ANALYSIS IN TRANSFORMATIONAL GRAMMAR

BY

JOYCE FRIEDMAN AND THEODORE S. MARTNER

STANFORD UNIVERSITY COMPUTER SCIENCE DEPARTMENT
COMPUTATIONAL LINGUISTICS PROJECT
AUGUST 1968

AF - 34
CS - 111

ANALYSIS IN TRANSFORMATIONAL GRAMMAR

by

Joyce Friedman* and Theodore S. Martner

*Present address: Computer and Communication Sciences Department,
University of Michigan, Ann Arbor, Michigan.

ABSTRACT

In generating sentences by means of a transformational grammar, it is necessary to analyze trees, testing for the presence or absence of various structures. This analysis occurs at two stages in the generation process -- during insertion of lexical items (more precisely, in testing contextual features), and during the transformation process, when individual transformations are being tested for applicability.

In this paper we describe a formal system for the definition of tree structure of sentences. The system consists of a formal language for partial or complete definition of the tree structure of a sentence, plus an algorithm for comparison of such a definition with a tree. It represents a significant generalization of Chomsky's notion of "proper analysis", and is flexible enough to be used within any transformational grammar which we have seen.

TABLE OF CONTENTS

Introduction

The notion of analysis described here is an outgrowth of a project which had as its primary goal the writing of a computer system to aid transformational grammarians [3]. Early in this project we realized that certain aspects of transformational grammar theory had never received the sort of formalization necessary for computer applications; this paper is essentially a description of our attempt to correct this situation in one such area. It should be noted that rigorous formalization is not simply an ad hoc matter in order to be able to use the computer; questions of the relative simplicity of grammars are realistically answerable only when the grammars have been placed in a precise system of notation, and, more important, a transformational generative grammar cannot be said to have succeeded in defining a language unless it is possible to generate sentences by using the grammar without any appeal to intuition.

In the first part of this paper we define our notion of a structural description of a sentence, and define the conditions under which a sentence may be said to be analyzable as such a structural description; later we discuss our implementation of these concepts, in particular the algorithm which determines in what order the various possible analyses of a sentence are produced.

Underlying concepts

We begin the discussion of structural description by explaining some underlying concepts and giving definitions of certain key terms. This is in line with one of the major goals of our project, namely uniformity, clarity, and precision of expression.

A transformational- generative grammar is a device for generating

sentences in a language. Note that this is a characterization rather than a definition; the only definition of transformational grammar given in this paper will be in terms of its three components: <u>phrase structure</u>, <u>transformations</u>, and <u>lexicon</u>.

The phrase structure component is a phrase structure grammar. One may commence with a <u>sentence symbol</u> (the letter S) and expand it by means of the grammar into a <u>base tree</u> which has the node labeled S as its top (root) node. In this tree, each <u>nonterminal node</u> (node with branches below it) corresponds to some phrase-structure rule in the sense that its label is the **lefthand** side of the rule and the labels of the nodes immediately below it are the symbols of the righthand side of the rule in the same left-to-right order. The labels of terminal nodes of the tree are terminal symbols of the grammar; the list of labels of terminal nodes, taken from left to right, is the <u>terminal string</u> of the tree. Nonterminal nodes of the tree are labeled with nonterminal symbols of the grammar. The nodes immediately beneath a given node are its <u>daughters,</u> and the given node <u>immediately dominates</u> them; a node <u>dominates</u> its daughters, the daughters of its daughters, etc. A tree node may have an associated <u>complex symbol</u> (see below); this complex symbol is not a daughter of the node, but is rather an adjunct to the label of the node. This tree is also known as the <u>constituent structure</u> of the sentence.

The transformational component contains <u>transformations</u> and a statement of the order in which these transformations are to be applied. A transformation consists primarily of a <u>structural description</u> and a <u>structural change;</u> it essentially makes the statement: "If the tree currently has this (given) structure, then change its structure in this manner."

2

The lexical component contains a list of vocabulary words, each of which has an associated _complex symbol_. A complex symbol is a collection of _feature specifications_ which describe both the inherent characteristics of the word (e.g., Noun or Verb, +HUMAN or -HUMAN (or neither), etc.) (_inherent features_), and the sort of sentence environment into which it can be inserted (_contextual features_). Lexical insertion attaches vocabulary words to the terminal nodes of a tree in positions where all of their feature specifications are met. It inserts their complex symbols into the tree at the same time.

Since both the contextual feature and the structural description of a transformation ask the question "Does the tree we are working with have this structure?", they can be treated in the same manner for most purposes. We will say in both cases that the sentence tree is _analyzable_ as the structural description if the answer to the above question is _affirmative_. The process of answering the question is _analysis_; a matching of nodes in the sentence tree with their counterparts in the structural description will be _an analysis_ of the sentence tree as the structural description.

Structural description

We have defined the formats for writing transformational grammars in our system in a modification of the Backus-Naur form (BNF) used to define computer programming languages [5]. In BNF, the definition of a structural description and a contextual feature description are:

    structural description ::= _structural analysis_ opt[,WHERE _restriction_].

    contextual feature description ::= (_structure_ opt[,WHERE _restriction_])

    structural analysis ::= list[_term_]

    term = opt[_integer_] _structure_ or opt[_integer_] _choice_ or _skip_

3

structure ::= element opt [complex symbol]

         opt[opt[¬] opt [/] (structural analysis)]

element ::= node or * or ___

choice ::= (clist[structural analysis])

skip ::= %

This definition can be thought of as a procedure for checking whether a string of characters is one of the underlined items. The ::=may be read "is a" . The operator opt[ ] means that whatever is between the brackets may or may not be present. The notation A or B is obvious. The operator list[ ] means that one or more of whatever is between the brackets should be present; for example, list[A or B] could be A or B or A A or A B or A B B A A etc. The operator clist [ ]¹ resembles list[ ], but separate occurrences of whatever is between the brackets are separated by commas; for example, clist[A or B] could be A or B or A,A or A,B or A,A,B,A,B etc. All other symbols which are not underlined mean themselves. There are four items left undefined by the above; these are restriction, which will be discussed later, complex symbol, which is defined in [4], integer, which is any positive integer, and node, which may be any string of letters and digits starting with a letter or may be a boundary symbol (#).

For example, % 1(EN,ING) 2(HAVE,BE) % . is a structural description which is the structural analysis % 1(EN,ING) 2(HAVE,BE) % followed by a . ; this structural analysis is a list of the terms % , 1(EN,ING) , 2(HAVE,BE) , % ; the first and last of these terms are skips, each of which is the symbol %, while the second and third are the choices (EN,ING)

---

¹ "clist" is pronounced see-list, and is a noun of the same type as "herd".

and (HAVE,BE) preceded by the integers 1 and 2; each choice consists
of a (followed by a clist of structural analyses EN,ING and `HAVE,BE
followed by a) ; each structural analysis here is a list of exactly
one term, which is a structure without any preceding integer; each of
these structures is an element without any of the optional items, and
each element is a node.

The above description has not in any way explained the meaning of
these items; it has simply defined how to write them., The meaning of
structural description and contextual feature description can be best
explained in terms of analyzability and analysis, since their purpose is
precisely-to test trees for analyzability and to provide analyses of
trees. Although a structural description contains a structural analysis
and a contextual feature description contains a structure, the recursive-
ness of their definitions makes them very similar. The difference stems
from the fact that when transformations are being applied the position
of the top node of the current tree is known, while during lexical
insertion only the terminal node at which insertion is being attempted
is known. For this reason, the contextual feature must specify the label
of a node somewhere above the insertion node which can serve as tree top.
In the following discussion, whenever a structural description is referred
to, we will mean either a structural description or a contextual feature
description.

Analyzability

We will define analyzability in two phases; first we will consider
a structural analysis or structure without any associated restriction,
and then we will consider how the presence of a restriction modifies the

5

definition.

    If a structural description is simply a list of elements,
analyzability is similar to Chomsky's notion of "proper analysis" [1].
A tree is analyzable as a structural description of this form if a
one-to-one match of certain tree nodes with all of the structural
description elements can be found such that:

1. Each terminal node in the tree is, or is dominated by, exactly
    one node in the match.

2. Left-to-right order of elements corresponds to left-to-right
    order of matching tree nodes.

3a. For each element which is a node, the label of the matching tree
    node is the same as the node.

3b. For each element which is a __ , the matching tree node is the
    node at which lexicon insertion is currently being attempted.
(Note that a * will thus match any one tree node, regardless of its
label.)

    A complex symbol following an element requires that a corresponding
complex symbol be attached to the matching tree node. "Corresponding"
has a different meaning for lexicon insertion than for transformations;
in the case of lexicon insertion the test is compatibility (roughly, no
conflicting feature specifications; see [4] for a precise definition),
while for transformations the test is inclusion (that is, the complex
symbol in the tree contains every feature specification of the one
in the structural description).

    A skip (the % symbol) matches not a single node, but any string
of adjacent terminal nodes. It may match a string of zero nodes, in
which case it is said to be null. The "range" of a skip is defined in

6

terms of the <u>elements</u> on either side; it is the set of tree nodes which
dominate (or equal) the nodes matching the <u>skip</u> and do not dominate
the nodes matched by these <u>elements</u>. In other words, the range of a
<u>skip</u> is precisely those tree nodes which would have to be deleted if the
<u>skip</u> were not present in order to have the analysis of the tree as the
structural description be the same as before.

The matching of a <u>choice</u> is somewhat more complex. The procedure
depends on whether the clist within the choice has only one <u>structural</u>
<u>analysis,</u> or more than one. If there is only one <u>structural analysis,</u>
it is regarded as optional; that is, the tree is analyzable either if
it is **analyszable** as a similar structural description without the
parentheses of the choice, or if it is analyzable as a similar structural
description without any of the choice being present. If there is more
than one <u>structural analysis</u> in the clist, a tree is analyzable if it
is analyzable as a similar structural description with some one of
the <u>structural analyses</u> in place of the <u>choice.</u> (Note that the only
requirement here is that at least one <u>structural analysis</u> will work;
if several different ones could be analyzably substituted; it merely
means that the tree is analyzable as this structural description in
several ways.)

A <u>structural analysis</u> within angle brackets following an element
represents a "subanalysis". The analysis of the whole tree as the
structural description is unchanged, but in order that the tree be
analyzable, there is a further requirement on analyzability of the
subtree headed by the node matched to the head <u>element</u> of the **angle-**
bracketed <u>structural analysis.</u> The exact requirement depends on the
presence of the optional modifiers ¬ and / . If only a / is

present, this sub-tree must be analyzable in the usual sense, with the minor exception that the top node of the subtree is not allowed to match any element in the structural analysis. If neither modifier is present, the subtree must be analyzable in the above sense, with the further restriction that any element in the structural analysis must match a tree node which is immediately dominated by the top node of the subtree. In the case of contextual features, this corresponds to Chomsky's notion of strict local subcategorization [2]. If a ¬ modifier is present, it means that the sub-tree must not be analyzable in the sense defined above.

Integers do not directly enter into the analysis process. They are used to permit reference to tree nodes in a restriction or a structural change. An integer preceding a structure refers to the tree node which matches the element heading that structure. An integer preceding a choice is handled exactly as if it had been written at the beginning of every structural analysis in the clist of the choice. Note that complex symbols are not numbered directly; the integer attaches to the tree node and will refer to the complex symbol associated with that node in any context which requires a complex symbol.

Restrictions

If a structural description or contextual feature description has an associated restriction, analysis proceeds exactly as above, except that the analysis of the tree must also meet the restriction in order for the tree to be analyzable. The BNF format for restriction is:

restriction ::= booleancombination[condition]

condition ::= unary condition or binary condition

unary condition ::= unary relation integer

binary condition ::= integer binary tree relation node designator or

       integer binary complex relation complex symbol designator

node designator ::= integer or node

complex symbol designator ::= complex symbol or integer

where booleancombination[condition] means any Boolean combination of

conditions which can be expressed using the connectives ⌐,&,|

(not, and, or) and parentheses.

 The conditions now in the system are:

 (unary conditions) the match must be to a terminal/tree node; or

null (in the case of an option); also a special condition useful where more

than one analysis is to be found, e.g. that the match in the current analysis

be to a different tree node than in any of the previous successful analyses.

 (binary tree conditions) equality of trees (including identity of

corresponding complex symbol); dominance without searching below a sentence

symbol; unrestricted dominance; domination by a specified node.

 (binary complex conditions) inclusion of complex symbols; nondis-

tinctness of complex symbols; and compatibility of complex symbols

(see [4]).

 The restriction on a structural description is tested whenever a

new match is found for a structure with a corresponding integer. If the

restriction fails, the structure does not match. In a conditional struc-

tural change, a restriction may be used to select one of two possible

structural changes (see below).

Analysis Algorithm

        In this section we discuss the algorithm used to find a particular
analysis of a tree as a structural description. This algorithm has nothing
to do with the question of analyzability; it merely decides the order
in which several possible analyses are taken if a sentence tree can be
analyzed in more than one way as a particular structural description.
This is particularly important if the transformation specifies that only
one analysis is to be found.

        Analysis commences with a tree marker pointing to the top node of the
tree and a structural description marker pointing to the first item in the
structural description. The procedure depends on the nature of this item.
Integers and skips are skipped but remembered. For an element (i.e., the
beginning of a structure), a match is attempted. A * will match any
tree node, a node will match a node with the same label, and a ___
will match the current lexical insertion node. If there is not a match,
the tree marker is moved to point to the leftmost daughter of the current
node, and matching is attempted again. If no match is found of a
terminal node and no skip preceded the current element, the backup
procedure is entered (see below). If a skip preceded, the tree marker
is moved to the top of the tree branch just right of the current branch,
and matching is attempted again; in this case, the backup procedure is
entered only if no match can be found for the rightmost terminal node of
the tree.

        If a match is found and a complex symbol follows the element, it
will be compared to the complex symbol attached to the matching tree
node for compatibility (in a contextual feature description) or inclusion

10

(in a structural description). If an <u>integer</u> precedes the <u>element,</u> any

conditions involving this <u>integer</u> are checked. In the case of a binary

condition, no checking is performed until both <u>integers</u> have been

matched. Failure of any of these 'tests causes analysis to proceed as

if the node had not matched the <u>element.</u>

 If the structural description marker is pointing to a <u>choice</u> instead

of an <u>element,</u> the procedure to be followed depends on whether the clist

of the <u>choice</u> contains only one <u>structural analysis</u> (an option), or

more than one (a true choice). For an option, the (of the <u>choice</u>

is ignored; options affect only the backup procedure. For a true choice,

a more complicated procedure is necessary. First, a list is made of

all <u>elements</u> which could possibly be first in the choice, in left-to-

right order. For example, if the choice were (A, (B)(C,D), % E, % (F,G)) ,

this list would be A - B - C - D - E - F - G . The element-matching

procedure is then followed as above, but at each tree node all of the

possible <u>elements</u> are tested for matches and for satisfactory <u>complex</u>

symbols and integers. ...ly , only those <u>elements</u> which are preceded

by skips are tested after a terminal-node failure. When a satisfactory

match has been found between a tree node and some <u>element,</u> analysis

proceeds along the associated <u>structural analysis</u> of the <u>choice,</u> at

the end of which it continues following the <u>choice.</u>

 If a <u>structural analysis</u> within angle brackets follows an <u>element</u>

that has been satisfactorily matched, a record is made of relevant

information about the current status of things, and analysis commences

again, using the angle-bracketed <u>structural analysis</u> and the <u>subtree</u>

headed by the node matched to the <u>element.</u> If no / preceded, 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 ¬ preceded and the **subtree** is not analyzable, or if no ¬ preceded and the **subtree** is analyzable, analysis continues following the angle-bracketed structural analysis; otherwise, analysis proceeds as if the head element had not matched its tree node.

When a structure has been successfully matched, the tree marker is moved to point to the top node of the tree branch immediately to the right of the tree node matching the head element, and analysis proceeds. The tree is analyzable as the structural description if the rightmost element not within angle brackets successfully matches a tree node on the rightmost branch of the tree, or if the rightmost such element has been successfully matched in any way and a skip follows it.

The backup procedure is entered when no tree node can be found which successfully matches the current element or choice. It moves the structural description marker backward to the left until it encounters a previously-matched element (in which case it pretends that this element did not match its tree node and starts forward again), or the (of a one-structural analysis choice (in which case it hops to the) of the choice and starts forward), or the **lefthand** end of the structural description (in which case the tree is not analyzable as the structural description).

For certain transformations, all possible analyses of the tree are required instead of just one. In this case, after each analysis is found, the backup procedure is entered to find the next one, until it

finally claims unanalyzability.

Structural Change

Because of the close relationship between the structural description
and structural change of a transformation, any comparison of our system
with others requires that the whole concept of transformation be con-
sidered at once. For that reason, we now give a description of the
structural change process. The BNF description of the form of a
structural change is:

<u>structural change</u> ::= clist[<u>change instruction</u>]

<u>change instruction</u> ::= <u>change</u> or <u>conditional change</u>

<u>conditional change</u> ::= IF ⟨ <u>restriction</u> ⟩ THEN

⟨ <u>structural change</u> ⟩ **opt**[ELSE ⟨ <u>structural change</u> ⟩]

<u>change</u> ::= **unary** <u>operator</u> <u>integer</u>

or <u>tree designator</u> <u>binary tree operator</u> <u>integer</u>

or <u>complex symbol designator</u> **binary complex operator**
        <u>integer</u>

or <u>complex symbol designator</u> <u>ternary complex operator</u>
        <u>integer</u> <u>g e r</u>

<u>tree designator</u> ::= (<u>tree</u>) or <u>integer</u> or **node**

<u>complex symbol designator</u> ::= <u>complex symbol</u> or <u>integer</u>

The operators are given by a list in the BNF form and are discussed
below.

If the current sentence tree is analyzable as a structural des-
cription and the transformation is to be performed, each <u>change</u>
<u>instruction</u> in the clist is performed in the order of occurrence
in the clist. Tree nodes have been matched to <u>integers</u> by the analysis

process; a change modifies the tree structure at the nodes matched to

its integer (s).

The change operators currently in the system are:

(unary operators) erasure of the node, all nodes dominated by it,

and all non-branching nodes dominating it,

(binary tree operators) left and right sister, daughter, and aunt

adjunction, and substitution, with or without erasure of the original

occurrence of the copied node, and optionally with special treatment

of the non-branching nodes which dominate (as in [9]).

(binary complex operators) erasure of, merging of, or erasure of

all but, specified feature specifications in the complex symbol associated

with the node,

(ternary complex operators) merging of specified features from

one node's complex symbol to another's

A conditional change causes the structural change following THEN

to be performed if the restriction is met; otherwise the structural

change following ELSE is performed, if there is one.

The change operators discussed above may be broken down into four

types: erasure, copying, moving and complex symbol manipulation.

Permutations are not given directly, since only one move can be made

at a time. The only transformation of this type that we have seen is

PASSIVE , for which we require three changes (copy, move, erase) to

interchange the subject and object.

The structural change operators include all of those of the

MITRE grammar [11] as well as those of the IBM core grammar [9].

The addition of "Chomsky-adjunction" is planned.

14

Comparisons with other notations

In a transformation, our structural analysis plays essentially the same role as the "structural description" and "structural analysis" which were first used by Chomsky. As an example, here is a transformation from Rosenbaum and Lochak [9]:

60. **WHPD2** WH pronoun Deletion 2   OB

$$
\text{X}\quad\text{WH + INDEF + (ever)}\quad \begin{bmatrix}<\text{+PRO}>\\<\text{+Sg}>\end{bmatrix}_\text{N}\quad \text{Y}
$$

$$
1 \qquad 2 \qquad\qquad 3 \qquad 4 \quad \text{-----}\rangle
$$

$$
1 \qquad 2 \qquad\qquad \emptyset \qquad 4
$$

In our system this would be written

TRANS 60 **WHPD2** "WH PRONOUN DELETION 2" OB II AACC .

SD % WH   INDEF   (EVER)   1 N |+PRO +SG| % .

SC   ERASE1 1 .

The first line gives the transformation identification and the conditions of applicability. In this case the transformation number and name are followed by a comment and by parameters specifying that the transformation is obligatory (OB) , is in group II, and that it is to be applied by first finding all possible analyses and then performing the changes for each of them (AACC) . A full discussion of the possible parameters is given in [6]. The second line is our structural description. As can be seen, the details of the representation are different, the major features are the same. We chose the % symbol rather than X, Y, Z to represent variables because these letters are possible labels for nodes. This decision reinforces the idea that a variable need not be a constituent. The standard use of parentheses for options is carried over into our no-tation; in addition, we reflect the use of curly brackets for a choice

by allowing a **clist** of structural analyses within parentheses. Our notation for complex symbols resembles standard notation except for the use of vertical bars in place of square brackets; see [4] for a complete discussion of complex symbols in the system. The most significant change is in our use of numbers, since we allow only constituents to be numbered, and do not require numbering of items which are not referred to in either the structural change or the restriction. This is a result of our treatment of transformations as changes of position of single constituents rather than rearrangements of the whole tree. In this we follow the approach taken in the **MITRE** grammars [11]; we have extended the approach to complex symbol operations.

Gross [7] and Londe and Schoene [8] have also developed notations for transformations, in both cases for use with grammar testers. Both notations differ from ours in form and have less power in the structural description. For example, Gross does not include complex symbols; neither allows any equivalent of ; Londe and Schoene require that immediate dominance be expressed as a restriction. However, both systems contain more powerful notations than ours for structural change.

Future directions

The analysis algorithm was designed to correspond to the linguistic theory based on <u>Aspects</u> [2]. Since that time there have been radical changes in the theory; the change of particular importance for analysis is the strong notion of general constraints on transformations, following from the work of Ross [10]. Thus, if the system is to be extended and kept current with the theory, the first changes will need to be in **devising** notations and algorithms for the implementation of

general conditions on the applicability of transformations.

REFERENCES

[1] Chomsky, N. and Miller, G. A. Introduction to the formal
       analysis of natural languages. in Luce, R. D., Bush, R. R.,
       and Galanter, E. (Eds.), Handbook of Mathematical Psychology,
       Volume II Wiley (1963).

[2] Chomsky, N.   Aspects of the Theory of Syntax. M.I.T. Press,
       Cambridge, Massachusetts (1965).

[3] Friedman, J. A computer system for transformational grammar.
       CS-84, AF-21, Computer Science Department, Stanford University
       (January, 1968).

[4] Friedman, J., and Bredt, T. H. Lexical insertion in transfor-
       mational grammar. CS-103, AF-25, Computer Science Department,
       Stanford University (June, 1968).

[5] Friedman, J., and Doran, R. W. A formal syntax for transfor-
       mational grammar. CS-95, AF-24, Computer Science Department,
       Stanford University (March, 1968).

[6] Friedman, J., and Pollack, B. W. A control language for
       transformational grammar. Computer Science Department, Stanford
       University (September, 1968).

[7] Gross, L. N. A computer program for testing grammars on-line.
       mimeographed (1968).

[8] Londe, D. L., and Schoene, W. J. TGT: transformational grammar
       tester. Systems Development Corporation (1967).

[9] Rosenbaum, R., and Lochak, K. The IBM core grammar of English.
       In Lieberman, D. (Ed.) Specification and utilization of
       a transformational grammar. AFCRL-66-270 (1966).

[10] Ross, J. R. Constraints on variables in syntax. M.I.T. Thesis
       (1967).

[11] Zwicky, A. M., Friedman, J., Hall, B. C., and Walker, D. E.
       The MITRE syntactic analysis procedure for transformational
       grammars. Fall Joint Computer Conference 27 (1965), 317-326.

18