STRID|,
 S=|+SEG +CONS -VOC -NASAL -LOW -HIGH -BACK -VOICE +ANT +COR +CONT
    +DELRE +STRID|,
 T=|+SEG +CONS -VOC -NASAL -LOW -HIGH -BACK -VOICE +ANT +COR -CONT
    -DELRE -STRID|.
 $ENDLEX
TRANSFORMATIONS
IMPLICIT AACC.
RULE R1 I.          |(U)SEG|     => |(-U)SEG|.
"CONSONANTAL"
RULE R2A AC II.     |(U)CONS|    =>|(U)CONS|
                   /<% ( ('|-SEG|, 'V) _ , _|-VOC| ) %>.
RULE R2B AC.        |(U)CONS|    => |(-U)CONS| /<% _|+VOC| %>.
"VOCALIC"
RULE R3A AC.        |(U)VOC|     => |(U)VOC| /<% 'C _ %>.
RULE R3B AC.        |(U)VOC|     => |(-U)VOC|
                   /<% ( _|+CONS|, ('V,'|-SEG|) _ ) %>.
RULE R4 III.        |+VOC|       => |+SON|.
```

```
"                    CONVENTIONS FOR THE VOWELS                              "
RULE R5.             |+VOC -CONS| => |-ANT -STRID +CONT +VOICE -LATERAL|.
"LOW"
RULE R6A. |-CONS +VOC ¬BACK ¬ROUND (U)LOW| => |(U)LOW|.
RULE R6B. |-CONS +VOC (U)LOW|   => |(-U)LOW|.
"HIGH"
RULE R7.  |-CONS +VOC +LOW|     => |-HIGH|.
RULE R8.  |-CONS +VOC (U)HIGH| => |(U)HIGH|.
"RULE R9 IS REDUNDANT"
"BACK"
RULE R10. |-CONS +VOC +LOW (U)BACK| => |(U)BACK|.
"ROUND"
RULE R11A.|-CONS +VOC +BACK -LOW (U)ROUND| =>|(U)ROUND|.
RULE R11B.|-CONS +VOC (U)ROUND|=> |(-U)ROUND|.
"TENSE"
RULE R12. |-CONS +VOC (U)TENSE|=> |(U)TENSE|.
"                    CONVENTIONS FOR THE CONSONANTS                          "
"NASAL"
RULE R13. |+CONS -VOC (U)NASAL|=> |(-U)NASAL|.
RULE R14. |+CONS -VOC -NASAL|   => |-SON|.
RULE R15. |+CONS -VOC +NASAL|   => |+SON -CONT -STRID|.
"LOW"
RULE R16. |+CONS -VOC (U)LOW|   => |(-U)LOW|.
RULE R17. |+CONS -VOC +LOW|     => |-HIGH|.
"HIGH"
RULE R18A.|+CONS -VOC ¬ANT ¬BACK (U)HIGH| => |(-U)HIGH|.
RULE R18B.|+CONS -VOC (U)HIGH| => |(U)HIGH|.
"RULE R19 IS REDUNDANT"
"BACK"
RULE R20A.|+CONS -VOC ¬ANT -LOW (U)BACK| => |(-U)BACK|.
RULE R20B.|+CONS -VOC (U)BACK| => |(U)BACK|.
"VOICE"
RULE R21A.|+CONS -VOC -SON (U)VOICE| => |(-U)VOICE|.
RULE R21B.|+CONS -VOC (U)VOICE|=> |(U)VOICE|.
"ANTERIOR"
RULE R22A.|+CONS -VOC +HIGH +COR (ALPHA)CONT (U)ANT| => |(-U)ANT|.
RULE R22B.|+CONS -VOC (U)ANT|   => |(U)ANT|.
"CORONAL"
RULE R23A.|+CONS -VOC -ANT (U)COR| => |(-U)COR|
         /<%( _|+NASAL|, _|+BACK|)%>.
RULE R23B.|+CONS -VOC (U)COR|   => |(U)COR|.
"CONTINUANT"
RULE R24A.|+CONS -VOC (U)CONT| => |(U)CONT| /<% '|-SEG| _ '|+CONS| %>.
RULE R24B.|+CONS -VOC (U)CONT| => |(-U)CONT|.
"DELAYED RELEASE"
RULE R25. |+CONS -VOC +CONT|    => |+DELRE|.
RULE R26A.|+CONS -VOC -ANT +COR (U)DELRE| => |(U)DELRE|.
RULE R26B.|+CONS -VOC (U)DELRE|=> |(-U)DELRE|.
"STRIDENT"
RULE R27A.|+CONS -VOC (U)STRID|=> |(-U)STRID|
         /<% ( _|+SON|, _|-ANT -COR|, _|-DELRE| ) %>.
RULE R27B.|+CONS -VOC (U)STRID|=> |(U)STRID|.
CP @(N) | ; RPT < || > ; |||.
$END $MAIN FTRIN TRAN.
 N<'|| '|SEG| '|SEG VOC +COR| '|SEG +BACK| '|SEG NASAL| '||>.
```

```
TREE READ BY FTRIN
 1 N          2 '
              3 '
              4 '
              5 '
              6 '
              7 '

         ' ' ' ' ' ' '


  3 '     |SEG|
  4 '     |VOC SEG +COR|
  5 '     |SEG BACK|
  6 '     |SEG NASAL|



*****     TRANSFORMATIONS      *****
SCAN CALLED AT      4      @
SCAN CALLED AT      5      |
ANTEST CALLED FOR     1"R1       "(AACC)     ,SD=  2. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 6**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       2
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       5
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       6
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       7
SCAN CALLED AT      6      ;
SCAN CALLED AT      7      RPT
SCAN CALLED AT      8      <
SCAN CALLED AT      9      ||
ANTEST CALLED FOR     2"R2A      "(AC  )     ,SD=  3. RESTRICTION=  2. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
ANTEST CALLED FOR     3"R2B      "(AC  )     ,SD=  4. RESTRICTION=  3. TOP=    1:N
ANTEST CALLED FOR     4"R3A      "(AC  )     ,SD=  5. RESTRICTION=  0. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
ANTEST CALLED FOR     5"R3B      "(AC  )     ,SD=  6. RESTRICTION=  4. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       3
SCAN CALLED AT     10      >
SCAN CALLED AT      4      @
SCAN CALLED AT      8      <
SCAN CALLED AT      9      ||
ANTEST CALLED FOR     2"R2A      "(AC  )     ,SD=  3. RESTRICTION=  2. TOP=    1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF    IN       4
ANTEST CALLED FOR     3"R2B      "(AC  )     ,SD=  4. RESTRICTION=  3. TOP=    1:N
```

```
ANTEST CALLED FOR    4"R3A     "(AC  )    ,SD=  5. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
ANTEST CALLED FOR    5"R3B     "(AC  )    ,SD=  6. RESTRICTION=  4. TOP=   1:N
SCAN CALLED AT    10     >
SCAN CALLED AT     4     @
SCAN CALLED AT     8     <
SCAN CALLED AT     9     ||
ANTEST CALLED FOR    2"R2A     "(AC  )    ,SD=  3. RESTRICTION=  2. TOP=   1:N
ANTEST CALLED FOR    3"R2B     "(AC  )    ,SD=  4. RESTRICTION=  3. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
ANTEST CALLED FOR    4"R3A     "(AC  )    ,SD=  5. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR    5"R3B     "(AC  )    ,SD=  6. RESTRICTION=  4. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
SCAN CALLED AT    10     >
SCAN CALLED AT     4     @
SCAN CALLED AT     8     <
SCAN CALLED AT     9     ||
ANTEST CALLED FOR    2"R2A     "(AC  )    ,SD=  3. RESTRICTION=  2. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR    3"R2B     "(AC  )    ,SD=  4. RESTRICTION=  3. TOP=   1:N
ANTEST CALLED FOR    4"R3A     "(AC  )    ,SD=  5. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR    5"R3B     "(AC  )    ,SD=  6. RESTRICTION=  4. TOP=   1:N
SCAN CALLED AT    10     >
SCAN CALLED AT     4     @
SCAN CALLED AT     8     <
SCAN CALLED AT     9     ||
ANTEST CALLED FOR    2"R2A     "(AC  )    ,SD=  3. RESTRICTION=  2. TOP=   1:N
ANTEST CALLED FOR    3"R2B     "(AC  )    ,SD=  4. RESTRICTION=  3. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR    4"R3A     "(AC  )    ,SD=  5. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR    5"R3B     "(AC  )    ,SD=  6. RESTRICTION=  4. TOP=   1:N
SCAN CALLED AT    10     >
SCAN CALLED AT     4     @
SCAN CALLED AT     8     <
SCAN CALLED AT     9     ||
ANTEST CALLED FOR    2"R2A     "(AC  )    ,SD=  3. RESTRICTION=  2. TOP=   1:N
ANTEST CALLED FOR    3"R2B     "(AC  )    ,SD=  4. RESTRICTION=  3. TOP=   1:N
ANTEST CALLED FOR    4"R3A     "(AC  )    ,SD=  5. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR    5"R3B     "(AC  )    ,SD=  6. RESTRICTION=  4. TOP=   1:N
SCAN CALLED AT    10     >
SCAN CALLED AT     4     @
SCAN CALLED AT    11     ;
```

```
SCAN CALLED AT    12     111
ANTEST CALLED FOR    6"R4      "(AACC)    ,SD=  7. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR    7"R5      "(AACC)    ,SD=  8. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR    8"R6A     "(AACC)    ,SD=  9. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR    9"R6B     "(AACC)    ,SD= 10. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
ANTEST CALLED FOR   10"R7      "(AACC)    ,SD= 11. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR   11"R8      "(AACC)    ,SD= 12. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
ANTEST CALLED FOR   12"R10     "(AACC)    ,SD= 13. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR   13"R11A    "(AACC)    ,SD= 14. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
ANTEST CALLED FOR   14"R11B    "(AACC)    ,SD= 15. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR   15"R12     "(AACC)    ,SD= 16. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      5
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      7
ANTEST CALLED FOR   16"R13     "(AACC)    ,SD= 17. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   17"R14     "(AACC)    ,SD= 18. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
ANTEST CALLED FOR   18"R15     "(AACC)    ,SD= 19. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   19"R16     "(AACC)    ,SD= 20. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   20"R17     "(AACC)    ,SD= 21. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR   21"R18A    "(AACC)    ,SD= 22. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   22"R18B    "(AACC)    ,SD= 23. RESTRICTION=  0. TOP=   1:N
```

```
ANTEST CALLED FOR   23"R20A    "(AACC)    ,SD= 24. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   24"R20B    "(AACC)    ,SD= 25. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR   25"R21A    "(AACC)    ,SD= 26. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
ANTEST CALLED FOR   26"R21B    "(AACC)    ,SD= 27. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   27"R22A    "(AACC)    ,SD= 28. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR   28"R22B    "(AACC)    ,SD= 29. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 3**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   29"R23A    "(AACC)    ,SD= 30. RESTRICTION=  5. TOP=   1:N
ANTEST CALLED FOR   30"R23B    "(AACC)    ,SD= 31. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   31"R24A    "(AACC)    ,SD= 32. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
ANTEST CALLED FOR   32"R24B    "(AACC)    ,SD= 33. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
ANTEST CALLED FOR   33"R25     "(AACC)    ,SD= 34. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
ANTEST CALLED FOR   34"R26A    "(AACC)    ,SD= 35. RESTRICTION=  0. TOP=   1:N
ANTEST CALLED FOR   35"R26B    "(AACC)    ,SD= 36. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 2**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      6
ANTEST CALLED FOR   36"R27A    "(AACC)    ,SD= 37. RESTRICTION=  6. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      4
ANTEST CALLED FOR   37"R27B    "(AACC)    ,SD= 38. RESTRICTION=  0. TOP=   1:N
ANTEST RETURNS ** 1**
CHANGE. HAVE CSEXCH FOR  MERGEF   IN      3
SCAN CALLED AT    13      .
```

TRANSFORMATIONS WHICH HAVE APPLIED ARE

1	1	R1
2	2	R2A
3	4	R3A
4	5	R3B
5	2	R2A
6	4	R3A
7	3	R2B
8	5	R3B
9	2	R2A
10	4	R3A
11	3	R2B
12	6	R4
13	7	R5
14	8	R6A
15	9	R6B
16	10	R7
17	11	R8
18	12	R10
19	13	R11A
20	14	R11B
21	15	R12
22	16	R13
23	17	R14
24	18	R15
25	19	R16
26	21	R18A
27	23	R20A
28	25	R21A
29	26	R21B
30	28	R22B
31	30	R23B
32	31	R24A
33	32	R24B
34	33	R25
35	35	R26B
36	36	R27A
37	37	R27B

TREE READ BY FTRIN

1 N	2 #
	3 S
	4 T
	5 U
	6 N
	7 #

S T U N

APPENDIX B

COMPLETE SYNTAX FOR PHONOLOGICAL GRAMMAR

0.01 phonological-grammar ::=

opt[conversion-lexicon] opt[everywhere-rules] opt[transformations] $END

1.01 tree ::= complex-node opt[< list[tree] >]

1.02 complex-node ::= node opt[complex-symbol] ▯ dummy-node complex-symbol-reference

1.03 node ::= word ▯ sentence-symbol ▯ boundary-symbol

1.04 sentence-symbol ::= S

1.05 boundary-symbol ::= # ▯ + ▯ =

1.06 dummy-node ::= '

1.07 complex-symbol-reference ::= complex-symbol ▯ complex-symbol-name

2.01 structural-description ::= structural-analysis opt[, WHERE restriction]

2.02 structural-analysis ::= list[term]

2.03 term ::= opt[integer] structure ▯ opt[integer] choice ▯ skip

2.04 structure ::= complex-element opt[opt[-] opt[/] < structural-analysis >]

2.05 complex-element ::= element opt[complex-symbol] ▯ dummy-node node-name

2.06 element ::= node ▯ * ▯ _

2.07 node-name ::= complex-symbol ▯ complex-symbol-name ▯ input-name

2.08 choice ::= (clist[structural-analysis])

2.09 skip ::= % ▯ bounded-skip

2.10 bounded-skip ::= ' integer opt[, integer] dummy-node node-name

3.01 restriction ::= booleancombination[condition]

3.02 condition ::= unary-condition ▯ binary-condition

3.03 unary-condition ::= unary-relation integer

3.04 binary-condition ::= integer binary-tree-relation node-designator

▯ integer binary-complex-relation complex-symbol-designator

▯ p-value value-relation p-value

3.05 node-designator ::= integer ▯ node ▯ dummy-node

3.06 complex-symbol-designator ::= complex-symbol ▯ integer

3.07 p-value ::= sign ▯ integer ▯ variable

3.08 unary-relation ::= TRM ▯ NTRM ▯ NUL ▯ NNUL

3.09 binary-tree-relation ::= EQ ▯ NEQ ▯ DOM ▯ NDOM ▯ DOMBY ▯ NDOMBY

3.10 binary-complex-relation ::= INCl ▯ NINCl ▯ CSEQ ▯ NCSEQ ▯ NDST ▯ NNDST

3.11 value-relation ::= VLT ▯ VLE ▯ VEQ ▯ VGE ▯ VGT ▯ VNE

4.01 complex-symbol ::= | opt[list[feature-specification]] |

4.02 feature-specification ::= opt[value] inherent-feature

4.03 value ::= sign ▯ integer ▯ ¬ ▯ (prefix) ▯ *

4.04 prefix ::= sign-prefix ▯ integer-prefix

4.05 sign-prefix ::= opt[-] variable

4.06 integer-prefix ::= variable opt[sign integer]

4.07 sign ::= + ▯ -

4.08 variable ::= word

4.09 inherent-feature ::= word

5.01 structural-change ::= clist[change-instruction]

5.02 change-instruction ::= change ‖ conditional-change

5.03 conditional-change ::= IF < restriction > THEN < structural-change >

opt[ELSE < structural-change >]

5.04 change ::= unary-operator integer

‖ tree-designator binary-tree-operator integer

‖ complex-symbol-designator binary-complex-operator integer

‖ complex-symbol-designator ternary-complex-operator integer integer

5.05 tree-designator ::= (tree) ‖ node ‖ integer

5.06 unary-operator ::= ERASE

5.07 binary-tree-operator ::= ADLAD ‖ ADFID ‖ ADRIS ‖ ADLES ‖ SUBST ‖

ALADE ‖ AFIDE ‖ ARISE ‖ ALESE ‖ SUBSE ‖

ADCHR ‖ ADCHL ‖ ACHRE ‖ ACHLE

5.08 binary-complex-operator ::= ERASEF ‖ MERGEF ‖ SAVEF

5.09 ternary-complex-operator ::= MOVEF

6.01 conversion-lexicon ::= PHONLEX feature-lexicon

opt[input-name-lexicon]

opt[output-name-lexicon]

opt[diacritic-lexicon]

$ENDPHON

6.02 feature-lexicon ::= FEATURES list[feature] .

6.03 input-name-lexicon ::= VARIABLE clist[input-name-definition] .

6.04 output-name-lexicon ::= PHONUNIT clist[output-name-definition] .

6.05 diacritic-lexicon ::= DIACRIT clist[diacritic-definition] .

6.06 input-name-definition ::= complex-symbol-name = complex-symbol

[] input-name = booleancombination[complex-symbol-reference]

6.07 output-name-definition ::= output-name = complex-symbol-reference

6.08 diacritic-definition ::= diacritic = complex-symbol-reference

6.09 complex-symbol-name ::= word

6.10 output-name ::= symbol

6.11 diacritic ::= symbol

7.01 everywhere-rules ::= RULES clist[redundancy-rule] .

7.02 redundancy-rule ::= complex-symbol-reference => complex-symbol-reference

8.01 transformations ::= TRANSFORMATIONS opt[IMPLICIT parameters]

list[transformation] CP control-program . $END

8.02 transformation ::= TRANS identification . rule ▯ RULE identification . simple-rule

8.03 rule ::= SD structural-description . opt[SC structural-change .]

8.04 simple-rule ::= operand => operator opt[/ < structural-description >] .

8.05 operand ::= complex-symbol-reference ▯ input-name ▯ *

8.06 operator ::= complex-symbol-reference ▯ *

8.07 identification ::= opt[integer] transformation-name opt[list[parameter]]

8.08 transformation-name ::= word

8.09 parameter ::= group-number ▯ optionality ▯ repetition

8.10 group-number ::= I ▯ II ▯ III ▯ IV ▯ V ▯ VI ▯ VII

8.11 optionality ::= OB ▯ OP

8.12 repetition ::= AC ▯ ACAC ▯ AACC

9.01 control-program ::= sclist[opt[directing-nodes-definition] instruction]

9.02 directing-nodes-definition ::= @ (clist[complex-element])

9.03 instruction ::= transformation-element ▯ control-element ▯ < sclist[instruction] >

9.04 transformation-element ::= transformation-name ▯ group-number

9.05 control-element ::= IN-instruction ▯ IF-instruction ▯ RPT-instruction
▯ STOP-instruction ▯ TRACE-instruction

9.06 IN-instruction ::= IN transformation-name (integer) DO < control-program >

9.07 IF-instruction ::= IF instruction THEN instruction opt[ELSE instruction]

9.08 RPT-instruction ::= RPT opt[integer] < control-program >

9.09 STOP-instruction ::= STOP

9.10 TRACE-instruction ::= TREE

BIBLIOGRAPHY

Anderson, Lloyd B. 1968. The left-to-right syllabic cycle. PEGS papers 32.

Backus, J. W. 1959. The syntax and semantics of the proposed International Algebraic Language of the Zurich ACM-GAMM Conference. Proc. 1st International Conference Information Processing, UNESCO, Paris. London, 1960.

Bobrow, Daniel G., and J. B. Fraser. 1968. A phonological rule tester. CACM 11.766-72.

Chomsky, Noam, and M. Halle. 1968. The sound pattern of English. New York: Harper and Row.

Friedman, Joyce. 1969. A computer system for transformational grammar. CACM 12.341-48.

Friedman, Joyce and P. Myslenski. 1970. Computer experiments in transformational grammar: the UCLA English grammar. (Department of Computer and Communication Sciences, Working Papers in Computational Linguistics M11.) Ann Arbor: The University of Michigan.

Friedman, Joyce, et al., 1971. A computer model for transformational grammar. New York: American Elsevier Press.

Fromkin, Victoria A., and L. Rice. 1969. An interactive phonological rule testing system. International conference on computational linguistics, Stockholm.

Johnson, C. Douglas. 1970. Formal aspects of phonological description. (Project on Linguistic Analysis, Second Series 11.) Berkeley: University of California.

Harms, Robert T. 1966. The measurement of phonological economy. Language 42.602-11.

McCawley, James D. 1968. The phonological component of a grammar of Japanese. The Hague: Mouton.

McCawley, James D. 1970. On the role of notations in generative phonology. (Mimeograph.)

BIBLIOGRAPHY (Concluded)

Morin, Yves Ch. 1971. Computer experiments in generative phonology: Low-level French phonology. (Department of Computer and Communication Sciences, Natural Language Studies 11.) Ann Arbor: The University of Michigan.

Morin, Yves Ch., and J. Friedman. 1971. Phonological grammar tester: Underlying theory. (Department of Computer and Communication Sciences, Natural Language Studies 10.) Ann Arbor: The University of Michigan.

Naur, P., ed. 1960. Report on the Algorithmic Language ALGOL 60. CACM 3.299-314.

Stanley, Richard. 1967. Redundancy rules in phonology. Language 43. 393-436.