DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Due to the poor quality of the original document, there is some spotting or background shading in this document.
A RECOGNITION PROCEDURE FOR TRANSFORMATIONAL GRAMMARS

by

STANLEY ROY PETRICK

B.S., Iowa State University
(1953)

S.M., Massachusetts Institute of Technology
(1955)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

June, 1965

Signature of Author...

Department of Modern Languages, May 14, 1965

Certified by...........

Thesis Supervisor

Accepted by .......................... Chairman, Departmental Committee on Graduate Students
A RECOGNITION PROCEDURE
FOR TRANSFORMATIONAL GRAMMARS

by

STANLEY ROY PETRICK

Submitted to the Department of Modern Languages on May 14, 1965, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

A class of transformational grammars is defined which is appropriate for the description of natural languages. The notion of a structural description for such grammars is made precise, and the problem of finding the set of structural descriptions that a given grammar assigns to a sentence (i.e. developing a recognition procedure) is proposed. Prior work on this problem is critically reviewed.

Several formal conditions on these grammars are presented which are required not only to make possible the solutions we give, but in some instances to make possible a recognition procedure by any means.

Psycholinguistic implications of recognition procedures in general and our in particular as models of perception are discussed.

Several specific algorithms for finding structural descriptions of sentences with respect to an arbitrary transformational grammar of the class considered are given, and each algorithm is proved to constitute a recognition procedure.

A LISP program implementation of one algorithm is documented and discussed. Results obtained through its use are presented, and arguments are given to support its utility as a tool in linguistic research.

Thesis Supervisor: George Hubert Matthews
Title: Associate Professor of Modern Languages
Acknowledgements

The author wishes to express his gratitude to G. H. Matthews who, in supervising this thesis, not only gave generously of his time in discussing the work and reading the manuscript, but also made many important specific contributions. T. Griffiths is to be acknowledged for giving the algorithm of page 70 and for valuable discussion on several points. J. Keyser is also to be thanked for many enlightening discussions and for the interest he showed in using the recognition program to investigate his own grammars. S. Schane contributed several grammars in the early stages of computer recognition and must also be thanked for helping to make the recognition program operational. T. Evans made the LISP implementation possible by providing a very convenient on-line LISP system and further contributed the entire LISP context-free recognition program. R. Vicksell wrote the assembly language IBM 7090 program for context-free recognition, integrated this subroutine into the LISP transformational recognition program, and contributed several LISP functions. Miscellaneous other LISP functions were written by T. Evans, S. Schane, and J. Keyser.

Conversations on various aspects of this thesis with N. Chomsky, B. Fraser, J. Friedman, B. Hall, M. Kay, S. Kuno, D. Walker, A. Zwicky, and many others were much appreciated.

Finally, my wife Mary is to be thanked for making this thesis possible by her encouragement, patience, and confidence.
# TABLE OF CONTENTS

I  INTRODUCTION  
1.1 Problem Statement; Reasons For Separating Grammar And Recognizer  
1.2 Reasons For Considering The Transformational Recognition Problem  
1.3 Previous Consideration Of The Transformational Recognition Problem  

II  TRANSFORMATIONAL GRAMMAR CONSIDERED  

III  RECOGNITION PROCEDURES  
3.1 Overview  
3.2 Definition Of Inverse Transformations  
3.3 The Simplified Procedure  
3.4 The Improved Procedure  
3.5 A Modification Of The Improved Procedure  
3.6 Determination Of Auxiliary Rules  
3.7 Extensions To Modified Grammars  
3.8 Reasons For Synthesis Phase  

IV  THE RECOGNITION PROGRAM  
4.1 Computer Implementation  
4.2 Usage Of Recognition Program  
4.3 Perceptual Model Considerations  

V  SUMMARY  

APPENDIX A : METHODS OF FINDING PROPER ANALYSES  
APPENDIX B : TRANSFORMATIONAL GRAMMARS RECOGNIZED  
APPENDIX C : RECOGNITION EXAMPLES  
APPENDIX D : LISP PROGRAM DOCUMENTATION
CHAPTER I: INTRODUCTION

1.1 PROBLEM STATEMENT; REASONS FOR SEPARATING GRAMMAR AND RECOGNIZER

The use of formal devices (grammars) to precisely specify the set of well-formed sentences of a language and to assign structural descriptions to those sentences is a rather recent and important development in linguistics. It is only by precisely defining a proposed linguistic theory that its adequacy can be investigated thoroughly, and a minimal requirement that it must meet is that the utterances of every natural language be characterizable in terms of it. The use of a generative grammar makes possible the meaningful consideration, with respect to a particular description of any language, of such questions as: (1) Are there grammatical sentences we have not enumerated? (2) Are there ungrammatical sentences we have falsely identified as grammatical? (3) Do the structural descriptions that are assigned to a sentence correspond to distinct semantic interpretations of that sentence? and (4) Can these structural descriptions be usefully employed in some applied area, say by mapping them into appropriate computer code or into an equivalent sentence in either the same or another natural language? Making explicit this concept of a generative grammar is, in my opinion, Chomsky's most important contribution to linguistics, far outweighing the results of his investigations of specific grammatical models. A theory of grammar that is expressed in terms of a class of generative grammars can be meticulously investigated and, if found lacking on one or more counts, can be either modified or discarded.
Among the syntactic models that have been proposed and sufficiently formalized to permit their serious consideration as natural language grammars are the finite state grammar, the context-free grammar, and the context-sensitive grammar.\(^2\) We will not discuss their inadequacies as descriptors of a natural language which led Chomsky to propose another model, the transformational grammar; the linguistic motivation for considering transformational grammars has already been well documented.\(^3\) Instead, we assume the utility of the transformational grammar and seek to obtain a recognition procedure for it, i.e., to obtain a procedure which, given a transformational grammar \(G\) and a string \(S\), decides whether \(S\) belongs to the language specified by \(G\) and, if so, determines the structural descriptions that \(G\) assigns to \(S\).

We are, of course, assuming that a class of grammars (in particular transformational grammars) and its associated recognition procedure can be separately considered. Furthermore, we assume that some advantage is to be derived from a separation of grammar and recognition procedure. We would not feel compelled to comment further on these assumptions except for the fact that certain recent papers have argued against the separation of grammar and recognition procedure. Thorne [7] suggests in a recent review that "perhaps" there may be some theoretical reason why grammar and recognition procedure cannot be separately considered. In reviewing a paper of Greibach he quotes three theorems related to standard form context-free grammars\(^4\) and concludes with "The author points out that a directed production analyzer is an inverse of infinitely many phrase structure generators: a fact which, perhaps, should make one think again about the claim made for certain automatic parsing systems.
that the grammar has been made separate from the program". I am at a loss to see how this tentative conclusion follows from the quoted premises or any other premises, for it most certainly is true that there is no theoretical reason why the grammar and its associated recognition procedure cannot be separate for standard form context-free grammars, arbitrary context-free grammars, context-sensitive grammars, or transformational grammars. Recognition procedures have been given for each of these classes of grammars.\footnote{5} 

A recent paper concerned with this question of separating grammar and recognition procedure was written by Garvin [16]. He states that, "The purpose of this paper is to examine the validity of the frequently repeated contention that the separation of algorithm and grammar is particularly desirable in automatic parsing programs." Garvin assumes a context-sensitive grammar as the appropriate device for linguistic description,\footnote{6} but in place of the usual contextual restrictions on environment, which he claims are not adequate for natural language specification, he seems to imply that contextual restrictions are unstateable by any means short of an arbitrary computer program. We note, first, that in restricting his argument by the assumption of this type of a context-sensitive grammar the argument is inapplicable to transformational grammars. Perhaps by implication Garvin would claim generality for his conclusions by excluding transformational grammars as acceptable devices for linguistic description. Not having made this exclusion, however, Garvin's argument is restricted to context-
sensitive grammars in the usual sense. We note, second, that Garvin
gives no concrete examples of environments required by a natural
language grammar that are too complex to permit description save
by an arbitrary computer program. Obviously we cannot prove there
exist no such environments, but the burden is Garvin's to establish
this by supplying concretes examples. Finally, we note that even
with a context-sensitive grammar of arbitrary complex environmental
restrictions it is possible to make use of a grammar-independent
context-free recognition procedure followed by a grammar-dependent
checking of structures to see that they satisfy arbitrary environ-
mental restrictions. To do this we merely discard all environmental
restrictions from the given context-sensitive rules to obtain a
corresponding context-free grammar. If recognition is then performed
on a sentence with respect to this context-free grammar, the set of
structural descriptions thus obtained includes all the structural
descriptions assigned to that sentence by the original context-
sensitive grammar. Hence, to determine all of those context-sensi-
tive structural descriptions it is only necessary to select all
associated context-free structural descriptions that satisfy the
context-sensitive restrictions. The question of whether this pro-
cedure is more efficient than some alternative procedure that makes
direct use of the context-sensitive restrictions is unknown and
probably depends upon the specific context-sensitive grammar in question.
We merely mention the possibility of this procedure for recognition to
demonstrate that even with a context-sensitive grammar of the type
advocated by Garvin, use can be made of a recognition procedure that varies from grammar to grammar only in the application of different environmental restrictions in validating or discarding potential structural descriptions.

Garvin's other arguments add nothing to his case and require no comment. The one substantial argument he might have given against separation of grammar and recognition procedure is the following: A recognition procedure that is valid for any member of a class of grammars is in general less efficient than one tailored to a particular grammar. If we wish to say something that is linguistically significant about a recognition procedure, it must be a statement about a procedure that is applicable to a class of grammars by means of which any natural language can be described. If, however, we are interested in the efficient practical utilization of a particular grammar, then special computations that are appropriate only to the recognition of that grammar are in order. Under these circumstances the division between a grammar and its associated recognition procedure is blurred.

This discussion suggests that a choice must be made between (1) staying true to providing a recognition procedure valid for a class of grammars and (2) allowing use of the properties of a specific grammar in order to obtain a more efficient recognition procedure. We consciously chose the first alternative, realizing that we probably sacrificed efficiency for generality.

By separating the grammar and its recognition procedure not only do we avoid separately programming a host of recognition procedures and we make possible modifications by just changing grammatical rules,
but, more important, we separately make possible the determination
of the adequacy of a grammar as a descriptor of a natural language
and the determination of the adequacy of its recognizer. If grammar
and recognizer are lumped together, these two aspects of linguistic
adequacy may (indeed, must) still be considered, but the investiga-
tion is enormously complicated.

1.2 REASONS FOR CONSIDERING THE TRANSFORMATIONAL RECOGNITION PROBLEM

Assuming then the utility of separating grammar and recognition
procedure, and further assuming the particular utility of the trans-
formational grammar as a means of linguistic description, we can still
ask what is to be obtained from considering the recognition problem
for transformational grammars. First, consideration of recognition
procedures contributes to our knowledge of the precise nature of
transformational rules. The more assumptions we can build in, and
consequently the tighter we can make our model without impairing its
facility to describe language, the better that model is, and the
closer we are to saying something of interest about a discovery pro-
cedure. A precise characterization of transformations is necessary
with respect to both their domain and range, and conclusions arising
from a consideration of the recognition problem include certain re-
quirements that must be met by the rules of a transformational grammar.

Another theoretical reason for studying the recognition problem
is to contribute to a perceptual theory of how internalized rules
can be used in a model of understanding for the hearer. A generative
grammar as we have defined it merely enumerates a set of sentences
and says nothing about how sentences may be produced or how they may be recognized. In commenting on the implications of this to models of perception [2], Chomsky has stated, "A grammar, in the sense in which I have been using the term ... is thus neutral as between speaker and hearer, in the sense that it says nothing specific about how either actually operates." A recognition procedure can always be considered with respect to its adequacy as a component of a perceptual model. It is not the case, however, that a given recognition procedure is necessarily a good or even a reasonable perceptual model component. Nevertheless, one of the valid reasons for considering the recognition problem for transformational grammars is that the problem of perception can thus be approached intelligently. In terms of such an underlying model which makes specific claims about perception, meaningful experiments are suggested on both human subjects and mechanizations of this abstract model.

Turning to more practical considerations, one reason for obtaining the structural descriptions of a sentence is to make use of them in some application. In the area of programming language compiler implementation, work has been done on mapping structural descriptions assigned by context-free grammars (that define programming languages) into appropriate computer code. [17, 18, 19, 20, 21, 22, 23]. The procedure used in most of these investigations involves the use of a translation rule corresponding to each given context-free rule. Each translation rule produces a block of computer code or a call on some subroutine, and there
is one argument of this code or subroutine corresponding to every constituent of the right hand member of the associated context-free rule. The application of translation rules begins at the bottom of a structural description tree and commences to successively higher nodes, ultimately yielding a string of symbols that constitutes the computer code translation of the given input program. It is natural to ask whether structural descriptions assigned by a transformational grammar can also be mapped by some procedure into computer code, making possible the use of natural language (to the extent it is described by the grammar in question) for programming purposes. The related problem of translating not directly to computer code but rather to some higher level programming language or to equivalent natural language sentences in the same or another language lies at the heart of such applications as machine translation, information retrieval, and automatic abstracting.

A final reason for considering the recognition problem for transformational grammars is related to the value of a computer implementation of a transformational recognizer to a linguist. A transformational grammar is an intricate and complex device whose correct functioning requires that the output of a transformation be properly analyzable so as to qualify for the application of certain subsequent transformations (which may include the original transformation). In order to insure that an entire grammar generates sentences as intended, the use of a mechanized
recognition procedure is extremely valuable. Checking a sequence of trees that constitutes a transformational derivation of a sentence is, of course, possible by hand, but the amount of labor required reduces this activity to less than a healthy level. In addition to checking derivations already thought out, the linguist also frequently suspects that a given sentence is defined by his grammar, and he is often interested in examining the structural descriptions of such sentences to see if they reflect relationships he believes are important. A computer implemented recognition procedure is a valuable tool in making available required structural descriptions. Finally, one measure of a grammar's adequacy is the extent to which it assigns structural descriptions that correspond to distinct interpretations of a sentence. One of the primary claims made by proponents of the transformational grammar is that this grammar is more successful in this respect than other types of grammars that have been proposed. ⁸ A fair question to raise, however, is whether large transformational grammars actually avoid unwanted ambiguity or whether, instead, such ambiguity exists but has so far remained undetected. The only way to insure the adequacy of transformational grammars with respect to this question of ambiguity is to make use of a mechanical recognition procedure in determining and examining sentences specified by a transformational grammar.
1.3 PREVIOUS CONSIDERATION OF THE TRANSFORMATIONAL RECOGNITION PROBLEM

In this section we review briefly the work which has been done and the observations that have been made on the recognition problem for transformational grammars. We first note that relatively little attention has been paid to this problem relative to that paid to the corresponding problem for context-free grammars. This is partially due to the fact that context-free grammars have proved, for the most part, adequate for specifying and assigning structural descriptions to programming languages. Because context-free structural descriptions are useful for relating a programming language input string to a corresponding target language string (usually assembly language code), considerable effort has been expended to find efficient context-free recognition procedures. For a survey and comparison of various context-free recognition procedures see Griffiths and Petrick [13].

Even those linguists who have accepted a transformational framework for the description of language have, for the most part, not been concerned with the recognition problem, being principally occupied with such fundamental problems as the writing of grammars for specific languages. Consequently, not very much has been done on the problem of transformational grammar recognition.

The first transformational grammar effort we examine is that of Joshi [26,27]. Joshi follows Harris in transforming morpheme class sequences into other such sequences rather than basing
transformational domains on conditions of tree structure analyzability and in transforming one tree structure into another. He permits the operations of permutations, deletions, repetitions of the morpheme class marks, and insertions of constants (terminal symbols).

Two principal assumptions are listed by Joshi. The first is that every sentence composed of a set of kernel sentences \( K = \{ K_i \mid 1 \leq i \leq n \} \) contains (for \( i = 1, \ldots, n \)) either the verb \( V_i \) of \( K_i \) or else a set of transformational constants that are characteristic of the transformations that operated on the \( K_i \). (I.e. the verb of every kernel sentence that underlies a given sentence \( S \) is either found explicitly in \( S \) or else its presence is implicitly indicated by morphemes that were transformationally introduced.) The second assumption is that in every sentence produced by a unary transformation acting on a kernel \( K_i \) there appears a unary constant together with possible other constants such as \textit{en}, \textit{by}, and \textit{to} which are sufficient to identify the unary transformation.

The limitation of possible rules by means of assumptions such as these is to be commended primarily because it contributes to the precise characterization of language but also because it makes possible a more efficient recognition procedure. In this paper we are interested in grammars the applicability of whose transforms is based on conditions of phrase structure analyzability (P-marker analyzability) rather than on analyzability as a sequence of morpheme classes. We are interested in transformations that map a complete tree into another complete tree
rather than mapping one sequence of morpheme classes into another.\(^9\)

Hence, we will not dwell on Joshi's procedure except to characterize it and observe in what respects it is relevant to our transformational grammar recognition problem.

It appears to this writer that transformations as defined by Joshi can be characterized as semi-Thue productions. All rules are of the form \(A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m\) where the \(A\)'s and \(B\)'s are morpheme class symbols and the only condition on applicability of a rule of the above type is that \(A_1 A_2 \ldots A_n\) is produced by the grammar.\(^10\) Hence, Joshi's recognition procedure, even if completed and working, would be of value to a transformational recognizer of the more general type we seek only insofar as a need existed for a recognizer of semi-Thue systems satisfying Joshi's assumptions. We will later argue that even with a context-sensitive base component, a transformational recognizer can effectively make use of context-free rules obtained from the context-sensitive rules.

Turning to the recognition of transformational grammars consisting of a base component that defines a set of trees and a transformational component that maps those trees into other trees, we find the work of Matthews \([8,9,10]\), Walker and Bartlett \([28]\), Fraser \([29]\), Herzberger \([30]\), and Zwicky, et al \([31]\).

Matthews was the first to write on the transformational recognition problem. Briefly, he showed how for every structural description of each sentence of a language defined by some transformational grammar, it is possible to set up a unique integer
(specifier) that denotes a unique transformational derivation of the sentence in question. By obtaining a bound $f(n)$ on the specifier of a sentence consisting of $n$ terminal symbols, he demonstrated that it is in principle possible to obtain all structural descriptions of a given sentence by applying the rules through a synthesis procedure no more than a bounded number of ways. He observed that actually following such a procedure would be prohibitively time consuming, and he proposed to make the procedure feasible by performing certain preliminary analyses on the sentence so as to preclude large numbers of potential specifiers and to reduce the amount of exhaustive sentence synthesis required to a reasonable magnitude. In his last paper he gave details of how a system of specifiers could be set up for a so-called deep structure transformational grammar, and he began to consider details of a preliminary analysis procedure. As we point out subsequently, we develop procedures for analysis in this thesis that find all structural descriptions of a given sentence along with, perhaps, one or more incorrect structural descriptions. Hence, our analysis procedures must be used as preliminary analysis procedures in conjunction with an analysis-by-synthesis algorithm as suggested by Matthews. Our procedures differ from his in that we do not eliminate blocks of sequential specifiers, testing other blocks by means of a synthesis component. Instead, we generate a set of potential specifiers that must include every valid specifier, and we test each of these with a subsequent synthesis.
Walker and Bartlett [28] also advocate the approach proposed by Matthews. They give a flow chart of the recognition procedure, but their paper is primarily concerned with other matters, and it does not define more precisely the details of how an analysis-by-synthesis procedure might be implemented. In particular, nothing new is included about how to relate specifiers to rule sequences or about how one might make use of a preliminary analysis to reduce the amount of required sentence synthesis.

Fraser [29] suggests that a possible transformational recognition procedure could consist of the following six steps:

1. Dictionary lookup,
2. Reduction,
3. Identification of transformations,
4. Reversal of transformations,
5. Phrase structure determination, and

The nature of most of these steps is evident from their titles. (1), for example, involves identification of all phrase structure or transformational rules which write a given terminal symbol, and (2) refers merely to the elimination of certain rules identified in (1) on the basis of such methods as a pair test whereby a listing of constituents that can occur next to specific other constituents is used.

Fraser implies that the first five steps can be performed sequentially and that the last step is not actually required but is only used as a check on the correctness of structural descriptions produced by steps one through five. We see that a major difference exists in this respect between the procedure advocated by Fraser and the analysis-by-synthesis procedure of Matthews, which involves sentence synthesis in an essential way.
The problem with Fraser's suggestion is that his steps cannot be carried out in the indicated order. Determination of the final derived constituent structure tree or at least a partial determination of this tree is required to establish the necessary structure for transformation reversals. 11 Hence, Fraser's suggestion, at least if it is interpreted strictly, must be discarded, and it sheds no light on whether or not a pure analysis recognition procedure for transformational grammars can be used in place of analysis-by-synthesis. Even if such a pure analysis procedure is possible, though, it is not necessarily more efficient than an analysis-by-synthesis scheme. The important question is not whether synthesis is essential but rather how much synthesis should be allowed in order to obtain an efficient recognition procedure.

Herzberger[30] realized the importance of this question, and he referred to erroneous structure produced by analysis and discarded by subsequent synthesis as the slack of the system. Although he did not definitively answer the question of how much slack to utilize, he suggested an approach which, insofar as it is specified, is quite close to the method we present in Chapter III. In particular, he advocated that the phrase structure rules be imposed on the structure existing at each step in order to determine the applicability of inverse transformations whose use further modifies and adds to the known structure. Although our early work was done in ignorance of Herzberger's already written paper, it is fair to describe our recognition procedure as
constituting in its essential aspects a realization of the approach outlined by him. It must be noted, however, that Herzberger stopped short of treating such important questions as precisely how can a phrase structure component be imposed on the existing structure to determine the applicability of inverse transformations. It must also be noted that our procedure differs explicitly in certain respects, due partially to the fact that our model of a transformational grammar is somewhat different from Herzberger's. The relegation of recursive devices to the phrase structure and the use of a cyclic application of transformations are relatively recent developments in transformational theory that are of great importance to the development of an efficient recognition procedure.

A distinctly different approach has been proposed\textsuperscript{12} by the Language Processing Techniques Sub-department of the MITRE Corporation.\textsuperscript{[31]} Briefly, they propose to make use of a so-called context-free surface grammar that assigns to each sentence all of the final derived constituent structure trees (surface structure trees) that are given by a transformational grammar. The surface grammar may also assign unwanted, erroneous derived constituent structure to sentences specified by the transformational grammar or even produce sentences not defined by that grammar. In the MITRE approach inverse transformations which map trees into trees are applied in reverse generative order to the set of surface trees given by the surface grammar, and the resulting trees are checked to insure that they satisfy the given base component phrase structure rules. Finally, potential structural descriptions
thus obtained are submitted to a synthesis component for verification or rejection.

This approach, as described in Reference 31, does not constitute a recognition procedure for any class of transformational grammars or even for the particular transformational grammar considered, and no claim is made by the authors of the MITRE report that it is a recognition procedure. There are at least three reasons why no such claim can be made: (1) no algorithm was given for deriving the required surface grammar from a given transformational grammar, (2) no algorithm was given for deriving the inverse transformations corresponding to a given transformational grammar, and (3) rather than both performing and not performing applicable inverse transformations and separately following each continuation, the MITRE program described obligatorily performs all applicable inverse transformations. The first of these objections can be overcome by following the auxiliary rule-producing algorithm we discuss in Section 3.6, if the transformational grammar in question meets the conditions we prescribe and if, in addition, sentences of \( n \) or less symbols are to be recognized. This number \( n \) can be as large as desired, but it can well be the case that the number of rules in the surface grammar is a strictly increasing function of \( n \) (not just monotone increasing). Hence without such a restriction on \( n \) it may be the case that there exists no finite surface grammar. We give such an example in Chapter II.

The third objection given above can be removed merely by performing and not performing each applicable inverse transformation.
and separately considering each continuation. We are obliged to do this in the recognition procedure of this thesis and, unfortunately, it leads to greatly increased computational time requirements.

I have not considered the problem of mechanically deriving inverse transformations of the type required by the MITRE approach, so I cannot comment on the magnitude of this obstacle to achieving a recognition procedure. In contrast to the MITRE approach, at least in its present state, the procedures given in Chapter III do constitute recognition procedures.
CHAPTER II: TRANSFORMATIONAL GRAMMAR CONSIDERED

We present in this chapter a precise formalization of a transformational grammar with which we will subsequently be concerned. This formalization differs in certain details from various others that are currently in use, but we assert that except as noted, it reasonably reflects current thinking on transformational linguistic theory. We make use of computer format notation to a large extent in order to simplify the subsequent discussion of computer implementation.

We confine our attention to the syntactic component of a transformational grammar. This consists of two subcomponents, a base component that defines a set of base structures and a transformational component that maps those base structures into so-called surface structures. An operation called debracketization produces the sentence corresponding to a given surface structure, and the set of sentences that can thus be generated constitutes the language specified by the syntactic component of a given transformational grammar.

We take as our base component a context-free grammar. We will subsequently discuss modifications of our basic procedure that permit taking as the base component a context-sensitive grammar or a context-free or context-sensitive grammar with ordered rules. It is to be noted that we do not include in the base component any use of complex features to restrict lexical rewriting rules, and
thus our "dictionary" consists of morphemes introduced either
by means of phrase structure rules or by means of transformations.

We formalize the notion of a context-free (CF) grammar as follows:
A CF grammar is a quadruple \( G = (N, T, S, P) \) where:
(a) \( N \) is a finite set called the non-terminal alphabet.
(b) \( T \) is a finite set disjoint from \( N \) called the terminal alphabet.
(c) \( S \) is a member of \( N \) called the distinguished symbol.
(d) \( P \) is a finite set of productions or rewriting rules of the form:
\[ A \rightarrow x_1 \] where \( A \in N \) and \( x_1 \) is a string in \( N \cup T \).

We will reserve our use of \( A \) and \( B \) to denote elements of \( N \) and will
use other initial capital letters (\( C, D, \ldots \)) to represent
single elements of \( N \cup T \). Strings of characters in \( N \cup T \) will be
represented by subscripted later lower case letters \( (x_i, y_i, z_i) \).

A sequence of strings \( y_i \) \( (i = 1, \ldots, n) \) is called an A-derivation
with respect to a CF grammar \( G \) if \( y_1 = A \) and if for each \( i \) \( (1 \leq i \leq n) \)
there are strings \( z_1 \) and \( z_2 \) (possible null) such that \( y_i = z_1 B z_2 \), \( y_{i+1} = z_1 x_1 z_2 \), and \( B \rightarrow x_1 \) is a rule of \( G \). Any string which is a member
of an S-derivation with respect to \( G \) and which consists entirely
of symbols belonging to \( T \) we will define to be a sentence specified
by \( G \). The set of such sentences constitutes the language specified
by \( G \).

We will define the structural description of a sentence with
respect to a given CF grammar \( G \) by giving a grammar \( G' = (N, T', S, P') \)
where \( T' = T \cup \{ \} \) \( \cup \{ [A] \mid A \in N \} \)
and for every rule \( A \rightarrow x_1 \) of \( P \) there corresponds a rule \( A \rightarrow [x_1] \)
of \( P' \). By debracketization of a string we mean deletion of all
symbols from \( \{ \} \cup \{ [ | A \in \mathbb{N} \} \). The sentences of \( G' \) will be said to be structural descriptions of the debracketed strings, which are sentences of \( G \). Clearly, these structural descriptions can be represented by labelled tree diagrams. The debracketization of a fully developed tree is, of course, the string of terminal symbols of that tree. We will represent the debracketization of a tree \( R \) by \( d(R) \).

We place the following restrictions on a CF grammar that is to serve as the base component of a transformational grammar:

(1) The only rule which expands the distinguished symbol \( S \) is \( S \to \# S' \# \) where \( \# \in \mathcal{T} \) is the so-called sentence boundary symbol.

(2) There is another symbol \( \text{COMP} \in \mathbb{N} \), distinct from \( S \), which is expanded only by the rule \( \text{COMP} \to \# S' \# \).

(3) The symbol \( \# \) appears only in the two rules enumerated in (1) and (2).

(4) Recursion is permitted only through the use of the symbol \( \text{COMP} \). That is, the members of \( \mathbb{N}, (S, S', \ldots, \text{COMP}) \) are partially ordered with respect to the ordering relation defined as follows: For every rule \( A \to x_1 B x_2 \) either \( A \) precedes \( B \) or else \( A \) is the symbol \( \text{COMP} \).

The structural descriptions generated by such a CF grammar will constitute the base structures of the transformational grammar model we consider. The transformational component consists of an ordered set of transformations, each of which maps a tree structure into another tree structure. These transformations are sequentially
applied in order and in a cyclic fashion until they are no longer applicable, the output of a transformation at each stage serving as the input to the next transformation. The first transformation is applied initially to a base structure given by the base component and the ultimate structure produced is called the surface structure or derived constituent structure.

The domain of a transformation is indicated by means of a structural index, which is an ordered set of symbols from \( N \cup T \cup M \) where \( M \) is a finite set of symbols disjoint from \( N \cup T \), denoting a collection of morphemes that are introduced by some transformation. A proper analysis of a tree \( R \) is an ordered set of subtrees \((r_1, r_2, \ldots, r_n)\) of \( R \) such that \( d(r_1) d(r_2) \ldots d(r_n) = d(R) \).

A proper analysis of a tree \( R \) satisfies a structural index \( I = (i_1, i_2, \ldots, i_n) \) if \( i_j \) is the name of the top-most node on the tree \( r_j \) for \( j = 1, \ldots, n \). A transformation is said to be applicable to a tree if the tree has a proper analysis which satisfies the structural index of that transformation.

We make use of another convention in defining the conditions under which a transformation is applicable to a given tree. If we include among the elements of a structural index the symbol \( X \), not included in \( N \cup T \cup M \), then we indicate an arbitrary sequence (including the null sequence) of symbols from \( N \cup T \cup M \cup \{\#\} \).

That is, use of \( X \) gives a rule schema index \( I = (i_1, i_2, \ldots, i_p, X, i_{p+2}, \ldots, i_n) \) which is satisfied by a proper analysis \((r_1, r_2, \ldots, r_m)\) if \( i_j \) is the name of the top-most node on the tree \( r_j \) for \( j = 1, \ldots, p \) and if \( i_{p+k} \) is the top-most node on the tree \( r_{m-n+p+k} \) for \( k = 2, 3, \ldots, n-p \). Use of more than
one X in a structural index is also permitted, and its meaning is the obvious extension of the above definition. Similarly to our use of symbol X we will use symbol Y, also not included in N U T U M, to indicate an arbitrary sequence from N U T U M.

We distinguish two types of transformations: singulary and binary, and the former are subdivided into sub-types, optional, obligatory, and repeated. The application of singulary transformations precedes that of binary transformations. Binary transformations are not ordered relative to each other but singulary transformations are, and may consist of any ordered mixture of optional, obligatory, and repeated transformations. If at any point in the derivation of a sentence by means of a transformational grammar an obligatory transformation is applicable to the tree, then the transformation must be performed. Optional transformations, as the name implies, may be performed if they are applicable but need not be. Repeated transformations, unlike the other two singulary types, and binary transformations may be performed on a tree more than a single time if applicable. Repeated transformations and binary transformations will be regarded as optional with respect to whether or not they must be performed if applicable. We remark that another type of repeated transformation to be found in the literature, one that we will not consider, determines multiple proper analyses and then performs a transformation several times on separate parts of a tree but does not reanalyze the tree between successive applications of the transformation.
In addition to a structural index, each transformation has a structural change that specifies the changes to be performed on the tree being transformed. We will make use of two elementary transformations, sister-adjunction and substitution (including deletion). Sister adjunction is an operation that adjoins two or more trees in an indicated order and substitution is an operation which substitutes an adjoined set of trees or a single tree or a null (deleted) tree for a given subtree $R'$ of some tree $R$. If an adjoined set of trees or a single tree is substituted for $R'$ then each substituted tree is connected to that node of $R$ which immediately dominated $R'$. If, however, a null tree is substituted for $R'$ then not only is $R'$ deleted but also all higher structure emanating from $R'$ up to the first node that dominates two or more nodes.

The structural change is an n-tuple where $n$ is the number of terms of the corresponding structural index. Each of its $n$ terms consists of either (1) the integer zero, denoting substitution of the null tree, (2) an integer $j$ between 1 and $n$, denoting substitution of the $j$th tree of the proper analysis which satisfies the structural index, (3) a member of $T \cup M$, or (4) the adjunction of two or more of the members of (2) and (3). The $i$th term of the structural change denotes the structure which is to be substituted for the $i$th tree of the proper analysis in question.

A few additional remarks on the use of rule schema elements $X$ and $Y$ must be made. The reason for using two separate symbols that differ only with respect to whether they can contain the sentence boundary symbol # is merely to allow us the formalism we require to
describe the cyclic fashion in which transformations are applied. This will be made clear subsequently. If the jth term of the structural index is an X or a Y, then the jth term of the structural change must be simply the number J, denoting that the structure designated by X or Y is not to be modified. We could relax this restriction somewhat and define conventions that permit trees to be adjoined to an X or Y, but we will not do this because it enormously complicates Section 3.6.

To give concrete examples of our definitions, we will at this point introduce the notation to be used hereafter. Consider the transformation

\[(\text{OBLIG}, ((\text{PRE})) \text{ NP AUX V NP BY PASS } ((\text{ADV}))),\]

\[(1 5 (3 \text{ BE EN} 4 0 6 2 8)),\]

\[(), \text{ PASSIVE})\]

We will represent all transformations by ordered 5-tuples, the coordinates being separated by commas (as above) or by spaces. The first coordinate, OBLIG in our example, denotes that this transformation is an obligatory singulary transformation. OPT is used to denote an optional singulary transformation, and REPEATED denotes the remaining type of singulary transformation.

The second coordinate of a transformation, \(((\text{PRE})) \text{ NP AUX V NP BY PASS } ((\text{ADV})))\) in our example, gives the structural index. Terms consisting of a list of symbols enclosed in double parentheses denote either no symbol or any one of the symbols thus enclosed in double parentheses. Thus by \((A B ((C)) D)\) we mean \((A B C D)\) and \((A B D)\) and by \((A B ((C D)) E)\) we mean \((A B C E)\), \((A B D E)\), and \((A B E)\). A term formed by enclosing a list of symbols in a single
parenthesis denotes the choice of exactly one symbol in all possible ways.

The third coordinate gives the structural change. Adjunction is indicated by grouping the set of trees to be adjoined, using a set of parentheses. In our example the optional tree dominated by PRE is to be left alone (substituted for itself) as are the trees dominated by V and BY. The second tree dominated by NP is to be substituted for the first, the morphemes BE and EN are to be adjoined as right sisters to the tree dominated by AUX; the second NP-tree is to be deleted, the first NP-tree is to be substituted for the structure dominated by PASS, and the tree dominated by ADV, if present, is to be left alone.

Terms of the structural index that are enclosed in parentheses are reflected by a corresponding term of the structural change even when applied to those cases for which an index term is vacuously satisfied by a null tree. In such a case, however, we do not permit a tree to be adjoined to this null tree. We make this restriction for the same reason we prohibit adjunction to an X or a Y; the derivation of so-called auxiliary rules discussed in Section 3.6 is enormously simplified by these restrictions.

The fourth coordinate is a list of pairs of numbers denoting pairs of trees that must be identical if the transformation is to be applicable. The fifth coordinate is merely the name of the transformation.
Binary transformations are distinguished by inclusion of two occurrences of the symbol #. They refer simultaneously to two adjacent levels of embedding, and they always erase the two sentence boundary symbols #.

For example, the transformation

\[(\text{BINARY, (X, DET, N, #, DET, ((ADJ)), N, ((LOC)), AUX, VP, #, X)} \]
\[(1, (2,6), (3,8), 0, 0,0,0,0, (\text{THAT}, 9), 10, 0, 12) \]
\[((2,5), (3,7)), \text{SUREL})\]

adopts the ADJ-tree, to the right of the first DET-tree, adjoins the LOC-tree to the right of the first N-tree, adjoins the morpheme THA to the left of the AUX-tree, and deletes the indicated trees, including the two occurrences of #.

We speak of the structure referenced by terms of a BINARY transformation lying between the two occurrences of # as the constituent sentence structure, and we refer to the structure referenced by terms lying outside the symbols # as the matrix sentence structure.

We will write and order transformations such that first singulary transformations apply at the deepest level of embedding. Next BINARY transformations will apply at the bottom two levels of embedding, erasing sentence boundary symbols and thus replacing the deepest two levels by a single new, deepest level. The process will then be cyclicly repeated starting with the application of the singulary transformations to this new deepest level of embedding.

To automatically accomplish this ordering it suffices to make every structural index of the form \((Y, #, \ldots, #, Y)\), where the structure indicated between the two occurrences of # is dominated by a single symbol \(S^1\), and to make every \(n\)-term structural change of the form \((1,2, \ldots, (n-1), n)\). Because every transformation
is of this form we simplify the statement of transformations by assuming an automatic Y and # on the left of each structural index and a # and Y on the right, and we further assume that these constituents are carried over intact in the result of each transformation. The remaining details of applying transformations are accomplished by ordering the singulary transformations and, after going completely through the list of transformations, beginning again with the first singulary transformation. The process is terminated only upon encountering a complete cycle when no transformation is applicable except possible optional transformations whose use is not selected.

Upon terminating the process, the tree produced is checked to verify that it contains no internal instances of #. If # is found anywhere other than attached to the top-most symbol S as reflected by the rule $S \rightarrow \# S^1 \#$, then the structure produced and the sentence which is its debracketization are discarded.

To illustrate this order of applying rules consider the transformational component consisting of the singulary transformation PASSIVE and the BINARY transformation previously given, the former ordered first as we require. We apply these rules to the base tree
On the first cycle of applying rules the singularly transformation is not applicable but the binary transformation is because the proper analysis satisfies the structural index of the binary transformation.

Performing this transformation gives the tree

![Tree Diagram]

34
Going through the transformational cycle again from the beginning we see that the PASSIVE transformation is applicable to this tree.

Performing it gives

which is the surface structure that after verb-affix permutation and morphophonemic transformations would give the sentence, "The book was lost by the man that is smoking the pipe."
Other elementary transformations have been proposed and utilized by different people at different times. A permutation transformation considered by Chomsky is now seldom used because it complicates the discussion of derived structure that is transformationally produced, and it is not absolutely necessary although Matthews has found such a permutation transformation useful in describing Hidatsa syntax. Work done at the MITRE Corporation has utilized the elementary transformations we consider plus several others that have been found useful. These include right and left daughter adjunction. The question of what set of elementary transformations are best suited for the description of any natural language requires additional investigation. The primitives we consider are at least a reasonable set, and have been selected for the transformational description of English by at least one group, at the Yorktown Heights, New York Research Center of the International Business Machines Corporation.

In addition to the requirements for applicability that have already been discussed, there are other conditions frequently put on transformational rules. For example, it is sometimes required that a subtree corresponding to a term of the structural index be dominated by a given symbol. We do not presently permit any such conditions, but they could be incorporated into our synthesis phase without modifying the recognition procedure we present in Chapter III.

In addition to allowing transformations to include conditions of the type we have discussed, we require that transformations meet
certain requirements necessary for recognition. Although these requirements were devised to make possible a recognition procedure, they may also be argued for various linguistic reasons. We limit ourselves here to considering their necessity in showing we have a recognition procedure.

The first such general condition on a transformational grammar is due to Matthews. Speaking loosely, it requires that in all cases where a subtree is deleted, either because of simple deletion or because other subtrees are substituted for it, that subtree must be uniquely reconstructable from the transformed tree. More precisely, in terms of our formulation of transformations the number of every term of the structural index belonging to $N \cup \{ X \} \cup \{ Y \}$ must either appear somewhere in the corresponding structural change or else must be equal to some number that does occur in the structural change.

A second general condition on a transformational grammar $G$ which is dictated by recognition considerations is that there be a bound on the depth of embedding as a function of the number of words in a sentence generated by $G$. If this condition is not met, a grammar can assign arbitrarily deeply embedded base structures to a sentence, and, in particular, can assign an unbounded number of structural descriptions to a sentence. Hence, requirements on a grammar must be established which guarantee that the depth of embedding be bounded in order to make possible a recognition procedure whose computation time is bounded as a function of sentence length.
We illustrate this problem with the following grammar:

Base Component:

$S \rightarrow \# S' \#$

$S' \rightarrow A B$ COMP

$S' \rightarrow A B c$

$A \rightarrow a$

$B \rightarrow b$

COMP $\rightarrow \# S' \#$

Transformational Component:

(BINARY, (A B # A B c #),

(1 2 6 0 0 0 0 0),

((1 4) (2 5)), EXAMPLE1)

Any structure given by the base component of the form

```
S
 /\  
# S' #
 /  
A B COMP
 /\  
# S' #
 /  
A B COMP
 /\  
# S' #
 ... 
S'
 /\  
A B COMP
 /\  
# S' #
 /  
A B c
 /\  
1 1
 a b
```
can be transformed by means of the given binary transformation into the structure

There are several ways of precluding this difficulty. To give one possible condition on a grammar that establishes a bound on the depth of embedding as a function of sentence length, let us define an embedding sequence of a structural description $D$ of a sentence produced by the CF grammar $G = (N, T, S, P)$ to be a sequence of subtrees $D_1, D_2, \ldots$ of $D$ such that each $D_i$ is a structural description of a sentence of $(N - \{COMP\}, T \cup \{COMP\}, S^1, P - \{COMP \rightarrow \# S^1 \#\})$ and also such that the distinguished symbol $S^1$ of $D_i$ is attached in $D$ to a COMP symbol of $D_{i+1}$ for $i = 1, 2, \ldots$. Consider any embedding sequence $D_1, D_2, \ldots$ of a structural description $D$. Let $D_1' = D_1$ and denote by $D_j'$ the tree produced in the course of a derivation by the binary transformation which acts on the matrix sentence structure of $D_j$ and the constituent sentence structure of $D_{j-1}'$ for $j = 2, 3, \ldots$. Let us impose the condition on a transformational grammar that $d(D_1') < d(D_2') < \ldots$. If in the course of a derivation this condition is violated, that is, if a binary transformation produces a tree at some point that has no more terminals than the tree produced by a previous binary transformation, then let us consider the derivation in question blocked. This convention is similar to our blocking convention associated with the elimination of sentences containing internal occurrences of the sentence boundary symbol $\#$.
Let us further define \( L(S^1) \) to be the length of (the number of terminal symbols contained in) the shortest sentence not containing the symbol COMP of the language specified by
\[(N - \{COMP\}, T \cup \{COMP\}, S^1, P - \{COMP \rightarrow \#S^1\#\} \cup Q)\] where \( Q \) is the set of rules representing structure that can be transformationally produced by one cycle of the singularity transformations.\(^16\) Now consider an embedding sequence \( D_1, D_2, \ldots, D_m \) which is at least as long as any other embedding sequence of a structural description \( D \). We have, by our previously postulated condition, \( d(D_1') < \ldots < d(D_m') \) and hence \( d(D_1') + (m - 1) \leq d(D_m') \). From the additional observation that \( L(S^1) \leq d(D_1') \) we obtain \( L(S^1) + (m-1) \leq d(D_m') \). and therefore the maximum depth of embedding \( (m-1) \) cannot exceed \( nL(S^1) \) where \( n = d(D_m') \) is the length of the sentence whose underlying base structure is \( D \).

An alternative to blocking a derivation if \( d(D_{i+1}) \leq d(D_i) \) for some \( i \) might be to go back to the last obligatory transformation that was performed and relax the condition that it be performed. The relative descriptive usefulness of this convention as opposed to simply blocking is an empirical question. Whichever convention proves most useful is immaterial with respect to the recognition procedures of Chapter III as they all can be adapted to either convention.

Finally we impose a restriction on the number of consecutive applications of a repeated transformation which are allowed at any level. To see the necessity of a condition of some sort, consider
the following grammar:
\[ S \rightarrow \# S^l \# \]
\[ S^l \rightarrow A B A \]
\[ A \rightarrow a \]
\[ A \rightarrow c \]
\[ B \rightarrow b \]

(REPEATED, (A B A), (3 2 1), ( ), EXAMPLE2)

Applying this transformation to the structure

```
  S
 /\  /
/  | /  |
S^l # # 
/   |   
A B A
| | |
/| | |
a b c
```

gives

```
  S
 /\  /
/  | /  |
S^l # # 
/   |   
A B A
| | |
/| | |
c b a
```

and reapplying this repeated transformation gives the original tree.

An unbounded number of applications of this transformation are possible.

We must somehow preclude such a state of affairs, and one way is to limit the number of consecutive applications of a repeated transformation. The number of consecutive applications of a repeated transformation that might be useful depends on the length and complexity of a sentence.
We could choose for this limit some monotone increasing function of the number of embedded sentences or of the maximum depth of embedding. If we choose the latter, we see from our previous considerations that the number of consecutive repeated rule applications must be less than some other monotone increasing function \( h(n) \) of sentence length \( n \). We will take \( h(n) = n \) for convenience, but if empirical results arising from descriptive work on various languages should dictate some other function \( h(n) \), we can modify the recognition procedures presented in the next chapter so as to make use of that function just as easily as \( h(n) = n \).

This concludes our definition of a class of transformational grammars, and we turn in the next chapter to the recognition problem for a grammar of this class.
CHAPTER THREE

THE RECOGNITION PROCEDURE

3.1 OVERVIEW

As we mentioned briefly in the first chapter, the procedure we present in this dissertation is indeed an analysis-by-synthesis procedure. We will be solely concerned, however, with the so-called "preanalysis" phase of analysis-by-synthesis. That is, we will develop an analysis procedure that produces a set of structural descriptions (among which are included all of the correct structural descriptions) for subsequent verification or rejection by a synthesis phase.

So far we have not defined the term structural description with respect to transformational grammars. It might reasonably be assumed that by structural description we mean some encoding of the complete derivational sequence. Instead of this, however, we denote by a transformational structural description of a sentence just the set of base structures that can be mapped into the given sentence by the transformational component. We do this for two reasons. First, the base structure reflects all of those relationships necessary for interpretation by the semantic component in the transformational grammar that we are concerned with. In addition, by defining structural description in this way we bypass certain problems of ambiguity related to the application of transformations.

Even though we define structural descriptions as base structures, we still require the use of specifiers for the synthesis phase. It was originally intended that specifiers be taken to consist of the
base tree plus an indication of what transformations were to be applied at each level of embedding and the number of times repeated transformations were to apply at each level. We were not concerned about the redundancy of such specifiers because we merely needed any complete specification of which transformations to apply and how to apply them to the base structures in order to get the derived constituent structures. Such a specification was required to generate sentences and thus check the legitimacy of every potential structural description. Accordingly, this type of specifier is presently used in the computer program described in Chapter IV. It was recently observed by J. Keyser, however, in using that computer program to recognize sentences with respect to a grammar of his own devising, that these "specifiers" do not necessarily give enough information to indicate a unique derivational sequence. If at any point in a derivation a transformation is applicable in more than a single way, that is, if two or more distinct proper analyses of the tree at some point satisfy the terms of the next transformation's structural index, then the specifier should indicate the proper analysis that is to be made relative to applying that transformation. The particular example in question was trivial and could have been handled merely by making a natural convention, but we have precluded any further troubles arising from multiple proper analyses by not insisting that "specifiers" denote completely specified derivational sequences; when in the synthesis of a sentence from an "incomplete" specifier a transformation can be applied in more than one way, we separately consider each possibility and see whether any choice produces the given structure.
Our analysis procedure works roughly as follows: we make use of inverse transformations that do not, as do true inverse transformations, map trees into trees. Our inverse transformations merely map a sequence of trees satisfying an inverse structural index into another sequence of trees. We mechanically compute an inverse transformation from each given transformation. The generative order is reversed in recognition so that identification of structure proceeds from outer to inner levels of embedding. Starting then at the outermost level, we determine whether the last-ordered singulary transformation could have been applied. This is done by trying to build up structure from the terminals of the sentence to obtain a sequence of trees satisfying the terms of the inverse structural index of the last singulary transformation. In building up this structure use is made of an augmented CF grammar which is the original context-free base component of the transformational grammar with its set of productions augmented by the inclusion of certain so-called auxiliary rules. These are context-free rules mechanically computed from the given transformations and base component context-free rules.

If structure can be built up satisfying the inverse structural index, the inverse transformation is performed and the procedure of building up the output sequence of trees to satisfy the terms of the next transformation's inverse structural index is repeated. As a separate consideration the transformation is also not applied.
After each COMP-embedding level cycle of inverse singulary transformations, the inverse structural indices of the binary transformations are used in the same manner to determine applicability of inverse binary transformations. Reversal of binary transformations gives sequences of trees corresponding to both the matrix and the constituent sentence. The condition on transformations of unique reconstructability insures that both matrix and constituent sentence outputs are sufficiently determined to permit separate continuations. If the constituent sentence continuation is eventually built up to the sentence symbol $S^1$, it is attached under the COMP symbol of the matrix sentence. Constituent sentence continuations are processed with respect to the entire complement of inverse transformations starting, as originally, with the last singulary transformation. Matrix sentence continuations, on the other hand, are only subjected to possible application by the same or other binary transformations. After no further binary transformations are applicable, each resulting sequence of trees is built up to the sentence symbol $S^1$ using the original base component rules. The set of structures thus obtained includes the set of structural descriptions of the given sentence, and it only remains to determine which of these structures are valid structural descriptions by synthesis.

We will leave our brief description at this and turn to a detailed presentation of our procedure.

3.2 DEFINITION OF INVERSE TRANSFORMATIONS

As we have already indicated, we make use of inverse transformations,
each of which is mechanically computed from a given transformation. Each singulary inverse transformation maps a sequence of trees satisfying an inverse structural index into another sequence of trees.

Each binary transformation, on the other hand, maps a single sequence of trees into two other sequences.

Every inverse transformation has a so-called inverse structural index that is defined as the sequence of symbols denoted by the structural change of the corresponding transformation. Recall that the structural change of a transformation consists of integers and members of T ∪ M, and the non-zero integer n refers to the nth element of the corresponding structural index. The inverse structural index of an inverse transformation consists of the sequence of symbols appearing in the structural change of the corresponding transformation, each non-zero integer n being replaced by the nth term of the structural index of that transformation and each occurrence of the integer 0 being omitted. A precise definition of the inverse structural index is given by the LISP program listing of Appendix D. The function SETTRANS computes inverse transformations from corresponding transformations, and, in particular, the inverse structural index of a transformation N3 is given by

(SIMPLIFY, (TRANSFORM, (NUMBARG, (CADR, N3)), (CADDR, N3))).

We say that a sequence of trees \((t_1, t_2, \ldots, t_n)\) satisfies an inverse structural index

\[I' = (i'_1, i'_2, \ldots, i'_n)\] if \(i'_j\) is the name of the top-most node on the tree \(t_j\) for \(j = 1, \ldots, n\). An inverse transformation \(\tau'\) is said to be applicable to a sequence of trees if the sequence satisfies the inverse structural index of \(\tau'\).
The **inverse structural change** of an inverse singulary transformation specifies a new sequence of trees into which a given sequence of trees is to be mapped. It consists of a sequence of symbols of $T \cup M$ and integers denoting the corresponding terms of the inverse structural index. There is no requirement that the number of terms in the inverse structural index be the same as the number of terms in the inverse structural change. The output sequence of trees produced by an inverse transformation is a sequence of trees satisfying the structural index of the corresponding transformation. More precisely, the inverse transformation is defined such that the following holds: Let $P$ be the proper analysis of a tree $R$ that satisfies the structural index $\Gamma$ of a transformation $\tau$. If $\tau$ is performed on $R$ giving another tree $R'$, then denote by $P'$ the proper analysis of $R'$ which consists of trees of $P$ and constant symbols from the structural change of $\tau$. Now the inverse transformation $\tau'$ corresponding to $\tau$ maps the proper analysis $P'$ back into the original proper analysis $P$. It is obvious from our condition of unique reconstructability that we can mechanically determine such an inverse transformation for every singulary transformation. As before, we refer to the LISP function SETTRANS of Appendix D for a precise definition of inverse transformation. The inverse transformation corresponding to a singulary transformation $N3$ is given by

$$
(\text{LIST}, \text{(SIMPLIFY, (TRANSFORM, (NUMBARG, (CADR, N3)), (CADDR, N3))}), \\
\text{(INVERS1, (NUMBARG, (CADR, N3)), (TRANSFORM, (NUMBARG, (CADR, N3)), \\
(CADDR, N3)), (CADR, (CADDR, N3)))}, \text{(CAR, N3), (CADDR, N3))}.
$$

Inverse binary transformations differ only in specifying two output sequences of trees. The constituent sentence portion of the output
of an inverse binary transformation consists of the sequence of
trees satisfying that portion of the forward transformation struc-
tural index lying between the two occurrences of the symbol #.
The matrix sentence portion of the output, on the other hand,
consists of the sequence of trees satisfying the two portions
of the forward transformation structural index lying to the
left of the first # and to the right of the second #, the symbol
COMP marking the junction between these two matrix sentence por-
tions. That portion of the function SETTRANS of Appendix D
beginning at B defines the computation of inverse binary trans-
formations.

To illustrate this definition of inverse transformation we
give the inverse transformations corresponding to the two sample
transformations of the previous chapter. Corresponding to the
PASSIVE transformation we have:
(((PRE)), NP, AUX, BE, EN, V, BY, NP, ((ADV))),(1, 8, 3, 6, 2, 7, PASS, 9), OBLIG, PASSIVE)

Notice that the four ordered components of an inverse singular
transformation are:

inverse structural index, inverse structural change, type of
transformation, and name of corresponding forward transformation.

The inverse transformation corresponding to the previously given
binary transformation is:
(BINARY, (X, DET, ((ADJ)), N, ((LOC)), THAT, AUX, VP, X ),
(1,2,4, COMP, 9), (2,3,4,5,7,8), SUREL)
3.3 THE SIMPLIFIED PROCEDURE

Having defined the inverse transformations with which we will deal, we turn our attention to a simplified recognition procedure that will be pedagogically useful in understanding the more efficient complex procedure that will subsequently be presented. We reverse the generative order of applying transformations and hence begin by considering whether or not the last singular transformation may have been performed at the top-most level of embedding. If it was performed, the ultimate surface structure tree must have a proper analysis that satisfies the terms of the inverse structural index corresponding to this last-ordered singular transformation. Assuming that this last transformation was performed, we partition the given sentence of \( n \) terminal symbols into \( m \) consecutive mutually exhaustive segments in all possible ways, \( m \) being the number of terms in the inverse structural index of the last singular transformation. If the \( j \) th term of the inverse structural index is the symbol \( X \) or an optional list of symbols (enclosed in double parentheses) the \( j \) th segment into which the \( n \) terminal symbols are divided may be null, but otherwise it must consist of at least one terminal symbol.

Each of these partitions represents a potentially correct proper analysis of the final surface tree, the derived constituent structure in question being unknown, of course. If then we perform the inverse transformation corresponding to the last singular transformation on such a partially specified sequence of trees, we obtain another partially specified sequence of trees that constitutes a (partially specified) proper analysis of the derived constituent
structure tree prior to application of the final transformation. In particular, the transformed string of terminals is the debracketization of the derived constituent structure tree prior to performing the last transformation.

It is necessary not only to separately consider continuations arising from each distinct partitioning of the given sentence, but also to consider the continuation in which the last transformation was not applied at all. Thus in next examining the potential application of the penultimate singulary transformation, it is necessary to consider each transformed terminal sequence and also the given sentence itself. In each case a terminal sequence is again partitioned in all ways commensurate with the inverse structural index of the penultimate singulary transformation and the corresponding inverse transformation is applied to give, after debracketization of unknown structure, a new sequence of terminals. By a repetition of this process we consider each singulary transformation in the reverse of the generative order. When all singulary transformations have been exhausted we have obtained a set of debracketizations of potentially valid derived constituent structure trees existing just prior to application of the final sequence of singulary transformations. That is, this set will include a debracketization of every derived constituent structure tree existing just prior to application of this final sequence of singulary transformations.

Further reversing the generative cycle, we consider the
potential application of binary transformations at the top-most level, i.e., at the top level and the next-to-the-top level.

We consider binary transformations also in the reverse of their generative order, and hence our recognition procedure is valid for a generative system employing ordered binary transformations, but since we postulate a system whose binary transformations are order-independent, our concern for binary order is of no consequence. If a binary transformation was applied, there must be a proper analysis of the derived constituent structure tree just prior to application of the final cycle of singulary transformations that satisfies the terms of the last inverse binary structural index. Hence, again we partition each terminal sequence at that point and apply the corresponding inverse binary transformation. Each such application gives two sequences of trees, a matrix sentence sequence and a constituent sentence sequence. From the definition of the inverse binary transformation it is obvious that if the constituent sentence sequence of trees is both preceded and followed by the symbol #, and if, furthermore, the resulting sequence is substituted for the symbol COMP of the matrix sentence sequence, then the resulting sequence of trees is a proper analysis of the tree existing prior to the performance of the binary transformation in question. As in the case of singulary transformations we do not completely know the structure of this proper analysis, but merely know its debracketization and the structural index that it
satisfies. It is clear, however, that the constituent sentence sequence of terminals is the debracketization of the constituent sentence tree just prior to embedding. Hence, in further reversing the generative cycle it is necessary to process this constituent sentence sequence of terminals in exactly the same fashion as the original sentence was processed, beginning with the last ordered singulary transformation. If this leads ultimately to a sentence structure dominated by the sentence symbol $S^1$, this structure, surrounded by a pair of symbols #, is attached under the COMP symbol of the matrix sentence sequence of partially specified trees and the recognition procedure is continued for that sequence. Further reversal of the generative cycle here requires merely that more highly ordered inverse binary transformations be considered. At every stage non-performance of an inverse transformation and also performance of the inverse transformation in all possible ways are separately considered in turn.

After exhausting the binary transformations, sequences of terminals are obtained which are the debracketizations of base structure trees. Of course, every such sequence of terminals is not a base tree debracketization, but the debracketization of every valid underlying structural description (i.e. base tree) must be included in the set of all these terminal sequences. Hence, if each potential sequence of terminals is recognized with respect to the given base component CF grammar, we will obtain a set of structures that includes all of the structural
descriptions of the given sentence.

It is necessary, of course, to determine which of these structures are structural descriptions, and this requires a synthesis for each structure. It is easy to make available for this synthesis not only base structure but also the sequence of transformations to be applied at each level.

The remaining matter that must be disposed of to establish that the procedure we have presented finds all of the structural descriptions of a given sentence is a proof that this procedure terminates. There are only two sources of potentially unbounded continuations that need concern us. First, we have seen that applicability of a binary transformation can lead to a recursive computation in which the constituent sentence terminal sequence is processed in exactly the same fashion as was the original sentence. Clearly this recursion must terminate if each successive constituent sentence terminal sequence contains less symbols than the preceding such sequence. Recall that in Chapter II we required precisely this condition. (In the notation of that chapter we required that

\[ d(D_1') < d(D_2') < \ldots \].) Hence, if in the course of recognition an inverse binary transformation produces a constituent sentence sequence which does not contain less terminals than the constituent sentence sequence of the previously performed binary transformation, then this continuation can be terminated. It could not possibly lead to a valid structural description because the condition

\[ d(D_i') < d(D_{i+1}') \] does not hold for some \( i \).
Another potential source of unbounded continuations is associated with the use of repeated singulary transformations and binary transformations, which may also be applied more than once at some point in a derivational cycle. Recall, however, that in Chapter II we required that the number of consecutive repeated rule applications must be less than the number of terminal symbols \( n \) in the sentence being recognized. Hence, in recognition, no inverse repeated transformation need be applied more than \((n - 1)\) consecutive times. If any continuation involving consecutive repeated applications of some transformation is terminated on the \( n \)th such application, and if any continuation for which we have \( d(D_{i+1}') \leq d(D_1') \) is terminated, then the recognition procedure must halt.

This recognition procedure is, of course, enormously inefficient, and we have included its treatment merely to simplify our subsequent presentation. To get some idea of just how inefficient this procedure is, we note that the number of parallel paths that must be followed for an underlying base structure of depth \( p \) assuming at most only a single COMP is found at each level of embedding is

\[
\sum_{i=1}^{p+1} \prod_{j=1}^{k} \left[ \frac{n_{ij} - 1}{m_{ij} - 1} + 1 \right]
\]

where \( m_{ij} \) is the number of terms in the \( j \)th inverse structural index, \( n_{ij} \) is the length of the terminal sequence in the \( i \)th iteration just prior to the application of the \( j \)th inverse singulary transformation, and \( k \) is the total number of singulary transformations. In particular,
if we replace
\( n_{ij} \) by \( n = \max \{ n_{ij} \} \) and
\( m_j \) by \( m = \min \{ m_j \} \) we
have the bound \((p+1) \left[ \left( \frac{n-1}{m-1} \right) + 1 \right]^k\).

for values \( p = 3, n = 15, m = 7, k = 20 \) this bound is \( 4 \cdot (7294)^{20} \),
clearly beyond practicality. \( ^{21} \)

Before leaving this crude recognition procedure, we mention
one obvious modification that greatly increases its efficiency.
Having partitioned the given sentence with respect to the first
inverse structural index, and having performed that first inverse
transformation, we obtain a partially specified proper analysis
of the derived constituent structure tree just prior to performing
the last transformation. Now in the procedure just described, we
dismissed our knowledge of that partially specified proper analysis
and allowed the sequence of terminal symbols to be partitioned into
as many segments as there are terms in the next singulary inverse
structural index in all possible ways. It is clear, however, that
although a sequence of terminals previously grouped together in a
single subtree can subsequently satisfy a sequence of indices, and
although several contiguous subtree terminal sequences can be grouped
together to satisfy a single index, it is not possible for some proper
subsequence of a debracketed proper analysis tree to be grouped with
adjacent terminals in satisfying some structural index term. Hence,
partitioning need not be exhaustive, but can, instead, be drastically
reduced at each step by taking cognizance of the partially specified
proper analysis that is known of the tree at every point. Although
such a modification of our procedure makes its significantly more efficient, the amount of computation required still exceeds the limits of a practical recognition procedure. In the sequel we consider an alternative procedure that is much more efficient although not so conceptually simple.

3.4 THE IMPROVED PROCEDURE

The procedure we actually programmed as is described in Chapter IV, and in Appendix D is easily described as a modification of the previously presented procedure. Suppose, for the moment, that for a given sentence whose structural descriptions we wish to find, we have available a set of auxiliary CF rules which includes rules representing all transformationally produced structure at each step of the derivation of the sentence. In Section 3.6 we consider the problem of how these auxiliary rules can be mechanically determined for a given transformational grammar.

We again make use of the previously defined inverse transformations and begin recognition by reversing the generative sequence and considering whether or not the last singulary transformation may have been applied at the top-most level. If this transformation was performed, there must be a proper analysis of the final surface structure tree satisfying the inverse structural index of the last singulary transformation. Each tree of this proper analysis must, of course, consist of structure given by the augmented CF grammar which has as its productions the original base component CF rules and the auxiliary CF rules. Hence, it must be possible to partition the sentence such that the $j$th segment is the debracketization of a tree that is:
(1) dominated by the $j$th term of the inverse structural index of the last transformation and is (2) composed entirely of structure given by the base component CF rules and by the auxiliary CF rules. The task of determining all such proper analyses is accomplished by the LISP function PROCES defined in Appendix D.

This function PROCES works roughly as follows: Denote by $G$ the CF grammar formed by supplementing the rules of the original base component CF grammar with the auxiliary CF rules. If we are to build the sentence consisting of terminal symbols $w_1, w_2, ..., w_n$ into a sequence of trees dominated by $i_1, i_2, ..., i_m$, it must be possible to recognize some string $w_1, w_2, ..., w_p$ ($p < n$) as an $i_1$ with respect to $G$. It is not difficult to generalize most of the known CF parsing algorithms so as to produce all strings $w_1, w_2, ..., w_p$ that are derivable from $i_1$ together with their associated structure. Having done this, we are faced with the problem of building $w_{p+1}, w_{p+2}, ..., w_n$ into a sequence of trees whose roots are $i_2, ..., i_m$.

By repeated application of the same procedure it is clear that all possible sequences of trees that satisfy the structural index $i_1, i_2, ..., i_m$ can be found. It is also possible to permit any of the $i_j$ to be rule schemata terms $X$. This could be done by adding additional rules that allow $X$ to generate the universal language, but it was in fact done by taking every initial subsequence of a given sequence $S$ (including the null sequence) when faced with the task of building up from $S$ to $X$. Finally, it is possible
to modify the above procedure so as to allow optional or alternative choice terms of an inverse structural index in a way which is more efficient than considering separate transformations.

Now if the sentence can be partitioned in one or more ways such that the segments are debracketizations of a proper analysis that satisfies the inverse structural index of the last singulary transformation, then we can apply the corresponding inverse transformation to obtain a new sequence of trees. Even if the last transformation is applicable, however, we have no assurance that it must indeed have been applied in generating the sentence in question, so we must also not perform the last inverse transformation, and we must follow this continuation as well as those resulting from performing the last inverse transformation.

Up to this point the only difference between the previously considered crude recognition procedure and the improved procedure is that the former partitions in all possible ways whereas the latter uses the augmented CF grammar. At this point, however, there is a further difference. We now retain all structure obtained, recognizing that it constitutes a proper analysis A of the derived constituent structure tree at this point. As before, we next consider whether or not the penultimate transformation might have applied, but in the present procedure we see that this is possible only if it is possible to obtain in the following manner a new sequence of trees S' that satisfies the inverse structural index I' of the penultimate singulary transformation: successive trees of A can be grouped together

59
in conformity with the rules of the CF grammar $G$ to satisfy a single term of $I'$ and/or any tree (or trees) of $A$ can be decomposed into a proper analysis of $q$ terms that satisfy $q$ successive terms of $I'$. It is evident how the previously discussed function $\text{PROCES}$ can be generalized to permit all combinations of building up trees to form larger trees and analyzing trees into their proper analyses so as to satisfy the terms of an inverse structural index, and this generalization is indeed reflected in the function $\text{PROCES}$ of Appendix D.

An additional operation makes possible the abortion of many continuations that could not possibly lead to a valid structural description. We note that after applying an inverse transformation we have a proper analysis of the derived constituent structure tree just prior to performing the corresponding forward transformation. If then by performing a CF recognition with respect to the augmented grammar we find that this proper analysis cannot be recognized as a sentence, we know that the continuation in question cannot possibly lead to a valid structural description and can, therefore, be abandoned. It might appear to the reader that it is always possible to build from a structural index up to the symbol $S^1$, but it must be kept in mind that as actually applied, various sequences of trees appear in places corresponding to the index term $X$, and this can lead to failure in building up to the sentence symbol. The question of whether or not the efficiency gained in eliminating continuations by employing this test is greater than the price which must be paid for conducting
the test itself is an empirical question that depends very much on the grammar in question. In particular, when the inverse structural index of an inverse transformation contains no instance of the symbol X, then the performance of the test in question can only be a waste of time. The version of the LISP function IMPOSE included in Appendix D does make use of this test.

The remaining details of our improved recognition procedure are identical to those of the previously given procedure. In particular, the same considerations that established the termination of the former procedure are applicable in the present case.

3.5 A MODIFICATION OF THE IMPROVED PROCEDURE

A significant modification of the procedure of Section 3.4 is suggested by the condition, made use of in that procedure, that it must always be possible to build up the sequence of trees that results from performing any inverse transformation up to the sentence symbol $S^1$ by means of the augmented CF grammar. If it is possible to build up to $S^1$, the sequence of trees is indeed a potentially valid proper analysis of the derived constituent structure tree at that point. We note, in addition, that the set of trees thus obtained must contain all of the derived constituent structure trees existing at this point which are compatible with the structure already determined. In the method of Section 3.4 only the existence of at least one such structure was made use of, and the structure itself was discarded; the sequence of trees produced by an inverse transformation was both built up
and decomposed by means of the function PROCES to satisfy the terms of the inverse structural index of the next inverse transformation. We observe, however, that, alternatively, it is possible to eliminate the use of the function PROCES by merely finding all proper analyses of the trees obtained in building up to $S^1$ which satisfy the inverse structural index of the next inverse transformation.$^{23}$ This essentially substitutes computation relevant to the finding of proper analyses of a tree for computation performed in building up and decomposing a set of trees to satisfy an inverse structural index. If the test of building up to $S^1$ is required, then the method of Section 3.4 and the modified procedure of this section find the same set of proper analyses. The relative efficiency of these two procedures, however, may differ significantly. Results for one grammar, which are discussed in Appendix C, showed the procedure of Section 3.4 to be more efficient.

As has already been pointed out, the method of 3.4 does not require that all structural descriptions be found in building up to $S^1$. Knowledge that a single structural description exists is sufficient. Hence, in performing the context-free parsing in question by means of an equivalent pushdown-store automaton program $^{[13]}$ it is necessary to find only a single automaton acceptance sequence, and there is no necessity to translate that acceptance sequence into the corresponding tree structure representation. The modified procedure of this section, however, must find all structural descriptions in building up to $S^1$. 

62
There is, however, a theoretical advantage (which is also at least a potentially important practical advantage) for the modified procedure; it can directly incorporate supplementary conditions on a transformation by means of procedures not applicable to the method of Section 3.4. To see this, recall that transformations can make use of conditions on the structure above the trees referenced in a proper analysis. The modified procedure, having this structure available, can verify that these conditions are met immediately after performing an inverse transformation and building up to $S^1$. Continuations leading from the further consideration of any structure for which some condition fails can be abandoned. The method of Section 3.4, however, does not obtain full derived constituent structure at each step and, hence, a final synthesis phase is essential in conjunction with this method.

In addition to conditions such as the requirement that a certain tree be dominated by a given symbol, we have the general condition that obligatory transformations must be performed when applicable. This, too, can be incorporated into our modified procedure but not into the procedure of 3.4. If an inverse transformation cannot be applied at some step, then the derived structure existing before and after consideration of this transformation at this point in generating the sentence in question must be identical. If, however, the structure at this point has a proper analysis satisfying the structural index of the corresponding forward transformation, then this transformation, being obligatory, must have been applied. This contradicts the previously established
fact that this transformation could not have been applied because its inverse transformation could not be made. Hence, the structure in question cannot be an intermediate structure of a valid derivation, and continuations leading from it can be abandoned. Similarly, if the inverse transformation corresponding to an obligatory transformation is applicable, and if, furthermore, this obligatory transformation itself can be applied to the structure existing at this point, then it is clear that not applying the inverse transformation leads to a contradiction. Hence, in such a case the performance of the inverse transformation is obligatory.

3.6 DETERMINATION OF AUXILIARY RULES

In this section we consider the problem of deriving a finite set of auxiliary CF rules reflecting transformationally produced structure distinct from that given by base component CF rules. We give an effective procedure for computing a finite set of auxiliary rules; this produces every rule that could possibly reflect derived constituent structure at any point in the derivation of a given sentence by means of a given transformational grammar. Note that we have limited our problem to finding rules reflecting the structure of a particular sentence. We will in fact consider finding a set of rules adequate for representing the structure of any sentence of \( n \) or less symbols, but it might well be profitable to make use of other properties of the particular sentence to be analyzed.

This limitation of our attention to sentences of length \( n \) or
less is an important observation. Having failed to make it, we labored for some time with problems related to the termination of auxiliary rule-generating algorithms, eventually abandoning the attempt to find such an algorithm in favor of a procedure scarcely better than the first crude procedure we discussed. We were, of course, thwarted by problems illustrated by the following simple abstract grammar: Suppose we have the base component

\[ S \rightarrow \# S' \# \]

\[ S \rightarrow A \text{COMP} B \]

\[ S' \rightarrow A B C \]

\[ \text{COMP} \rightarrow \# S' \# \]

\[ A \rightarrow a \]

\[ B \rightarrow b \]

\[ C \rightarrow c \]

and the transformational component:

\( \text{(OBLIG, (X B C X), (1 (2 3) 3 4), ( ), SING)} \)

\( \text{(BINAR Y, (A \# A B C X \# B), (1 0 0 4 5 6 0 8), ((1 3)), BIN1)} \)

It is clear that by taking an arbitrarily deep base structure of the form
and applying both rules at each level, it is possible to obtain
structure of the form

\[ S^1 \]

\[ B \quad C \quad C \quad \ldots \quad C \]

Hence, we see that there is an unbounded number of auxiliary rules
required, and there can be no finite set of rules adequate for
representing the structure of any sentence generated by the
grammar.

Attempts were made to avoid this essential difficulty by
relaxing our demands and asking merely for a weakly equivalent
grammar, but we need not dwell further on them because the
limitation to a sentence of bounded length made possible the
procedure we describe below. This follows from our condition
that bounds the maximum depth of embedding of a sentence as a
function of its length. The procedure we give runs through the
transformations in generative order, producing auxiliary rules
reflecting every structure those transformations could generate.
A bound on the number of passes through the generative cycle is
required to insure termination of the algorithm. We observe that
the procedure of transformational derivation starts with applica-
tion of singulary transformations, continues with binary transform-
ations, alternates repetitions of this singulary-binary cycle,
and concludes with a final application of just the singulary
transformations. We duplicate this order of considering trans-
formations in producing auxiliary rules at each point in our cycle
reflecting structure that could have been produced by the trans-
formational grammar at the corresponding point in its derivational cycle.

Suppose first that transformations make no use of the rule schema symbol \( X \) in their structural indices. Then if we find the set of trees, each of which is dominated by the symbol \( S^1 \) and which terminates in the sequence of terms comprising the structural index of the first singulary transformation, we will have enumerated the upper structure of all possible base trees on which the first transformation can initially act. Performing this first transformation on each of these partially specified trees can only modify the structure above the nodes satisfying this transformation's structural index, and this structure is completely known. Hence, we see that by merely transforming each of these partially specified trees and extracting from the resulting transformed structures the set of rules reflecting structure not given by base component CF rules, we will have found a set of auxiliary CF rules which includes all structure produced transformationally by an initial application of the first transformation.

If the first transformation is a repeated transformation, the auxiliary rule-producing procedure must be repeated as many times as the bound on the number of times that transformation can be repeated in a single cycle. In each repetition the CF grammar used, however, is not the base component grammar, but is the previously used grammar augmented by the inclusion of the auxiliary rules produced by the previous application of the
index-parsing, transforming, rule-extracting procedure. Upon completion of this procedure for either a repeated or a non-repeated transformation, the procedure is applied to the next transformation, the previously augmented grammar being used for CF parsing in every case.

After thus considering the singulary transformations, the identical process is carried out on each of the binary transformations in turn. When they have been exhausted, the procedure is repeated starting with the first singulary transformation. The CF grammar used is, again, the augmented grammar resulting from application of the process to the first cycle of transformations, so additional auxiliary rules can be produced. There is a bound on the depth of embedding of a base structure corresponding to a sentence of n symbols and hence our auxiliary rule-producing procedure can be terminated after a finite number of cycles.

It is clear that the auxiliary rules thus obtained represent every possible structure that could possibly be formed in generating a sentence of n symbols. This procedure was suggested by Matthews as a simpler alternative to a procedure we will discuss subsequently. Matthews' procedure is very elegant but, unfortunately, it is not applicable to transformations whose structural indices include occurrences of the rule schema symbol X. There are several alternatives that can be employed in handling this frequently occurring situation. One possibility is to eliminate the use of X entirely by replacing transformations that contain it by a finite number of separate transformations.
each of which contains a sequence of nonterminal symbols in place of \( X \). \( X \) cannot stand for an unbounded sequence of trees because we are limiting our treatment to sentences of \( n \) or less symbols. Hence, this procedure is valid although it is not convenient because it requires enumeration of the set of nonterminal sequences that each \( X \) can denote.

An alternative approach is to modify our basic CF recognition procedure so as to allow input strings containing the symbol \( X \). Although any of the CF recognition procedures discussed in Reference 13 can be modified to do this, we will merely give a modification of one of those procedures, the STB (selective top-to-bottom) algorithm of that paper.

We make use of a rather bizarre formulation of a Turing Machine, the machine \( TM \) of Reference 13. It is represented by a pair \((V,R)\), where \( V \) is an alphabet and \( R \) is a set of TM instructions. This machine has two pushdown tapes \( \alpha \) and \( \beta \) whose ends are marked by a special symbol \#. The rules in \( R \) have the form \((A_1 \ldots A_m, C_1 \ldots C_p) \rightarrow (B_1 \ldots B_n, D_1 \ldots D_q)\) where the \( A_1 \ldots A_m, B_1 \ldots B_n, C_1 \ldots C_p, \) and \( D_1 \ldots D_q \) are strings of symbols including the null string \( \lambda \). The effect of executing this TM rule is that if the top \( m \) symbols on tape \( \alpha \) are \( A_1 \ldots A_m \) and the top \( p \) symbols on tape \( \beta \) are \( C_1 \ldots C_p \), then these symbols are erased and the symbols \( B_1 \ldots B_n \) are written on tape \( \alpha \) and \( D_1 \ldots D_q \) on tape \( \beta \).

The TM is nondeterministic. When \( m \) rules are applicable to a given configuration, the TM is to be thought of as following all
m paths in parallel. A TM path terminates when no TM instruction is applicable to the path’s current configuration. The TM halts when all its paths have terminated. A TM recognizes a string \( A_1 \ldots A_m \) as a sentence \( S^1 \) if there is some finite sequence of its rules producing \( \# \) on \( \alpha \) and on \( \beta \) when it is started with \( A_1 \ldots A_m \# \) on \( \alpha \) and \( S^1 \# \) on \( \beta \). Every such sequence of rules will be called an acceptance sequence for \( A_1 \ldots A_m \) (recognized as \( S^1 \)).

In terms of such a TM program we give the following algorithm for recognition of a string containing \( X \)'s with respect to a CF grammar \( (N, T, S^1, P) \):

<table>
<thead>
<tr>
<th>Conditions</th>
<th>TM Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow C_1 \ldots C_n \in P ) and either ( P(a, C_1) ) or else ( a = C_1 )</td>
<td>((a, A) \rightarrow (a, C_1 \ldots C_n))</td>
</tr>
<tr>
<td>( D \in T \cup N )</td>
<td>((D, D) \rightarrow (A, A))</td>
</tr>
<tr>
<td>( A \in N )</td>
<td>((X, A) \rightarrow (X, A))</td>
</tr>
<tr>
<td></td>
<td>((X, A) \rightarrow (A, A))</td>
</tr>
</tbody>
</table>

\( P(a, C_1) \) is a truth function whose value is \( t \) if and only if \( (N, T, S^1, P) \) permits a bracketing of the form \([ r^n [a \ldots ], (n \geq 0) \), that is, if the grammar permits any sequence of left branchings from \( C_1 \) down to \( a \).
The TM programs given by this algorithm must be run with a
different 'shaper' convention than that usually used. Any path
is to be aborted whenever the number of symbols on the β tape
exceeds the difference between the number of remaining symbols
on the α tape (not counting occurrences of X) and the number of
previous applications of a rule \((X, A) \rightarrow (X, \Lambda)\).

A drastic decrease in the efficiency of the STB procedure
results from making these changes, and it is not clear that
practical usage of this procedure is feasible. Hence, we present
yet another procedure that may or may not be more efficacious.
A procedure for determining a set that includes all new auxiliary
CF rules that can be produced by a transformation \(\tau_1\) operating on
a tree whose structure reflects rules of a given augmented CF grammar
is as follows:

The structural change of \(\tau_1\) consists of terms indicating either
deletions (if the terms are 0) or adjunction-substitution, the adjunc-
tion frequently not occurring. Recall furthermore that the \(j\) th
term indicates a change to be made to the proper analysis tree
satisfying the \(j\) th term of the structural index. We denote deletion-
type elementary structural changes by \(A > \Phi\) (i.e., for some term \(A\)
of a structural index there is a 0 in the corresponding position
of the structural change). Similarly, we denote adjunction-
substitution elementary structural changes by \(A_1 > A_2 \ldots A_p\)
(i.e., for some term \(A_1\) of a structural index there is in the
corresponding position of the structural change a sequence of
numbers and morphemes denoting the string \(A_2 \ldots A_p\)).
It is obvious that the only possible new auxiliary CF rules are found by applying the elementary structural changes of $r_i$ to the rules of the current augmented CF grammar. To simplify our presentation we give first a rather crude procedure for obtaining a sufficient set of auxiliary rules to reflect all possible derived constituent structure; later refinements will be added to eliminate many auxiliary rules given by this crude procedure.

Pick a rule $R_j$ of the base component grammar. From the set of elementary changes (deletions and adjunction-substitutions of $r_i$) select those elementary changes whose left hand members are included among the symbols of the right hand member of $R_j$. Denote the set of these elementary changes by $C_{ij}$. Note that $C_{ij}$ may contain more than one copy of an elementary change, distinct copies coming from separate terms of the structural change of $r_i$ when a symbol is repeated in the structural index of $r_i$. Denote by $U_{ij}$ the set of all subsets of $C_{ij}$. Now for each member of $U_{ij}$ apply the indicated elementary changes to $R_j$, (i.e., $A > \phi$ indicates the deletion of $A$ and $A_1 > A_2 \ldots A_p$ indicates the substitution of $A_2 \ldots A_p$ for $A_1$). If an elementary change applies in more than one way to $R_j$ (because of a multiple occurrence of symbol of $R_j$), then separately apply the elementary changes to $R_j$ in all distinct ways.

Append all these auxiliary rules with non-null right hand members which are thus produced to an initially empty list $L_j$. If, however, an auxiliary rule is produced with an empty right hand member, then delete the left hand symbol of this rule from the right member of
each augmented grammar rule and append the rules so obtained to \( L_j \). If in this process another rule is produced which has an empty right hand member, this rule is not appended to \( L_j \), but again the deletion process must be repeated for the left hand member symbol of such a rule. This process must terminate.

The same procedure is to be followed for each rule \( R_j \), the set of auxiliary rules thus produced comprising the updated list \( L_j \). At this point one occurrence of each member of \( L_j \) not included among the current augmented grammar rules is to be added to those rules, giving a new current augmented grammar.

Several important refinements of the above procedure can be made to eliminate certain auxiliary rules that do not reflect possible structure. Consider a member of the set \( U_{ij} \) that denotes two or more elementary changes, say
\[ A_1 > X_1, A_2 > X_2, \ldots, A_m > X_m, \quad (m \geq 2), \]
and let us suppose without loss of generality that \( A_p \) appears to the left of \( A_q \) in the structural index of \( \tau_i \) if \( p < q \). Then it is clear that each corresponding \( A_p \) of \( R_j \) must precede \( A_q \) in that \( R_j \) if \( p < q \) for \( \tau_i \) applies only to an ordered sequence of trees that satisfy its structural index. If the relative order of the \( A_p \) is not the same in both \( R_j \) and the structural index of \( \tau_i \) then the auxiliary rule(s) corresponding to application of this member of \( U_{ij} \) to \( R_j \) need not be produced because the structure thus reflected could not be produced by \( \tau_i \). We will refer to this requirement as the joint ordering condition.

A further refinement in eliminating rules is also possible. Let
A_p be a term of the structural index of \( \tau_1 \), and suppose that the CF rule \( R_j \) in question is \( A_1 \rightarrow A_2 A_3 \ldots A_m A_p A_{m+1} A_{m+2} \ldots A_n \).

We consider the question of whether or not an elementary change \( A_p > X_p \) can be applied to \( R_j \) to give possible derived constituent structure. Now the structural index of \( \tau_1 \) is of the form \((M_q, M_{-q+1}, \ldots, M_2, M_1, A_p, M_1, M_2, \ldots, M_u)\).

If \( \tau_1 \) is to apply to a tree containing the structure given by \( R_j \) then clearly the \( A_{m+1}, A_{m+2}, \ldots, A_n \) must either satisfy structural index terms \((M_1, M_2, \ldots)\) or there must be derivations of the \( A_{m+k} \) which satisfy successive terms of \((M_1, M_2, \ldots)\).

For example, we might have the structure
where \( u \geq k \).

In order to determine if there exist \( A_{m+k} \)-derivations satisfying initial subsequences of \( (M_1, \ldots, M_u) \) it is only necessary to build structure in all possible ways from initial substrings of \( (M_1, \ldots, M_u) \) up to \( A_{m+k} \) (for \( k = 1, 2, \ldots, (n-m) \)) where \( M_1, \ldots, M_{i-1} \) were previously satisfied by \( A_{m+1}, A_{m+2}, \ldots, A_{m+k-1} \). This is efficiently done by the function PROCES which was discussed in Section 3.4.

In a manner completely analogous to our treatment of \( A_{m+1} A_{m+2} \cdots A_n \) with respect to the structural index terms \( M_1, M_2, \ldots, M_u \) we can impose conditions on \( A_2 A_3 \cdots A_m \) with respect to terms \( M_{-q}, M_{-q+1}, \ldots, M_{-2}, M_{-1} \). The only difference in this case is that the matching procedure works from right to left instead of left to right. If inverted sequences \( M_{-1}, M_{-2}, \ldots, M_{-q} \) and \( A_{m} A_{m-1} \cdots A_2 \) are used, and if, furthermore, CF recognition is performed using a grammar obtained from the augmented grammar by replacing every rule \( B_1 \to B_2 B_3 \cdots B_t \) by the corresponding rule \( B_1 \to B_t B_{t-1} \cdots B_2 \), then the function PROCES can again be used.

This requirement, which we will refer to as the individual applicability condition, can be extended to the case where one of the structural index terms \( M_i \) is the rule schema symbol \( X \). We merely insist that there be an \( A_{m+k} \)-derivation satisfying a sequence of terms \( M_i \) up to but not including the first occurrence of \( X \). This is accomplished by changing the function PROCES to build structure from the top down rather than from the bottom up, say by using the STB Algorithm of Reference 13 rather than the SBT Algorithm.
Still another refinement in eliminating rules is the following:
Suppose in analyzing the string $A_{m+1} A_{m+2} \ldots A_n$ into a sequence of
trees satisfying $(M_1, M_2, \ldots, M_r)$ it is discovered that the shortest
such sequence $(M_1, M_2, \ldots, M_r)$ includes one or more terms with which
are associated other elementary structural changes. Then it must
be the case that each of those elementary structural changes must
be jointly applied with the change $A_p > x_p$. We will refer to this
requirement as the mutual dependence condition.

Finally, we note that if all of the structural index terms $M_i$
are not exhausted in satisfying the individual applicability condition
for some term $A_p$, elementary change $A_p > x_p$, and rule $A_1 \rightarrow A_2 \ldots A_m
A_p \ A_{m+1} \ \ldots \ A_n$, then it must be possible to analyze the remaining
extremal terms $M_i$, with $A_1$ inserted in place of those terms which
were used, as a tree dominated by the sentence symbol $S^1$. We will
refer to this requirement as the higher structure condition, and
we note that it is complicated by occurrences of the symbol $X$ for
precisely the same reason as was the procedure suggested by Matthews
which we discussed above. It is, of course, possible to ignore
this condition for transformations containing $X$, thus avoiding
the resultant complications at the price of generating certain
unnecessary auxiliary rules.

To illustrate the various auxiliary rule generating procedures
presented in this section let us find the auxiliary rules reflecting
derived structure produced by applying the PASSIVE transformation
($\text{OBLIG, (NP AUX V NP BY PASS), (4 (2 BE EN) 3 0 5 1), ( ), PASSIVE}$)
to the base component:
$S \rightarrow \# S^1 \#$

$COMP \rightarrow S^1 \#$

$S^1 \rightarrow NP \ AUX \ VP$

$AUX \rightarrow AUXA \ AUXB$

$AUXA \rightarrow AUXA$

$AUXA \rightarrow TNS$

$AUXA \rightarrow TNS M$

$TNS \rightarrow PRES$

$TNS \rightarrow PAST$

$AUXB \rightarrow HAVE \ EN \ BE \ ING$

$AUXB \rightarrow HAVE \ EN$

$AUXB \rightarrow BE \ ING$

$VP \rightarrow V \ NP \ BY \ PASS$
VP → BE PRED
VP → V NP
VP → V
PRED → NP
PRED → ADJ
PRED → LOC
LOC → PREP NP
LOC → HERE
NP → DET N
NP → DET N COMP
DET → DEF
DET → INDEF
We first make use of the procedure suggested by Matthews. We perform a CF recognition of the structural index of PASSIVE, (NP, AUX, V, NP, BY PASS) to see if it can be analyzed as a tree dominated by $S^1$. The only such tree is found to be

$$
S^1 \rightarrow \begin{array}{c}
\text{NP} \\
\text{AUX} \\
\text{VP}\end{array}
\begin{array}{c}
\text{V} \\
\text{NP} \\
\text{BY} \\
\text{PASS}\end{array}
$$

If we transform this tree using PASSIVE, we obtain

$$
S^1 \rightarrow \begin{array}{c}
\text{NP} \\
\text{AUX} \\
\text{BE} \\
\text{EN} \\
\text{VP}\end{array}
\begin{array}{c}
\text{V} \\
\text{BY} \\
\text{NP}\end{array}
$$

This structure contains the two new auxiliary rules $S^1 \rightarrow \text{NP AUX BE EN VP}$ and $\text{VP} \rightarrow \text{V BY NP}$.

Considering this same problem of auxiliary rule generation using our alternative procedure, we have the elementary structural changes:

1. $\text{AUX} > \text{AUX BE EN}$
2. $\text{NP} > \phi$
3. $\text{PASS} > \text{NP}$

We consider the effect of changes 1-3 on the given CF grammar, each rule being considered in sequence. The first rule containing either AUX, NP, or PASS is the rule $S^1 \rightarrow \text{NP AUX VP}$. If we apply our crude
procedure, the following auxiliary rules are produced:

\[ S^1 \rightarrow NP \text{ AUX BE EN VP} \]

\[ S^1 \rightarrow AUX VP \]

\[ S^1 \rightarrow AUX BE EN VP \]

If, however, we make use of the refinements previously discussed, the last two rules are eliminated. The last rule is excluded because of the joint ordering condition; NP precedes AUX in the rule \( S^1 \rightarrow NP \text{ AUX VP} \) but the NP which is deleted follows AUX in the structural index of PASSIVE. Furthermore, change 2 is not possible at all either alone or in combination with other changes because of the individual applicability condition. In order that the symbol NP of the structure

\[
\begin{array}{c}
S^1 \\
\hline
NP & AUX & VP
\end{array}
\]

be deletable this NP must satisfy the underlined term of the structural index (NP, AUX, V, NP, BY, PASS). This, in turn, is possible only if the remaining structural index terms BY, PASS can be analyzed as a sequence of trees that are dominated by an initial subsequence of the constituents AUX VP. There exists no structure for which AUX dominates a tree containing BY, and, hence, elementary change 2 cannot apply to \( S^1 \rightarrow NP \text{ AUX VP} \). This eliminates both \( S^1 \rightarrow AUX VP \) and \( S^1 \rightarrow AUX BE EN VP \) as auxiliary rules.

The next CF rule containing AUX, NP, or PASS is VP \( \rightarrow V \text{ NP BY PASS} \).

Applying changes 2 and 3, both alone and jointly, the crude procedure gives:
VP → V BY NP
VP → V BY PASS
VP → V NP BY NP

These changes satisfy the joint ordering condition and the individual applicability condition but by the mutual dependence condition, elementary change 2 requires change 3 and vice versa. This follows because the individual applicability condition is satisfied only by matching the V, BY, NP, and PASS of the structural index with the identical constituents of the rule VP → V NP BY PASS, and PASS is required to satisfy the individual applicability condition for NP just as NP is required to satisfy this condition for PASS.

Making use of the mutual dependence condition then, only the auxiliary rule VP → V BY NP must be retained.

The rule VP → V NP undergoes elementary change 2, but the resulting rule VP → V is already a rule and so is not duplicated. Change 2 is also applicable to PRED → NP. If the higher structure condition is ignored, PRED must be deleted from the right member of rules in which it occurs. This gives the additional auxiliary rule VP → BE.

However, if we make use of the higher structure rule we see that change 2 need not be applied to the rule PRED → NP because the string NP, AUX, V, PRED, BY, PASS cannot be analyzed as an S₁.

The only remaining rule that could possibly be a candidate for alteration by the PASSIVE transformation is LOC → PREP NP. Change 2 does not meet the individual applicability condition, however, because PREP, the constituent to the left of NP, neither matches V, the structural index term to the left of NP, nor can PREP
dominate this index term V. Hence, no auxiliary rule is produced.

Collecting the auxiliary rules that remain after application of all four conditions we see that they consist only of $S^1 \rightarrow NP$ AUX BE EN VP and VP $\rightarrow$ V BY NP, the same rules that were produced by the procedure suggested by Matthews.

3.7 EXTENSIONS TO MODIFIED GRAMMARS

In the preceding sections of this chapter we have presented an analysis procedure for the specific type of transformational grammar defined in Chapter II. More precisely, we have presented an algorithm for determining a set of structures from a given sentence $S$ and a transformational grammar $G$ that includes all of the structural descriptions assigned by $G$ to $S$. Hence, by performing a synthesis on each of these structures, we can determine that subset of this set of structures which is precisely the set of all structural descriptions of $S$ with respect to $G$.

In this section we consider two modifications to the type of transformational grammar of Chapter II, and we modify our basic recognition procedure to make it valid for each of these modifications.

We first consider a transformational grammar which differs from our model only in its use of a context-sensitive rather than context-free base component. By context-sensitive we mean that only a single symbol is rewritten in a single rule, but this symbol may be restricted to a given environment. If those environments are discarded, a context-free grammar is obtained. Obviously, the set of base structures generated by this context-free grammar includes all of the base structures generated by the context-sensitive grammar from which
this context-free grammar was obtained. Hence, if we perform our recognition procedure as before, using the associated context-free grammar, it is only necessary to check structural descriptions thus obtained to verify that they satisfy the context-sensitive restrictions in order to find the set of all structural descriptions assigned to a sentence by a given transformational grammar with a context-sensitive base component.

The other modification to our basic transformational grammar we consider is to permit the use of so-called initial singulary transformations. As the name implies, these are ordered singulary transformations that are not in the generative cycle but rather are applied just once to each level of embedding prior to application of the transformational cycle in the usual way. It is clear that nothing significant is changed if the application of the initial singulary transformations to a level L is delayed until just before the application of a binary transformation to level L and the level immediately below L in the course of the transformational cycle. Reversing this order of applying transformations in order to determine the order of applying inverse transformations in recognition, we see that inverse initial singulary transformations apply, in reverse order, to the matrix sentence continuations of inverse binary transformations.

In view of these observations we conclude that the recognition procedures of this chapter are applicable to a grammar containing initial singulary transformations without making any modification except to order the inverse singulary transformations properly within the inverse transformational cycle. This order must begin
with the inverse singular transformations of the transformational cycle in reverse order, must continue with the binary transformations (also in reverse order if they are ordered), and must conclude with the inverse initial singular transformations in reverse order. The function SETTRANS of Appendix D, which computes the inverse transformations, orders them in this required way.

3.8 REASONS FOR A SYNTHESIS PHASE

We have already discussed several reasons why a final synthesis phase is required by certain of the recognition procedure variants of this chapter. In this section we briefly recapitulate these and mention one reason so far not considered.

All of the methods of this chapter require a final check on the phrase structure if the base component is a context-sensitive grammar and it is replaced by a context-free grammar as is discussed in Section 3.7. This final check need not be performed in its entirety at the very end of the analysis phase, however. As the structure of each complete constituent structure tree is determined and this tree is embedded in a matrix sentence continuation, a check can be made to insure that the structure in question is compatible with any context-sensitive base component rules.

The simplified procedure of Section 3.3 makes no use of auxiliary rules but, instead, groups terminal symbols in all possible ways. The final structures obtained are all given by the base component CF grammar, but a synthesis phase is still needed because there is no guaranty that these may be transformed into any sentence, let alone
the given sentence.

The procedures of Sections 3.4 and 3.5 make use of auxiliary rules, and hence they both require that the final structure be checked to make sure that no structure given by an auxiliary rule is included.

The procedures of Sections 3.3 and 3.4 make no provision for a check that conditions are met on structure above a proper analysis of a tree with respect to some transformation. Likewise, they can produce a structure that is not a valid structural description because at least one applicable obligatory transformation is not performed in the sequence of transformations that are supposed to map this structure into a sentence of the language. Consequently, both the procedures of Sections 3.3 and 3.4 require a final synthesis. The method of 3.5, as we have already pointed out, can directly incorporate these conditions.

Finally, there is still another source of incorrect structural descriptions being found. If a transformation reduplicates some tree, then the corresponding inverse transformation will repeat the symbol in question in its inverse structural index. This reduplicated term of the inverse structural index must, of course, denote the same tree, but there is nothing in the operation of inverse transformations as we have defined them to demand that the trees satisfying reduplicated symbols must be identical. A final synthesis would discard any erroneous structural descriptions thus obtained, but alternatively we could add a condition to inverse transformations to insure the equality of trees satisfying re-duplicated symbols. We have not incorporated such a condition into
our inverse transformations, and for this reason we would incorrectly assign to such strings as * a strong man is here wasn't a yellow house the same structure as a strong man is here isn't he if the recognition was carried out with respect to a grammar containing the tagquestion transformation of Grammar 1 in Appendix B.
4.1 COMPUTER IMPLEMENTATION

The recognition procedure presented in Section 3.4, with the exception of the algorithm for mechanically producing auxiliary context-free rules and the final synthesis phase, has been implemented by means of a LISP program\(^{24}\). This program has been used primarily on the AFCRL UNIVAC M-460 Computer, but a version for the IBM 7090 computer is also available.

As is usually the case in programming long, complex algorithms, the details of the procedure programmed (as they are reflected by the description in Chapter III) were not worked out completely prior to beginning the programming. Indeed, the recognition of several hundred sentences with respect to perhaps twenty to thirty different transformational grammars (many closely related) in the course of debugging the LISP program disclose many oversights requiring modification of the underlying recognition procedure. Documentation of the various LISP programs and the results obtained from them was complicated not only by the innumerable modifications of the underlying recognition procedure that have proved necessary, but also was complicated by numerous substitutions of one group of LISP functions for another to increase programming efficiency, substitutions representing different implementation of details too minute to require discussion in the body of this thesis. We have, nevertheless, called a temporary halt to our efforts to produce a
transformational recognition program that is general, reasonably efficient, and convenient for a linguist to use, and the UNIVAC M-460 version of our program at this juncture is documented as Appendix D. Appendices B and C contain details of operating instructions and input/output formats. In the present section we will limit ourselves to general remarks about the operating program, and in the next section we will discuss the use we have made of this program in recognizing specific grammars.

The program was originally written as a collection of fifty-six LISP functions that compiled into about 4000 words of computer storage. The precedence table and its associated look up functions required for context-free grammar recognition were eventually replaced by assembly language-coded subroutines for the UNIVAC M-460 Computer, and the entire context-free recognition procedure was replaced by an assembly language-coded program for the IBM 7090. In its present form, as documented in Appendix D, five functions accept generative transformations typed in a convenient format and produce corresponding inverse transformations. Nine functions set up internal tables for a given grammar that are subsequently used in context-free recognition, and another ten functions are used for performing context-free recognition. Another twenty-two functions are concerned with the transformational aspects of recognition.

Speed of computation can at best be estimated for several reasons. Virtually all computer runs were made on the UNIVAC M-460 Computer. Runs were submitted to the IBM 7090 Computer only to
produce a 7090 version of the already operational M-460 program. In running on the M-460 Computer specific interest in one or more functions usually necessitated running those functions interpreted rather than compiled in order to trace them and thus obtain extra printouts. The sixty-to-one price paid for running each function that was interpreted rather than compiled slowed the program down overall by a factor of about three. Moreover, the ten character per second flexowriter printouts added substantially to the time required. In addition to tracing different functions from run to run in order to verify correct functioning of certain sub-procedures or to determine the computations on which most time was spent, certain functions were invariably traced to obtain a continuous record of the inverse transformations performed and the structures produced during the course of the analysis. This added extra time but supplied results that were valuable in finding programming mistakes and errors in writing the transformational grammar. These results also suggested worthwhile programming modifications and modifications to the underlying recognition procedure. Another factor that added to computing time is that modified functions were not always immediately recompiled. Instead, they were frequently run interpreted for several weeks until a number of such functions had accumulated and then the entire batch of modified functions was compiled.

For all these reasons we can only say that the time required to recognize a single sentence varied from fifteen seconds to more
than two hours on the M-460 Computer. This depended to a large extent, of course, on the size of the grammar. The trivial little test grammar I included in Appendix B of six transformations and eleven context-free rules required, for example, only 15 to 30 seconds for recognition of the sentences it defines. The number of transformations alone, however, seems to be not as significant a factor in determining the time required to recognize a sentence as is the complexity of the sentence in terms of the embedded structure it contains. To illustrate this, we include in Appendix C the recognition of a sentence with respect to a grammar of only two transformations (a singular number transformation and a binary relative clause transformation) written by J. Keyser. This sentence required from 20 minutes to 70 minutes to be recognized using several different program variants.

The M-460 Computer does have a millisecond clock accessible to the LISP system, and, hence, it was possible to separately determine the time spent in certain parts of the recognition procedure. Running with the context-free recognition functions completely compiled but with various other functions often not compiled, it was found that from one-fourth to one-third of the total time was spent in context-free recognition. Of this context-free recognition time, one-third to one-half was spent in actually performing the necessary parsing to obtain a set of trees represented as pushdown store automaton acceptance sequence, and the remaining time was spent in converting those representations of trees to ordinary LISP S-expression representations of trees.
Relatively little is known about the speed of the IBM 7090 version of the program as it has not been run on any sentences containing more than a single embedded sentence within a main sentence. Short sentences were used merely to get the program running, and no extended production runs have been made. The basic speed of the 7090 is about five times as fast as the M-460, and the machine language coding of the context-free language recognition component makes the overall program about one and one half times faster. These factors, when combined with compiling all of the LISP functions, gave a resulting 7090 program that was about twenty times as fast as the M-460 program which we principally used in recognizing sentences with respect to various grammars. One sentence that took eight seconds on the 7090, for example, required 165 seconds on the M-460, 64 seconds of which was required for performing context-free grammar recognition. The ratio between 7090 time and corresponding M-460 time becomes much larger if we require any appreciable amount of supplementary printing. This is due to the 7090's use of magnetic tape and off-line printing rather than on-line flexowriter printouts.

A late change that has been made to the program is the replacement of one algorithm for finding all proper analyses of a tree that satisfy a given structural index by another algorithm. This modification, which is discussed in Appendix A, has approximately halved the overall time required, allowing us to estimate that the speed of the present 7090 program is about forty times as fast as the M-460 program which has been our principal tool in considering various grammars and sentences.
The LISP programming language, and in particular the AFCRL M-460 LISP system, proved extremely convenient not only for initial programming but also for debugging, for investigating the effect of modifications on the speed of the program, and for experimentally investigating that grammars work as intended and do not assign unwanted structural descriptions. LISP itself is basically well-suited for this type of programming because the data structures to which it addresses itself (S-expressions) are readily interpreted as trees. Hence, for most required functions, compiled LISP programs are probably quite efficient. There are, of course, a few exceptions, and it was for this reason that the portion of the program concerned with context-free recognition was coded in assembly language.

4.2 USAGE OF RECOGNITION PROGRAM

In Appendix C we give detailed explanations of the recognition of several specific sentences. We content ourselves in this section with a general account of how the existing program can be used in "debugging" large transformational grammars, and with a few observations derived from using that program.

As the program now stands, computing time requirements preclude its use in such information processing applications as information retrieval, machine translation, and compiler implementation. Even in its present state, however, the program is of use to a linguist who wishes to carefully investigate a transformational grammar he has written to insure that it both generates sentences it was intended to generate and assigns adequate structure to those sentences without also assigning structural descriptions that do not correspond to distinct interpretations. With a recognition program available,
a working grammar is most quickly obtained by following these steps:

(1) Make sure each transformation works as intended by generating a few sentences whose derivations make use of that transformation. There is no substitute for actually producing the sequence of trees in a derivation, because errors are invariably made when many steps are done at once mentally. A synthesis program is not only required to eliminate spurious structural descriptions, but it is also useful in expediting this step.

(2) Test a large transformational grammar by making up a number of smaller grammars, each containing an inter-related set of transformations that by themselves produce certain sentences. Insure that these sub-grammars each work as intended by using the recognition program.

(3) After completion of both the preceding steps, recognize sentences with respect to the whole transformational grammar to insure that no unacceptable structural descriptions are produced by unexpected relationships between transformations in different sub-grammars.

By following these steps not only is machine time saved, but also trivial errors are usually localized and then pinpointed more easily than if sentences had been recognized with respect to a large grammar directly. When this is attempted it is usually the case that a host of sentences presumed grammatical are assigned no
structural descriptions. This calls for performance of steps (1) and (2) which could more profitably have been done in the first place. It is unfortunately the case that in putting together a system of rules as complex as those of a transformational grammar, mistakes are inevitable. The situation is not much different in this respect from the problems involved in debugging a large computer program. Binary transformations especially are a potential source of difficulty. As a result of writing and testing our own binary transformations and also as a result of carefully studying binary transformations formulated by others, we can certify that a common error is for these transformations to work for only a single level of embedding, falling to apply correctly when the constituent sentence has been produced by a previous application of the same transformation.

It should not be concluded from the stress we have placed on the difficulties of precisely writing and investigating a transformational grammar that any inherent theoretical deficiencies in transformational grammars have been observed. On the contrary, once the shortsighted errors and omissions have been eliminated, it has been found that the resulting grammar is free of the host of unwanted structural descriptions that have plagued proponents of context-free grammars \cite{24,25}. Although we have run across unexpected structural descriptions, in every case it was possible to associate a distinct interpretation with each structural description. The reason we could do this seems to be that in a
transformational structural description there are only a small number of primitive relationships representable, such relationships as subject-object, negation, interrogative indication, and qualification of manner or time. The semantic content of complex sentences is represented by a set of such primitive relationships that are themselves interrelated by the embedding structure of a structural description. Conversely, because we have limited ourselves to a small set of primitives it is possible to give an interpretation to every structural description assigned by a transformational grammar. The extent to which these interpretations coincide with the readings a native speaker gets from a sentence can only be determined by writing grammars and finding all of the structural descriptions assigned by those grammars to certain sentences. A mechanical recognition procedure is a valuable tool in carrying out this work.

To illustrate these comments more concretely, we consider Grammar 3A of Appendix B. The sentence the boy was examined by the doctor is assigned two structural descriptions. The first denotes the usual meaning, an interpretation represented by the structural description

```plaintext
Transformational structural description there are only a small number of primitive relationships representable, such relationships as subject-object, negation, interrogative indication, and qualification of manner or time. The semantic content of complex sentences is represented by a set of such primitive relationships that are themselves interrelated by the embedding structure of a structural description. Conversely, because we have limited ourselves to a small set of primitives it is possible to give an interpretation to every structural description assigned by a transformational grammar. The extent to which these interpretations coincide with the readings a native speaker gets from a sentence can only be determined by writing grammars and finding all of the structural descriptions assigned by those grammars to certain sentences. A mechanical recognition procedure is a valuable tool in carrying out this work.

To illustrate these comments more concretely, we consider Grammar 3A of Appendix B. The sentence the boy was examined by the doctor is assigned two structural descriptions. The first denotes the usual meaning, an interpretation represented by the structural description

```
The second, denoting that "someone examined the boy over there by the doctor," is represented by

This second structural description and its associated meaning were not anticipated prior to computer running, but the tree was readily interpreted as the indicated acceptable interpretation. We observe that this grammar always identifies the construction "BY NP" as a locative, assigning to such sentences as the party was enjoyed by all the guests the interpretation, "Someone enjoyed the party over there by all the guests." This is undoubtedly an unwanted interpretation, but it merely points out the obvious inadequacy of the grammar in question in its generation of locatives. If by all the guests is not wanted as a valid locative then the grammar must be refined in its treatment of locatives so as to exclude it.
A slightly more complicated example is provided by the sentence
John likes the old book on the table. The same grammar previously
considered assigns four structural descriptions to this sentence,
namely, those shown in figures 1 through 4.
Figure 1.

STRUCTURAL DESCRIPTION (1)
Figure 2.

STRUCTURAL DESCRIPTION (2)
Figure 3.

STRUCTURAL DESCRIPTION (3)
Figure 4.

S

NP | AUX | VP

NPOP | AUXA | V | NP | ADV

JOHN | TNS | LIKE | DET | N | COMP | LOC

PRES | DEF | BOOK | # | S' | # | PREP | NP

THE | NP | AUX | VP | ON | DET | N

DET | N | AUXA | BE | PRED | DEF | TABLE

DEF | BOOK | TNS | ADJ | THE

THE | PRES | OLD

STRUCTURAL DESCRIPTION (4)
Corresponding to these four structural descriptions we have the four interpretations:

(1) John likes the old book (that is) on the table (not the new book that is on the table).

(2) John likes the old book (that is) on the table (not the old book that is in the bookcase).

(3) John likes the old book (to be) on the table (not to be in the book case).

(4) John likes the old book (when he is) on the table (not when he is in bed).

These four structural descriptions and corresponding meanings were not expected prior to machine recognition. Instead, one structural description was expected to indicate the meaning

(5) "John likes the book that is old and that is on the table" and another expected structural description was the one given by (3).

Number (4) was unexpected as was the presence of (1) and (2) expressing differently stressed variants of meaning (5). It is clear on reflection that the grammar in question, having no coordination, cannot produce a structure corresponding to (5).

Neither (1) nor (2) is available to denote this meaning because of their contrast to each other. We do not claim that the grammar in question is necessarily "correct" in any sense; the decision about whether both (1) and (2) are necessary depends upon whether this distinction is one the linguist wishes his grammar to reflect. We do claim, however, that a mechanical recognition program can be
a valuable tool to the linguist in determining those semantic
distinctions that his grammars make.

As a final remark on this sentence we note that if (as was
originally run) the context-free component contains the rule VP→BE
PRED ADV then we get the additional structural description

\[
\begin{align*}
S' & \quad \leftarrow \quad \text{VP} \\
\quad \leftarrow \quad \text{NP} \\
\quad \leftarrow \quad \text{NPROP} \\
\quad \leftarrow \quad \text{AUX} \\
\quad \leftarrow \quad \text{V} \\
\quad \leftarrow \quad \text{NP} \\
\quad \leftarrow \quad \text{TNS} \\
\quad \leftarrow \quad \text{LIKE} \\
\quad \leftarrow \quad \text{DET} \\
\quad \leftarrow \quad \text{COMP} \\
\quad \leftarrow \quad \text{DEF} \\
\quad \leftarrow \quad \text{BOOK} \\
\quad \leftarrow \quad \text{S'} \\
\quad \leftarrow \quad \text{N} \\
\quad \leftarrow \quad \text{AUX} \\
\quad \leftarrow \quad \text{V} \\
\quad \leftarrow \quad \text{NP} \\
\quad \leftarrow \quad \text{DET} \\
\quad \leftarrow \quad \text{N} \\
\quad \leftarrow \quad \text{AUX} \\
\quad \leftarrow \quad \text{BE} \\
\quad \leftarrow \quad \text{PRED} \\
\quad \leftarrow \quad \text{ADV} \\
\quad \leftarrow \quad \text{DEF} \\
\quad \leftarrow \quad \text{BOOK} \\
\quad \leftarrow \quad \text{TNS} \\
\quad \leftarrow \quad \text{ADJ} \\
\quad \leftarrow \quad \text{LOC} \\
\quad \leftarrow \quad \text{THE} \\
\quad \leftarrow \quad \text{PRES} \\
\quad \leftarrow \quad \text{OLD} \\
\quad \leftarrow \quad \text{PREP} \\
\quad \leftarrow \quad \text{NP} \\
\quad \leftarrow \quad \text{ON} \\
\quad \leftarrow \quad \text{DET} \\
\quad \leftarrow \quad \text{N} \\
\quad \leftarrow \quad \text{DEF} \\
\quad \leftarrow \quad \text{TABLE} \\
\quad \leftarrow \quad \text{THE}
\end{align*}
\]

With roughly the meaning* " John likes the book that is old over
there on the table". The rejection of this interpretation stems
from the unrestricted use of VP→BE PRED ADV and ADV→LOC
which produce such sentences as* The book is old on the table.\textsuperscript{27}
Our point is not that just any transformational grammar, by virtue of the fact that it is a transformational grammar, must assign structural descriptions that denote each distinct interpretation and nothing else. This is, of course, false, as linguistically inadequate transformational grammars are as easy to write as any other kind. A transformational grammar does provide, however, a principled procedure for relating meaning and structure, and thus a mechanical recognition program allows the linguist to determine all the "readings" assigned to a sentence by a given grammar; this gives him information valuable for determining how to modify the grammar if the readings implied by the grammar do not correspond to the set of intuitively acceptable interpretations of the sentence.

In all the work that was done on actually using the recognition program to investigate various grammars, we encountered only structural descriptions that either corresponded to distinct acceptable meanings or which indicated a basic inadequacy of the grammar in generating certain simple declarative sentences. Hence, to the extent they go, our limited efforts have disclosed none of the seemingly inherent problems of unwanted ambiguity that have plagued the proponents of context-free grammars of natural languages. A disclaimer must be made on this point, however. Until a procedure was devised to mechanically compute a sufficient set of auxiliary context-free rules there was no assurance that all structural descriptions were being found. One could only say that every structural description was found whose transformational derivation reflected structure that had been foreseen in providing auxiliary
rules. The only grammars considered for which auxiliary rules were algorithmically produced were Grammar 1 and Grammar 2B of Appendix B, and hence the above proviso is applicable to results obtained for all other grammars.

4.3 PERCEPTUAL MODEL CONSIDERATIONS

A final topic we discuss in this chapter is the role played by a recognition procedure as a model of perception. We say at the outset that our procedure was not motivated by a desire to account for linguistic perception, we make no claims that it does so, and we made no use of psycholinguistic considerations in devising our recognition procedure. Nevertheless, as we stated in justifying the study of transformational recognition procedures, psycholinguistic considerations are important in their own right, and we will at least discuss the facts that bear on the adequacy of our procedure and other procedures as models of perception.

We have already discussed the manner in which the transformational structural description is related to meaning. If we accept this view, then one aspect of the process of understanding a sentence is the necessity to obtain structural descriptions of that sentence. In evaluating whether a particular mechanical recognition procedure models certain aspects of human linguistic perception it is necessary to determine characteristics exhibited by the mechanical procedure and to investigate whether or not these characteristics are also exhibited by the human perceiver of language. Similarly, it is necessary to determine measurable characteristics of understanding.
a sentence that are exhibited by humans and to investigate whether or not a correspondence is found in machine recognition. We will consider an example of each of these evaluative procedures.

The recognition procedure we have presented pieces together any particular structural description by determining structure from the top kernel down through successive levels of embedding. It is reasonable to ask whether this essential order of our recognition procedure is matched by a corresponding temporal ordering of the process of understanding a sentence, i.e., do we understand relationships reflected in higher structure prior to understanding those reflected by lower levels of embedding? This might be tested by asking subjects questions about sentences whose complete comprehension is made difficult, perhaps by assigning extraneous simultaneous tasks. If a plausible method could be devised for stopping the process of understanding "in the middle" somehow, and if, in addition, biases related to position in the sentence could be determined and allowed for, then we could determine whether humans exhibit the same temporal recognition characteristics as our transformational recognition procedure. We have conducted no such experiments. Bever [personal communication] reports that in a few preliminary experiments, he has observed a slight tendency to recall higher level information over lower level information, but some exceptions were noted. For one thing noun phrases containing an adjective modifier were perceived as a unit, and the adjective, which reflects lower level structure, was seldom
omitted without concomitant omission of the entire noun phrase. It might be interesting to see if such exceptions to the general higher-to-lower temporal ordering of structure determination correspond to structure given by the auxiliary rules. This would give even these rules some psycholinguistic justification. We are, of course, just speculating and suggesting relevant experiments.

Conversely looking for human recognition characteristics, we recall a recent paper by Miller and Isard [35]. They compare the intrinsic relative difficulty of subjects to understand such pairs of sentences as

(1) The man that said that the rat was killed by a cat that was chased by the dog is a liar.

(2) The man that said that a cat that the dog chased killed the rat is a liar.

They attempt to explain the relative ease of understanding the first sentence in terms of various properties of the surface structure of these two sentences. We would, instead, investigate the relative difficulty experienced by our recognition procedure in obtaining structural descriptions of these two sentences. To do this we wrote a grammar that generated sentences of this type, assigning to the first the structural description shown in Figure 5
Figure 5.
and the second the same structure save for omission of the two occurrences of BY PASS. Unfortunately, results depend upon the particular grammar utilized, but for each two grammars considered, the second sentence was recognized more quickly than was the first sentence. For one grammar the first sentence took twice as much time to be recognized as the second sentence, and for the other grammar the factor was one and one-half times. Hence, our procedure does not reflect human difficulty of understanding sentences, at least from the standpoint of computational time required. Other measures such as storage requirements are, of course, also possible. Finally, we observe that it may well be the case that people don't find all structural descriptions and then select one on some basis. Rather, they, might find one and see if it "fits" the situation, going on to find another only if this structural description is rejected as a possible meaning by a semantic component. If this were the case, we should look at the time required to find particular structural descriptions rather that the set of all structural descriptions of a sentence.

Of course the extent to which our recognition procedure reflects linguistic perception can also be investigated with respect to other observed aspects of linguistic perception. We merely picked one aspect for the purpose of illustration.

We repeat in closing that our purpose in discussing the role our procedure might play as a perceptual model was not to make any claim as to its adequacy. We were instead concerned merely
with assessing factors that might be relevant in evaluating the adequacy of our procedure as a perceptual model and perhaps the adequacy of other models which have been proposed or will be proposed.
CHAPTER V: SUMMARY

In this thesis we have: (1) considered the reasons for devising ways of finding all structural descriptions assigned to a sentence by a transformational grammar, (2) surveyed the procedures that have been proposed for finding those structural descriptions, (3) precisely defined a class of transformational grammars and their associated deep-structure structural descriptions, (4) presented three preliminary analysis procedures which, when used in an analysis-by-synthesis procedure, constitute recognition procedures for this class of transformational grammars, (5) discussed a LISP computer program implementation of two of those preliminary analysis procedures, (6) commented on the utility of this program to a linguist, furnishing illustrative examples of sentences recognized, and (7) considered the role a recognition procedure plays in describing how sentences are understood.

The investigations carried out in conjunction with this thesis do not reflect the consideration of an isolated little problem, completely solved and no longer of interest. Instead, our work has shown the need for many further studies branching off in several directions, many of which we hope to pursue ourselves. For one thing, comparative studies of the relative efficiencies of the procedures of Sections 3.4 and 3.5 are needed. Each of these methods has several variants, the efficiency of which also require comparative evaluation with respect to grammars of

111
linguistic interest. Typical of the questions to be considered is whether in the method of Section 3.4 enough time is saved by aborting certain continuations (i.e., those which lead from a sequence of trees that cannot be recognized with respect to the augmented CF grammar as an $s^i$) to justify the computation time required to determine that these continuations may be terminated.

We plan to take steps designed to speed up the recognizer and thus make its use more economically feasible. This has not been a problem to the author because of the easy availability of the AFRL M-460 Computer. Extensive printouts were of interest to us in our use of the program in investigating various grammars, and the computer produced those printouts faster than we could follow and digest them. Under usual computer operating conditions, however, the time required, even by the IBM 7090, may be excessive for large grammars and long sentences; the reprogramming of certain functions may be required and refinements of some basic aspects of the program may be in order. As one such example, the present practice of using a fixed context-free grammar (the augmented grammar) throughout the course of the recognition of a sentence may have to be replaced by a procedure in which the context-free grammar is dynamically modified. There is no necessity to make use of auxiliary rules in attempting to build a matrix sentence continuation up to the sentence symbol, but this effect is now rather wastefully achieved by using these rules and subsequently discarding any structural description reflecting the use of an auxiliary rule. Another refinement that would be possible if the context-free component could be dynamically
altered is that auxiliary rules incorporated in the context-free grammar could reflect the structure possible at each level of embedding. As recognition proceeded to each successively deeper level of embedding, extra auxiliary rules could be discarded. This capacity to parse with a changing context-free grammar is not, however, as simple a change as might be supposed. As each change is made the precedence relations must either be entirely recomputed or else certain relationships must be changed and others added.

Independent of our efforts to speed up our program and to make it more convenient, we hope to make use of our grammar in the investigation of more comprehensive grammars than we have so far been able to consider. The author's planned participation as a user of a very large storage capacity, experimental LISP system should offer an excellent opportunity to consider some sizeable grammars. We also hope to interest other linguists in using both this contemplated system and the presently existing M-460 and 7090 programs. To this end we plan to integrate a version of our program into the MIT time-sharing system and to document a user's manual relative to the IBM 7090 program.

In conjunction with the investigation of large transformational grammars it may prove wise to forsake our insistence upon retaining a recognition program equally applicable to a class of grammars. In order to achieve greater speed it may prove necessary to perform a preliminary analysis to initially exclude certain transformations from consideration. It may also be possible to make use of certain
mutual dependencies on transformations such as either requiring or excluding one transformation if the application of another is assumed.

We will probably not pursue further any psycholinguistic investigations, but we may cooperate with linguists who wish to pursue such a studies making use of a mechanical recognition procedure.

One of the most pressing opportunities and challenges afforded by the existence of a programmed recognition procedure is the problem of mapping transformational structural descriptions into appropriate computer code or into another higher level language, either natural or artificial. This is important not merely because of the obvious practical implications, but also because this appears to be a rational approach to the study of semantics. In order to make practical use of the shades of meaning that can be reflected in the structural descriptions of a transformational grammar, procedures must be developed which cannot help but contribute to an area in which much original basic work is necessary.

In assessing the significance of the work reported in this thesis we note that the program produced is the first implementation of a recognition procedure for transformational grammars. Although that program is presently too slow to permit its use in a practical information processing system, it is a presently useful tool in writing and debugging grammars. It is hoped that contemplated improvements to the program will increase its usefulness as a research tool in linguistics.
APPENDIX A

COMPUTATION OF THE PROPER ANALYSES OF A TREE

A recurrent computational problem encountered in programming the recognition procedure of this thesis is the task of finding all of the proper analyses of a tree or, more frequently, all of the proper analyses satisfying a given structural index. Suppose we draw a curve through a tree from left to right in such a way that we neither cross a solid line nor bypass a terminal symbol by going under it, passing our curve, instead, only through the lettered nodes (including the terminal nodes). Then from the definition of proper analysis given in Chapter II it is clear that the subtrees dominated by the sequence of nodes through which such a curve passes constitute a proper analysis of the given tree. Moreover, to every proper analysis there corresponds a unique curve of this type through the tree.

Now to enumerate the set of possible curves through a tree, assign to each node a distinct literal and define a path to be a sequence of these literals passed through in traversing the structure from the top-most symbol downward to some terminal symbol. It is clear that a left to right curve of the type we seek must cross some node of every path. Let the literal assigned to each node represent the proposition that a given curve passes through that node. Denote by $A_{ij}$ the literal assigned to the $j$th node of the $i$th path, and consider the truth function

$$F = (A_{11} \lor A_{12} \lor \ldots) (A_{21} \lor A_{22} \lor \ldots) \ldots (A_{n1} \lor A_{n2} \lor \ldots).$$

115
Clearly, the prime implicants of $F$ denote precisely the set of curves we are seeking. Several efficient algorithms have been developed for the determination of the prime implicants of a function such as $F$ which is in conjunctive normal form [36]. Hence, by means of any of these algorithms it is possible to find all of the proper analyses of a given tree:

To illustrate this method of finding the proper analyses of a tree consider the following tree:

The associated truth function $F$, found by enumerating each path, is

$$F = (S \lor A \lor D) (S \lor A \lor E \lor I) (S \lor A \lor E \lor J)$$

$$\quad (S \lor A \lor E \lor K) (S \lor B \lor F \lor V \lor G) (S \lor B \lor F \lor V \lor H)$$

$$\quad (S \lor C \lor V \lor I) (S \lor C \lor V \lor M \lor V \lor O)$$

$$\quad (S \lor C \lor V \lor N \lor V \lor P) (S \lor C \lor V \lor N \lor V \lor Q)$$

and applying the algorithm of Reference 36 we obtain the alternation of prime implicants
\[ F = S \lor ABC \lor ABLMN \lor ABLMPQ \lor ABLOPQ \lor AFC \lor AFLMN \lor AFLMPQ \lor AFLON \lor AFLOPQ \lor AGHC \lor AGHLMN \lor AGHLMPQ \lor AGHLON \lor AGHLOPQ \lor DEBC \lor DEBIMN \lor DEBLMPQ \lor DEBLOPQ \lor DEFC \lor DEFLMN \lor DEFLMPQ \lor DEFLON \lor DEFLOR \lor DEGC \lor DEGHLMN \lor DEGHLMNPQ \lor DEGHLC \lor DEGHLMN \lor DEGHLMNPQ \lor DEGHLOPQ \lor DIJKBC \lor DIJKBIMN \lor DIJKBLMPQ \lor DIJKBLOPQ \lor DIJKFC \lor DIJKFLMN \lor DIJKFLMPQ \lor DIJKFLON \lor DIJKFLOPQ \lor DIJKGH \lor DIJKGHC \lor DIJKGHLMN \lor DIJKGHLMPQ \lor DIJKGHLON \lor DIJKGHOPLQ. \]

Each prime implicant denotes a proper analysis of the tree in question. To determine whether or not a tree can be analyzed in such a way as to satisfy a given structural index, it is only necessary to see whether the disjunction of the symbols in that index are included among the prime implicants of the function \( F \).

This algorithm was programmed and utilized in virtually all computer recognition considered to date. In seeking inefficient areas of the program, however, it was recently observed that rather than find all proper analyses and then see if any of them satisfy a given structural index, it might be more efficient to directly look for only those proper analyses satisfying the given index. The procedure programmed works roughly as follows on our sample tree: Suppose we wish to find all proper analyses satisfying the structural index \( (D, E, F, C) \). We will represent trees by corresponding LISP S-expressions. The given tree, for example, will be written as

\[
(S (A D (E I J K )) (B (F G H)) (C L (M O) (N P Q))).
\]

Starting with a sequence of trees consisting only of the given tree, we observe that the top node is not the first term \( D \) of the structural index. Hence, we remove the structure associated with the top-most node of the first tree of the current sequence of trees and thus obtain
\{(A \ D \ (E \ I \ J \ K)), \ (B \ (F \ G \ H)), \ (C \ L \ (M \ O) \ (N \ P \ Q))\}.

The first tree does not satisfy the first index term D, and hence we further remove the top structure of this tree obtaining
\{D, \ (E \ I \ J \ K), \ (B \ (F \ G \ H)), \ (C \ L \ (M \ O) \ (N \ P \ Q))\}.

Now the first tree (a degenerate tree to be sure) does satisfy the first index term D. Hence, we take this tree D to be the first term of the proper analysis we seek and go on to see if the remaining sequence of trees
\{(E \ I \ J \ K), \ (B \ (F \ G \ H)), \ (C \ L \ (M \ O) \ (N \ P \ Q))\} has a proper analysis satisfying the remaining index terms (E, F, C). The first tree satisfies the first term E so we take the tree (E I J K) to be the second tree of the proper analysis we seek and go on to see if
\{(B \ (F \ G \ H)), \ (C \ L \ (M \ O) \ (N \ P \ Q))\} has a proper analysis satisfying (F C). The first tree must be decomposed, giving
\{(F \ G \ H), \ (C \ L \ (M \ O) \ (N \ P \ Q))\}. The first of these trees satisfies the next index term F, and the second satisfies the remaining term C.

In our description we did not follow the continuations resulting from removing top structure even when the first tree satisfies the first index term. It is necessary to do this, however, in order to obtain all proper analyses satisfying a given structural index. For a precise description of this algorithm see the LISP function FIND\(^1\) listed in Appendix 5, whose first argument is a structural index and whose second argument is a tree.

Using the first algorithm we discussed, two minutes of M-460 were required to find all of the proper analyses of the sample tree we have been discussing. The function FIND\(^1\), however, took less than a second
to find the proper analysis satisfying any given structural index. Using this function FIND, the recognition of the last sentence of Appendix III took 24 minutes exclusive of printout time, whereas the time required using our first algorithm was about 60 minutes.
APPENDIX B

TRANSFORMATIONAL GRAMMARS RECOGNIZED

In this appendix we give several examples of grammars we have utilized in the mechanical recognition of various sentences. We illustrate our format and operating instructions in considering the first grammar presented below. The recognition of a few sentences generated by these grammars is discussed in Appendix C.

Several problems make it difficult to document in a meaningful way precisely what grammars and sentences were considered with respect to what recognition procedures. For one thing, the formats of the context-free rules and transformations changed many times. Not only were changes made for convenience but also to permit mechanical computation of inverse transformations and to reflect conditions on a grammar that were found to be necessary in order to obtain a recognition procedure. It was originally required that inverse transformations be computed by hand. Later, singularity transformations were allowed to be given as forward transformations, and much later, forward statement of binary transformations was allowed.

A more serious obstacle to documenting our experimental investigations of various grammars is related to the multiplicity of changes that have been made to the recognition procedure itself. Prior to obtaining an algorithm for finding all auxiliary rules that reflect potential derived structure for a sentence of length n or less, our recognition procedure went through several stages that were all variants of the method of Section 3.3. A universal grammar was

120
used to partition and assign a sequence of trees to nonterminals in programs that existed at this point, and, hence, these programs were in general quite slow, although fewer auxiliary rules had to be identified.

As a result of our constant modifications much was learned about how to improve the recognition procedure, but only a few grammars can be exhibited that conform to the conditions we presently require for recognition by means of our present program.

The first grammar we give is an up-dating of one of the first grammars we considered.

Grammer 1

Base Component:
S → NP VP
NP → DET N
NP → DET COMP N
DET → A
N → MAN
VP → V PRED
V → IS
PRED → ADV
PRED → ADJ
ADV → HERE
ADJ → STRONG

Transformational Component
(OPT (DET ((ADJ)) N V ADV)
  (O O THERE 4 (1 2 3 5)) ( ) THERE)

121
(OPT (NP V ANY)
  ((2 1) 0 3) ( ) INTERROG)

(OPT (ANY V ANY)
  (1 2 NT) 3) ( ) NEGINSERT)

(OPT (NP V ((NT)) PRED)
  (1 2 3 (4 2 NT 1)) ( ) TAGQUEST)

(OBLIG (NP V NT PRED V NT NP)
  (1 2 3 4 5 0 7) ( ) NEGDEL)

(BINARY (DET SENTB DET N IS ADJ SENTB N VP
  ((1 6) 0 0 0 0 0 0 8 9) ((1 3) (4 8) ADJEMB)

Note that the rule schemata symbol previously called X is here denoted by ANY.

To set a transformational component in the computer (i.e., to compute and store a set of inverse transformations), it is only necessary to execute the LISP function SETTRANS, which has as its single argument a list of transformations in the form given above. Initial singulary transformations, if there are any, are typed after the binary transformations. To set up the base component in the computer the following steps must be carried out:

(1) Define a list called CFGRAM whose elements are the given base component context-free rules.

(2) Define a list called AUXRULES which contains the set of auxiliary rules required.

(3) Define a list called TERMINABLE which consists of the set \{A → A⁰ | A ∈ N\} This list is necessary only for internal purposes with the UNIVAC M-460 program and is not required by the IBM 7090 version of the program. It is required to make possible the CF recognition of strings containing nonterminals.
The format to be used is illustrated by

SET (CFGRAM (  
(S¹ NP VP)   (NP DET N)   (VP V PRED)   (PRED ADV)   (NP DET COMP N)  
(N MAN)     (V IS)       (ADV HERE)     (PRED ADV)     (ADJ STRONG)  
(DET A) ))

SET (AUXRULES (  
(NP THERE)   (NP DET ADJ N)   (PRED DET N ADV)   (PRED DET ADJ N ADV)   
(S¹ V NP VP) (VP PRED)       (VP V NT PRED)     (VP V NT PRED V NT NP)   
(VP V PRED V NT NP)  (S¹ V NT NP VP)   (VP V NT PRED V NP)))

SET (TERMTABLE (  
(NP NF¹)     (N N¹)        (PRED PRED¹)   
(VP VP¹)     (V V¹)        (ADV ADV¹)     
(DET DET¹)   (ADJ ADJ¹)    (COMP COMP¹)  ))

After thus defining these rules, the internal tables required for subsequent context-free recognition are set up by executing the function of no arguments SETCFG.

Several self-contained fragments of a transformational grammar have recently been written and tested by J. Keyser, using the grammar format and computer recognition program documented in Appendix D. We will denote by 2A and 2B two of these grammars which share the common base component

(S¹ S)  
(S NP AUX VP)  

(VP V NP AGNNT ADV)  
(VP V NP ADV)  
(VP V NP AGNNT)  
(VP V NP)  
(VP V ADV)  
(VP V)  
(VP BE PRED)  
(V VTR)  
(V VINT)  
(ADV LOC)  
(PRED AP)  
(AP ADJ)
(AGNT BY)
(NP DET N NU COMP)
(NP DET N NU)
(DET NL)
(DET ART)
(DET WH ART)
(ART A PSAR)
(ART THE PSAR)
(ART DEM PSAR)
(DEM THAT NBR)
(PSAR ADM)
(N NPR)
(N NCM)
(NCM NCT)
(NCM NNS)
(NU SG)
(NU PL)
(LOC AT NPP)
(NPP DET N NU)
(AUX AUXA M HAVE EN BE ING)
(AUX AUXA M HAVE EN)
(AUX AUXA M BE ING)
(AUX AUXA HAVE EN BE ING)
(AUX AUXA HAVE EN)
(AUX AUXA BE ING)
(AUX AUXA)
(AUXA TNS)
(TNS PRES)
(TNS PST)

Note that our program does not require explicit mention of the rules $S \rightarrow \# S^1 \#$ or $\text{COMP} \rightarrow \# S^1 \#$. The symbol $S$ in the first two rules of the above grammar is not the distinguished symbol, but rather it is merely an unnecessary intermediate symbol, and the first two rules could just as well be replaced by the single rule $S^1 \rightarrow \text{NP AUX VP}$. Note also that the terminal symbols of this grammar are grammatical categories, no lexical rewriting rules having been given.

We will refer to the combination of this base component with the following transformational component as Grammar 2A.

Grammar 2A

(OPT (ANY NP AUXA ANY VTR NP BY ANY)
 (1 6 3 (4 BE EN) 5 0 (7 2) 8) ( ) PASSIVE)

124
(OPT (ANY N NU ((COMP)) TNS ANY)
   (1 2 3 4 (5 3) 6) ( ) NU)

(OPT (NP ANY NP ANY)
   (3 1) 2 0 4) ( ) WH^1)

(REPEATED (ANY ADM N NU THAT PRES NU BE ADJ ANY)
   (L (9 2) 3 4 0 0 0 0 0 10) ( ) ADJ)

(BINARY (ANY DET N NU SENTB WH THE ADM N NU ((S^1)) ((NP)) AUXA NU VP SENTB ANY)
   (1 2 3 4 0 0 0 0 0 0 (11 THAT) 12 13 14 15 0 17)
   ((3 9) (4 10)) RELEMB)

Grammar 2B consists of the same phrase structure component, and it has for its transformational component the two rules

Grammar 2B

(OPT (ANY N NU ((S^1)) TNS ANY)
   (1 2 3 4 (5 3) 6) ( ) NU)

(BINARY (ANY DET N NU SENTB WH THE ADM N NU ((S^1)) ((NP)) AUX VP SENTB ANY)
   (1 2 3 4 0 0 0 0 0 0 (11 THAT) 12 13 14 0 16)
   ((3 9) (4 10)) REL)

Finally, we include a pair of grammars that were considered when the recognition procedure used was a variant of our simplified procedure of Chapter III. These grammars share the common base component

(S^1 FRE NP AUX VP)
(S^2 NP AUX VP)

(PRE QU NEG)
(PRE QU)
(PRE NEG)

(AUX AUXA AUXB)
(AUX AUXA)

(AUXA TNS M)
(AUXA TNS)
(AUXA NOM)

(TNS PRES)
(TNS PAST)

(NOM FOR TO)
(NOM POSS ING)

(AUXB HAVE EN BE ING)
(AUXB HAVE EN)
(AUXB BE ING)
(VP BE PRED ADVB)
(VP BE PRED)
(VP V NP BY PASS ADV)
(VP V NP BY PASS)
(VP V NP)
(VP V ADV)
(VP V)

(PRED NP)
(PRED ADJ)
(PRED LOC)

(ADV ADVB)
(ADV LOC)

(LOC PREP NP)
(LOC HERE)

(NP DET N COMP)
(NP DET N)
(NP IT COMP)
(NP IT)
(NP NNUM)
(NP NPROP)
(NP NPROP COMP)
(NP PRO)
(NP PRO COMP)

(DET DEF)
(DET INDEF)
(DEF THE)
(INDEF A)
(INDEF S)

(M MAY)
(M CAN)
(M WILL)
(M SHOULD)
(M MUST)

(PREP ON)
(PRED IN)
(PREP UNDER)
(PRED BEHIND)
(PREP BY)

(NPROP JOHN)
(NPROP MARY)
(NPROP BOSTON)

(PRO I)
(PRO YOU)
We will denote by 3A the grammar consisting of this base component and the transformational component:
Grammar 3A

(OBLIG (((PRE)) NP AUX V NP BY PASS (((ADV))))
  (1 8 3 BE EN 4 0 6 2 8) () PASSIVE)

(OPT (((PRE)) NP AUX BE EN V BY NNUM (((ADV))))
  (1 2 3 4 5 6 0 0 8) () ACTDEL)

(OPT (ANY DET ((ADJ)) N (LOC) THAT PRES BE ADJ ANY)
  (1 2 9 3 4 5 0 0 0 10) () ADJPLMT)

(OPT (ANY DET ((ADJ)) N THAT PRES BE LOC ANY)
  (1 2 3 4 5 0 0 0 9) () LOCPLMT)

(OPT (ANY V1 NP TO BE LOC) (1 2 3 0 0 6) () TCBEDEL)

(BINARY (ANY DET N SENTE DET ((ADJ)) N (LOC)) AUX VP SENTE ANY)
  (1 2 9 3 8 0 0 0 0 0 10 12)
  (2 5 3 7) COMP1)

(BINARY (ANY DET N SENTE NP AUX V DET, ((ADJ)) N (LOC)) SENTE ANY)
  (1 2 3 8 11 0 0 0 0 12)
  (2 9 3 10) COMP2)

(BINARY (((PRE)) NP AUX V IT SENTE NP FOR TO VP SENTE ANY)
  (1 2 3 4 0 0 7 0 9 10 0 12) () COMP3)
A final grammar (3B) we include was written by S. Schane, and consists of the previously given base component and the transformations given below. We have translated the form in which these transformations were originally written to the present format, leaving, however, binary transformations in the inverse form that had to be supplied by the linguist at the time Schane wrote this grammar. This grammar is the largest we have considered to date.

Grammar 3B

(OBLIG (((PRE)) INDEF N AUX BE LOC ANY) (1 0 THERE 4 5 (2 3 6) 7) () THERE)

(OPT (((PRE)) NP AUX V NP BY PASS ANY) (1 5 (3 BE EN) 4 0 6 2 8) () PASSIVE)

(OPT (((PRE)) NP AUX BE EN V BY NNUM ANY) (1 2 3 4 5 6 0 0 9) () AGTDEL)

(OPT ( ANY DET N ANY THAT PRES BE ADJ ANY) (1 2 3 (4 8) 0 0 0 0 9) () RELDEL)

(OPT (ANY DET N ANY THAT PRES BE LOC ANY) (1 2 3 (4 8) 0 0 0 0 9) () RELDEL)

(OPT (ANY DET N THAT PRES V ANY) (1 2 3 0 ING 6 7) () RELDEL)

(OPT (((PRE)) IT THAT NNUM FOR TO ANY) (1 2 3 0 0 6 7) () FORDEL)

(OPT (((PRE)) IT THAT NNUM POSS ING ANY) (1 2 3 0 0 6 7) () POSSDEL)

(OPT (((PRE)) IT THAT ANY VP AUX VF) (1 2 0 0 0 6 (7 3 4 5)) () COMSHIFT)

(OPT (((PRE)) IT THAT ANY VP AUX VF) (1 0 3 4 5 6 7) () ITDEL)

(OPT (((PRE)) NP AUX V IT THAT ANY) (1 2 3 4 0 6 7) () ITDEL)

(OPT (ANY THAT (((NP)) NOM ANY) (1 0 3 4 5) () THATDEL)

(OPT (PRE ANY TNS HAVE ANY) (1 2 (3 4) 0 5) () MODINC)

(OPT (PRE ANY TNS BE ANY) (1 2 (3 4) 0 5) () MODINC)
(OBLIG ((QU) NEG NP TNS M ANY) (1 0 3 4 (5 INT) 6) () NEGCONT)
(OBLIG ((QU)) NEG NP AUXA ANY) (1 0 3 (4 NOT) 5) () NEG)
(OPT (QU NP AUXA ANY) (3 2 0 4) () QUEST)
(OPT (ANY DET N ING V ANY) (1 (2 4 5) 3 0 0 6) () INTOPLAC)
(REPEATED (ANY DET ANY N ADJ ANY) (1 2 (5 3) 4 0 6) () ADJPLAC)
(REPEATED (ANY TNS M ANY) (1 0 (3 2) 4) () TNSPLAC)
(REPEATED (ANY TNS V ANY) (1 0 (3 2) 4) () TNSPLAC)
(REPEATED (ANY S N ANY) (1 3 2 4) () ADHOC)
(BINARY (ANY DET N ANY THAT AUX VP ANY) (1 2 3 COMP 8) (2 3 4 6 7))
(BINARY (ANY DET N ANY THAT NP AUX V ANY) (1 2 3 COMP 9) (6 7 8 2 3 4))
(BINARY (ANY IT THAT NP AUX VP ANY) (1 2 COMP 7) (4 5 6))
(BINARY ((PRE) NP AUX V NP TO VP ANY) (1 2 3 4 5 COMP 8) (5 FOR 6 7))
(BINARY ((PRE) NP AUX V NP THAT NP AUX VP ANY) (1 2 3 4 5 COMP 10) (7 8 9))
APPENDIX C

RECOGNITION EXAMPLES

We discuss in this Appendix the recognition of a few sentences with respect to grammars listed in Appendix B. Consider first Grammar 1 of that appendix. The ordered list of inverse transformations is given by

(((NP V NT PRED V NP) (1 2 3 4 5 NT 6) OBLIG NEGDEL) ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST) ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((V NP ANY) (2 1 3) OPT INTERROG) ((THERE V DET ((ADJ)) N ADV) (3 4 5 2 6) OPT THERE) (BINARY (DET ADJ N VP) (1 COMP 3 4)) (1 3 IS 2)))

As examples of our basic computer output format consider the following computer results.

TIMERE C ( (THERE IS A MAN HERE IS NT THERE) )

22409
(PARSETIME 3541)
(CFRECTIME 8004)
(((S1 (NP (DET A) (N MAN)) (VP (V IS) (PRED (ADV HERE))))))
(THERE TAGQUEST))

TIMERE C ( (THERE IS NT A MAN HERE IS THERE) )

31732
(PARSETIME 10711)
(CFRECTIME 1 4931)
(((S1 (NP (DET A) (N MAN)) (VP (V IS) (PRED (ADV HERE)))))) (THERE NEGINSERT TAGQUEST NEGDEL)))

TIMERE C ( ((IS NT THERE A STRONG MAN HERE)) )

56239
(PARSETIME 15325)
(CFRECTIME 25092)
(((S1 (NP (DET A) (COMP ((S1 (NP (DET A) (N MAN)) (VP (V IS) (PRED (ADJ STRONG))))) NIL) (N MAN)) (VP (V IS) (PRED (ADV HERE))))) (THERE INTERROG NEGINSERT)))

131
The first number given is the total computing time in milliseconds (printing time included), PARSETIME refers to the time spent in context-free recognition exclusive of translating pushdown store automaton acceptance sequence representations of trees to equivalent LISP S-expressions, and CFFRECTIME refers to total time spent in context-free recognition. The structural descriptions are given as trees in the normal LISP Cambridge-Polish notation.

\[(S' \ (NP \ (DET \ A) \ (N \ MAN)) \ (VP \ (V \ IS) \ (PRED \ (ADV \ HERE))))\],

for example, denotes the tree

```
  S'
    /\          \ NP  VP
   /   \        /   /  \  \
  NP   VP      DET N  V  PRED
   |     |       A' MAN IS  ADV
   |     |           HERE

```

Transformations applied at each level are indicated by inserting a list of transformation names in parallel with the tree to which they apply.

More detailed printouts illustrating the progress of the recognition procedure are available. A complete display of the sequence of trees existing at every step is possible, for instance. In the recognition of the sentence \textit{a man is here} no transformations apply, of course, and the attempts to analyze this sentence into a sequence of trees satisfying one of the inverse structural indices are illustrated by the computer output
A more interesting example of the same type of print-out is the following recognition of there is nt a strong man here.

\text{TIMERECC ((THERE IS NT A STRONG MAN HERE IS THERE))}

1 \text{ ((THERE IS NT A STRONG MAN HERE IS THERE) (NP V NT PRED V NP) NIL)}
   \text{(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
   (V IS) (NP THERE)))
   \text{INVTRANARGS ((NP V NT PRED V NP) (1 2 3 4 5 NT 6) OBLIG NEGDEL)}
5 \text{ ((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) (NP THERE)))
   \text{((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE))}
   \text{((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE))}
   \text{(((S¹ (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN)
   (ADV HERE)))) (V IS) NT (NP THERE))))}
10 \text{(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE)))}
   \text{INVTRANARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)}
   \text{((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE))}
15 \text{(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE)))}
   \text{INVTRANARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)}
   \text{((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
   (V IS) NT (NP THERE))}
20 \text{(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
   (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
   (S¹) NIL)
On line 1 the given sentence is considered with respect to the inverse structural index (NP, V, NT, PRED, V, NP) of the last singulary transformation, NEGDEL. The result of successfully finding the proper analysis

\[
\begin{array}{c}
\text{NP, V, NT,} \\
\text{there is} \\
\text{a strong man here}
\end{array}
\]

is given on lines 2 and 3. Lines 4-6 indicate that the inverse NEGDEL transformation is to operate on this sequence of trees, and the resulting sequence of trees, differing only by the insertion of an extra NT, is given on lines 7 and 8. On lines 9-12 a check is made to verify that this resulting sequence of trees is a possible proper analysis of the derived constituent structure tree prior to application of the NEGDEL transformation, and on lines 13 and 14 this sequence is analyzed with respect to the inverse structural index of the TAGQUEST transformation, (NP, V, ((NT)), PRED, V, NT, NP). A sequence of trees, found on lines 15 and 16, is computed that satisfies this sequence, and lines 17-24 perform the inverse TAGQUEST transformation and verify that the output is a possible proper analysis of the derived constituent structure tree prior to application of TAGQUEST. Lines 25-35 similarly determine the applicability of the inverse NEGINsert transformation and perform this inverse transformation, giving the sequence of trees of line 31. This sequence of trees is analyzed with respect to the inverse structural index (V, NP, ANY) of the INTERROG transformation as indicated on lines 36 and 37, and the NIL of line 38 indicates no such analysis was possible. Lines 39-46
determine the applicability of the inverse THERE transformation and perform it, giving the sequence of trees of line 44. Lines 47-49 determine that the inverse binary transformation is applicable, and the matrix sentence continuation inverse transformation is first performed and built up to $S^1$ on lines 50-54. The constituent sentence continuation inverse transformation is performed and built up to $S^1$ on lines 55-59, and on lines 60-73 this continuation is put through the inverse transformational cycle from the beginning. No inverse transformations can be applied, and the phrase structure analysis of this continuation as a sentence is found on line 73. On lines 76-79 the constituent sentence structure is attached to the COMP symbol of the matrix sentence continuation and that continuation is built up to $S^1$ using only original base component rules (i.e. no auxiliary rules). The resulting structural description is found on lines 78-79, and it is repeated on lines 83-85 with the names of all transformations that were used attached.

We have simplified this print-out by making all inverse transformations (except the binary transformation) obligatory. It so happens for this example that this does not cause any path to be discarded that would have led to a structural description. This is not always the case, however, and continuations resulting from not performing applicable transformations must also be followed. We include below an extended print-out in which all paths were followed as is, in general, necessary.
TIMEREC (((THERE IS NT A STRONG MAN HERE IS THERE))
((THERE IS NT A STRONG MAN HERE IS THERE) (NP V NT PRED V NP) NIL)
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))
(INVTNARGS ((NP V NT PRED V NP) (1 2 3 4 5 NT 6) OBLIG NEGDEL)
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE))
(((S^1 (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN)
(ADV HERE))) (V IS) NT (NP THERE))))
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE))
(INVTNARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
(S^1 NIL)
(((S^1 (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN)
(ADV HERE))))))
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE))
(((ANY (NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (ANY (NP THERE)))
(((ANY (NP THERE)) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
(V IS) NT (ANY (NP THERE))))
(INVTNARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE)
(V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
(V IS)
NT (ANY (NP THERE))))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE))
(((S^1 (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN)
(ADV HERE))) (V IS) (NP THERE))))

137
40 (INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE)) (V IS) NT (ANY (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS) NT (NP THERE))))
   ((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE))
45 (((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (S'1 NIL)
   (((S'1 (NP THERE) (VP (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS) NT (NP THERE))))
   (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (V NP ANY) NIL))
50 NIL
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (THERE V DET ((ADJ) N ADV) NIL)
NIL
55 (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (DET ADJ N VP) NIL)
NIL
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (S'1 NIL)
60 (((S'1 (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS) NT (NP THERE))))
   (((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (V NP ANY) NIL))
NIL
65 (((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (THERE V DET ((ADJ) N ADV) NIL)
NIL
(((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (DET ADJ N VP) NIL)
NIL
70 (((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (S'1 NIL)
   (((S'1 (NP THERE) (VP (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS) NT (NP THERE))))
   (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (V NP ANY) NIL))
NIL
75 (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (THERE V DET ((ADJ) N ADV) NIL)
NIL
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (DET ADJ N VP) NIL)
NIL
80 (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (V NP ANY) NIL)
NIL
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE)) (S'1 NIL)
   (((S'1 (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS) NT (NP THERE))))
   (((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
   (V IS) NT (NP THERE))))))
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (ANY V NT ANY) NIL)
(((ANY (NP THERE)) (V IS) NT (ANY (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE))
 (V IS) NT (ANY (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))))
(((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (S¹ NIL)
(((S¹ (NP THERE) (VP (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
 (V NP ANY) NIL)
NIL)
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (V THERE V DET ((ADJ)) N ADV) NIL)
NIL)
(((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (S¹ NIL)
(((S¹ (NP THERE) (VP (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN)
 (ADV HERE))))))
(((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (V NP ANY) NIL)
NIL)
(((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (V THERE V DET ((ADJ)) N ADV) NIL)
(((There (V IS) (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
(INVTRANARGS ((There V DET ((ADJ)) N ADV) (3 4 5 2 6) OPT THERE)
 (There (V IS) (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 ((DET A) (ADJ STRONG) (N MAN) (V IS) (ADV HERE)))
 (((DET A) (ADJ STRONG) (N MAN) (V IS) (ADV HERE)) (S¹ NIL)
 (((S¹ (NP (DET A) (ADJ STRONG) (N MAN)) (VP (V IS) (PRED (ADV HERE)))))
 (((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (DET ADJ N VP) NIL)
NIL)
(((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
 (S¹ NIL)
(((S¹ (NP THERE) (VP (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
 (((DET A) (ADJ STRONG) (N MAN) (V IS) (ADV HERE)) (DET ADJ N VP) NIL)
NIL)
(((DET A) (ADJ STRONG) (N MAN) (VP (V IS) (PRED (ADV HERE))))
 (INVTRANS'ARGS ((DET ADJ N VP) (1 COMP 3 4)) ((DET A) (ADJ STRONG)
 (N MAN) (VP (V IS) (PRED (ADV HERE))))))
(((DET A) COMP (N MAN) (VP (V IS) (PRED (ADV HERE))))

139
Consider, for example, the inverse transformation of TAGQUEST, which is performed on lines 17-20. The resulting sequence of trees

\[
(1) \quad \left( \begin{array}{c}
\text{NP, V, NT, PRED, ADV} \\
\text{THERE IS} \\
\text{DET ADJ N HERE} \\
\text{A STRONG MAN}
\end{array} \right)
\]

is successfully recognized as a sentence with respect to the augmented CF grammar on lines 21-24. Rather than continuing on with this sequence of trees, however, they are temporarily stored, and the recognition proceeds on, starting at line 25 with the sequence of trees that were previously acted upon by TAGQUEST, namely

\[
(2) \quad \left( \begin{array}{c}
\text{NP, V, NT, PRED, ADV, V, NT, NP} \\
\text{THERE IS} \\
\text{DET ADJ N HERE IS THERE} \\
\text{A STRONG MAN}
\end{array} \right)
\]

This sequence of trees is analyzed with respect to inverse NEGINsert (lines 25-30), and this transformation is performed and the resulting continuation built up to \( S^1 \) on lines 31-39 and 40-48. Starting on line 49 the continuation in which inverse NEGINsert isn't applied is followed, i.e. the sequence of trees (2) is analyzed with respect to inverse INTERROG (lines 49-51), inverse THERE (lines 52-54), and the inverse binary transformation (lines 55-58). No proper analyses satisfying any of these inverse structural indices are found, so on lines 59-61 this sequence of trees is built up to the structure
However, since this structure reflects the use of auxiliary rules the continuation in question is terminated.

On lines 62-74 the continuation leading from application of inverse NEGINsert on lines 40-44 is followed and then discarded, and, likewise, the continuation resulting from performing inverse NEGINsert of lines 31-35 is followed and terminated on lines 75-88. Starting on line 89 the sequence of trees (1) resulting from application of TAGQUEST is again considered. Hence, we see that lines 25-38 were concerned with terminating all continuations leading from not performing the inverse TAGQUEST transformation of lines 17-20.

A long print-out such as this is useful in suggesting improvements in the recognition procedure as well as useful in finding certain types of errors in writing transformational grammars, but it is usually of no concern to the linguist user of the program. A shortened version of this print-out is also possible, in which only inverse transformations that are performed are printed, along with the sequences of trees they operate on and those they produce. For the previously considered example this shortened print-out is
TIMERECS ((THERE IS NT A STRONG MAN HERE IS THERE))

(INVTRANARGS ((NP V NT PRED V NP) (1 2 3 4 5 NT 6) OBLIG NEGDEL)
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))

(INVTRANARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE)))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINsert) ((ANY (NP THERE)
(V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))) (V IS)
NT (ANY (NP THERE))))
((NP THERE) (V IS) NT (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) (NP THERE)))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINsert) ((ANY (NP THERE)
(V IS) NT (ANY (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE)) (V IS)
NT (NP THERE))))
((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))
(V IS) NT (NP THERE)))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINsert) ((ANY (NP THERE)
(V IS) NT (ANY (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))
((NP THERE) (V IS) (PRED (DET A) (ADJ STRONG) (N MAN) (ADV HERE))))

(INVTRANARGS ((THERE V DET ((ADJ)) N ADV) (3 4 5 2 6) OPT THERE)
(THERE (V IS) (DET A) (ADJ STRONG) (N MAN) (ADV HERE)))
((DET A) (ADJ STRONG) (N MAN) (V IS) (ADV HERE)))

(INVTRANS'ARGS ((DET ADJ N VP) (1 COMP 3 4)) ((DET A) (ADJ STRONG)
(N MAN) (VP (V IS) (PRED (ADV HERE))))
((DET A) COMP (N MAN) (VP (V IS) (PRED (ADV HERE))))

(INVTRANARGS ((DET ADJ N VP) (1 3 IS 2)) ((DET A) (ADJ STRONG) (N MAN)
(VP (V IS) (PRED (ADV HERE))))
((DET A) (N MAN) IS (ADJ STRONG)))

314183
(PARSETIME 19257)
(CFRETTIME 48925)
(((S' (NP (DET A)) (COMP ((S' (NP (DET A) (N MAN)) (VP (V IS) (PRED
(ADJ STRONG))))) NIL) (N MAN)) (VP (V IS) (PRED (ADV HERE)))) (THERE
NEGINsert TAGQUEST NEGDEL)))

143
A few sentences were recognized by means of a modified program that made use of the procedure of Section 3.5. The following are examples obtained through the use of that program:
TIERREC (((THERE IS NT A MAN HERE IS THERE))

(INVTRANARGS ((NP V NT PRED V NP) (1 2 3 4 5 NT 6) OBLIG NEGDEL)
 ((NP THERE) (V IS) NT (PRED (DET A) (N MAN) (ADV HERE)) (V IS) (NP THERE)))
 ((NP THERE) (V IS) NT (PRED (DET A) (N MAN) (ADV HERE)) (V IS) NT (NP THERE)))

(INVTRANARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)
 ((NP THERE) (V IS) NT (PRED (DET A) (N MAN) (ADV HERE)) (V IS) NT (NP THERE)))
 ((NP THERE) (V IS) NT (PRED (DET A) (N MAN) (ADV HERE)) (V IS) (NP THERE)))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE))
 (V IS) NT (ANY (PRED (DET A) (N MAN) (ADV HERE))) (V IS) NT (ANY (NP THERE))))
 ((NP THERE) (V IS) NT (PRED (DET A) (N MAN) (ADV HERE))) (V IS) NT (NP THERE))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE))
 (V IS) NT (ANY (PRED (DET A) (N MAN) (ADV HERE)))
 ((NP THERE) (V IS) (PRED (DET A) (N MAN) (ADV HERE)))))

(INVTRANARGS ((THERE V DET ((ADJ)) N ADV) (3 4 5 2 6) OPT THERE)
 ((THERE (V IS) (DET A) EMPTY (N MAN) (ADV HERE)))
 ((DET A) (N MAN) (V IS) (ADV HERE)))

460805
 (PARSETIME 5190)
 (CFCRECTIME 18628)
 (((S (NP (DET A) (N MAN)) (VP (V IS) (PRED (ADV HERE)))) (THERE
 NEGINSERT TAGQUEST NEGDEL))))

TIERREC (((THERE A MAN HERE IS NT THERE))

(INVTRANARGS ((NP V ((NT)) PRED V NT NP) (1 2 3 4) OPT TAGQUEST)
 ((NP THERE) (V IS) EMPTY (PRED (DET A) (N MAN) (ADV HERE)) (V IS)
 NT (NP THERE)))
 ((NP THERE) (V IS) (PRED (DET A) (N MAN) (ADV HERE)))))

(INVTRANARGS ((ANY V NT ANY) (1 2 4) OPT NEGINSERT) ((ANY (NP THERE))
 (V IS) (PRED (DET A) (N MAN) (ADV HERE))) (V IS) NT (ANY (NP THERE))))
 ((NP THERE) (V IS) (PRED (DET A) (N MAN) (ADV HERE))) (V IS) NT (NP THERE))

(INVTRANARGS ((THERE V DET ((ADJ)) N ADV) (3 4 5 2 6) OPT THERE)
 ((THERE (V IS) (DET A) EMPTY (N MAN) (ADV HERE)))
 ((DET A) (N MAN) (V IS) (ADV HERE)))

280433
 (PARSETIME 16902)
 (CFCRECTIME 20097)
 (((S (NP (DET A) (N MAN)) (VP (V IS) (PRED (ADV HERE)))) (THERE
 TAGQUEST)))))
Note that even subtracting about a minute for printing time in each case, substantially more time was required for the same sentences by this method than by the method documented in Appendix D (i.e., by the procedure of 3.4). We do not wish to conclude anything from such limited testing, and more extensive experimenting is under way. Preliminary results indicate that function FIND', which finds all proper analyses of a tree that satisfy a given structural index, is inefficient for indices containing X (ANY), and this function is being rewritten before conducting further comparative studies.

We present in closing, a pair of examples of the recognition of sentences with respect to certain of the other grammars of Appendix B. The first is the recognition of the boy was examined by the doctor with respect to Grammar 3A, and the second is the sentence THE ADM NCT SG PRES SG VTR THE ADM NCT SG THAT PRES SG BE ADJ THAT PRES SG BE ADJ with respect to Grammar 2B. This sentence might be, for example, the student ignores the book that is worthwhile that is uninteresting. This example illustrates the complexity that can be encountered in the recognition of deeply embedded sentences even when the grammar in question is quite simple.
(((NP NDUM) (AUX (AUXA (TNS PAST)))) (V EXAMINE) (NP (DET (DEF THE)))
  (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE))) (N DOCTOR))))
  (ANY DET ((ADJ)) N ((LOC)) THAT AUX VP ANY) NIL)
NIL
(((NP NDUM) (AUX (AUXA (TNS PAST)))) (V EXAMINE) (NP (DET (DEF THE)))
  (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE))) (N DOCTOR))))
  (ANY DET ((ADJ)) N ((LOC)) THAT NP AUX V ANY) NIL)
NIL
(((NP NDUM) (AUX (AUXA (TNS PAST)))) (V EXAMINE) (NP (DET (DEF THE)))
  (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE))) (N DOCTOR))))
  ((((PRE)) NP AUX V NP TO VP ANY) NIL)
NIL
(((NP NDUM) (AUX (AUXA (TNS PAST)))) (V EXAMINE) (NP (DET (DEF THE)))
  (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE))) (N DOCTOR))))
  (S¹) NIL
(((S¹ (NP NDUM) (AUX (AUXA (TNS PAST)))) (VP (V EXAMINE) (NP (DET
  (DEF THE))) (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE)))
  (N DOCTOR))))))
(A STRUCTURAL DESCRIPTION IS) (((S¹ (NP NDUM) (AUX (AUXA (TNS PAST))))
  (VP (V EXAMINE) (NP (DET (DEF THE))) (N BOY)) BY PASS (ADV (LOC (PREP
  BY) (NP (DET (DEF THE))) (N DOCTOR)))))) (PASSIVE AGTDEL) 937236
57801 125371)
962568
(PARSETIME 57801)
(CFREETIME 125371)
(((S¹ (NP NDUM) (AUX (AUXA (TNS PAST)))) (VP (V EXAMINE) (NP (DET
  (DEF THE))) (N BOY)) BY PASS (ADV (LOC (PREP BY) (NP (DET (DEF THE)))
  (N DOCTOR)))))) (PASSIVE AGTDEL) (((S¹ (NP (DET (DEF THE))) (N
  DOCTOR)) (AUX (AUXA (TNS PAST)))) (VP (V EXAMINE) (NP (DET (DEF THE)))
  (N BOY)) BY PASS)))) (PASSIVE))
(INTRANARGS ((ANY N NU ((S1)) TNS NU ANY) (1 2 3 4 5 7) OPT NU)
((ANY THE ADM) (N (NCM NCT)) (NU SG) EMPTY (TNS PRES) (NU SG) (ANY
VTR THE ADM NCT SG THAT PRES SG BE ADJ THAT PRES SG BE ADJ))
(THE ADM (N (NCM NCT)) (NU SG) (TNS PRES) VTR THE ADM NCT SG THAT
PRES SG BE ADJ THAT PRES SG BE ADJ)
(INTRANARGS ((ANY DET N NU ((S1)) THAT ((NP)) AUX VP ANY) (1 2
3 4 COMP 10)) ((ANY THE ADM NCT SG PRES SG VTR) (DET (ART THE (PSAR
ADM))) (N (NCM NCT)) (NU SG) EMPTY THAT EMPTY (AUX (AUXA (TNS PRES)
(NU SG))) (VP BE (PRED (AP ADJ))))) (ANY THAT PRES SG BE ADJ)))
(THE ADM NCT SG PRES SG VTR (DET (ART THE (PSAR ADM))) (N (NCM NCT))
(NU SG) COMP THAT PRES SG BE ADJ)
(INTRANARGS ((ANY DET N NU ((S1)) THAT ((NP)) AUX VP ANY) (1 2
3 4 COMP 10)) ((ANY THE ADM NCT SG PRES SG VTR) (DET (ART THE (PSAR
ADM))) (N (NCM NCT)) (NU SG) (S1 (S THAT (AUX (AUXA (TNS PRES)
(NU SG))) (VP BE (PRED (AP ADJ)))) THAT EMPTY (AUX (AUXA (TNS PRES)
(NU SG))) (VP BE (PRED (AP ADJ))) (ANY))))
(THE ADM NCT SG PRES SG VTR (DET (ART THE (PSAR ADM))) (N (NCM NCT))
(NU SG) COMP)
(INTRANARGS ((ANY DET N NU ((S1)) THAT ((NP)) AUX VP ANY) (1 2
3 4 COMP 10)) ((ANY THE ADM (N (NCM NCT)) (NU SG) (TNS PRES) VTR)
(DET (ART THE (PSAR ADM))) (N (NCM NCT)) (NU SG) EMPTY THAT EMPTY
(AUX (AUXA (TNS PRES) (NU SG))) (VP BE (PRED (AP ADJ))))) (ANY THAT
PRES SG BE ADJ)))
(THE ADM (N (NCM NCT)) (NU SG) (TNS PRES) VTR (DET (ART THE (PSAR
ADM))) (N (NCM NCT)) (NU SG) COMP THAT PRES SG BE ADJ)
(INTRANARGS ((ANY DET N NU ((S1)) THAT ((NP)) AUX VP ANY) (1 2
3 4 COMP 10)) ((ANY THE ADM (N (NCM NCT)) (NU SG) (TNS PRES) VTR)
(DET (ART THE (PSAR ADM))) (N (NCM NCT)) (NU SG) (S1 (S THAT (AUX
(AUXA (TNS PRES) (NU SG))) (VP BE (PRED (AP ADJ)))) THAT EMPTY
(AUX (AUXA (TNS PRES) (NU SG))) (VP BE (PRED (AP ADJ))) (ANY))))
(THE ADM (N (NCM NCT)) (NU SG) (S1 (S THAT (AUX (AUXA (TNS PRES)
(NU SG))) (VP BE (PRED (AP ADJ)))) (AUX (AUXA (TNS PRES) (NU SG)))
(VP BE (PRED (AP ADJ))))
(INTRANARGS ((ANY DET N NU ((S1)) TNS NU ANY) (1 2 3 4 5 7) OPT NU)
((ANY WH THE ADM) (N (NCM NCT)) (NU SG) (S1 (S THAT (AUX (AUXA (TNS
PRES) (NU SG))) (VP BE (PRED (AP ADJ)))))) (TNS PRES) (NU SG) (ANY
(VP BE (PRED (AP ADJ)))))
(WH THE ADM (N (NCM NCT)) (NU SG) (S1 (S THAT (AUX (AUXA (TNS PRES)
(NU SG))) (VP BE (PRED (AP ADJ)))))) (TNS PRES) (VP BE (PRED (AP
ADJ))))

151
APPENDIX D

LISP PROGRAM DOCUMENTATION

We include in this appendix a rough flow chart of the procedure of Section 3.4 that shows which LISP functions call other functions, thus providing a rough hierarchy of the functions in question. SETTRANS converts generative transformations to their inverses as required for subsequent recognition. SETCFG sets up the base component for subsequent context-free grammar recognition. TIMESREC (and RECOGNIZE) perform transformational recognition, the recognition procedure itself being given, except for details, by the function IMPOSE. BUILD performs context-free recognition, FIND finds all proper analyses of a tree with respect to a structural index, and PROCES analyzes a sequence of trees into another sequence of trees satisfying a given inverse structural index. INTRAN and INTRANS perform singulary and binary (matrix sentence continuation) inverse transformations and PUTFORCOMP attaches a constituent structure sentence under an unused matrix sentence COMP symbol. JAY and RULES merely check a tree to determine whether or not it contains structure not given by the original base component context-free rules.

In addition to the function hierarchy we also include a listing of all the LISP functions. We have not included functions which synthesize a surface structure tree from a given specifier nor have we included functions to mechanically compute auxiliary rules. These functions have not yet been extensively tested nor have they in all cases been made fully operational.
ORGANIZATION OF LISP FUNCTIONS

Figure 6. 154
DEFINE ((
  (IMPOSE (LAMBD (L) (PROG (I1 I2 I3 I4 I5 I6 I7 I8 I9 I10)
    (COND (CONT (SETQ I9 L))
           (T (SETQ I8 (LIST L))))
    (SETQ I8 NIL))
  (SETQ CONT NIL))
A (COND ((AND (NOT (SW I1)) I2) (GO SAVE))
    ((NULL I9) (RETURN I2)))
  (SETQ I8 (CAR I8))
  (SETQ I8 (CDR I8))
  (SETQ I1 (CAR I5))
  (SETQ I2 (CADR I3))
  (SETQ I3 (CAADDR I2))
B (COND ((NULL I2) (GO G))
    ((EQ (CAAR I2) (QUOTE BINARY)) (GO J)))
  (SETQ I7 (PROCES (LIST I1 (CAAR I2) NIL)))
  (COND ((NULL I7) (GO C)))
E (SETQ I8 (INTRANS (CAR I2) (CAR I7)))
  (SETQ I5 (PROCES (LIST I8 (LIST (QUOTE S1)) NIL)))
  (COND ((NULL I5) (GO R)))
    ((EQ (CADDR I2) (QUOTE REPEATED)) (GO Q)))
  (SETQ I8 (CONS (LIST I6 (CDR I2)) (CONS (CAR (CDADDAR I2)) I3)) I8))
  (SETQ I7 (CDR I7))
R (COND ((NOT (NULL I7)) (GO E))
  (COND ((EQ (CADDR I2) (QUOTE OBLIG)) (GO A)))
P (SETQ I8 (CDR I2))
C (GO B)
J (SETQ I7 (PROCES (LIST I1 (CADAR I2) NIL)))
  (COND ((NULL I7) (GO C)))
  (SETQ I8 (INTRANS (LIST (CADAR I2) (CADDAR I2)) (CAR I7)))
  (SETQ I5 (PROCES (LIST I8 (LIST (QUOTE S1)) NIL)))
  (COND ((OR (NULL I5) (AND ONEBPL (JAY AUKRULES (RULES I5))) (GO M)))
    (SETQ I5 NIL)
    (SETQ I8 (INTRANS (CONS (CADAR I2) (CADDAR I2)) (CAR I7)))
    (SETQ I5 (PROCES (LIST I1 (LIST (QUOTE S)) NIL)))
    (COND ((NULL I5) (GO M)))
    (SETQ I3 (RECOGNIZE I10))
  )
  (COND ((NULL I5) (GO M)))
  (SETQ I8 (CONS (LIST (PUFFFORCOMP (CAR I5) I8) (COND (ONEBPL NIL)
       (T I2)) I3) I8))
    (SETQ I5 (CDR I3))
    (GO L)
M (SETQ I7 (CDR I7))
  (COND ((NULL I7) (GO C)))
  (GO N)
G (SETQ I7 (PROCES (LIST I1 (LIST (QUOTE S1)) NIL)))
H (COND ((NULL I7) (GO A))
    ((JAY AUKRULES (RULES (CAAR I7))) (GO S)).
    ((AND SDAUVAIL I5) (PRINT (LIST (QUOTE (A STRUCTURAL
       DESCRIPTION IS)) (LIST (CAR I7) I3) (CLOCK PARSETIME CFREETIME))))
    (SETQ I4 (CONS (LIST (CAR I7) I3) I4))
S (SETQ I7 (CDR I7))
  (GO H)
Q (SETQ I8 (CONS (LIST I8 I2 (CONS (CAR (CADDAR I2)) I3)) I8))
  (GO R)
SAVE (SETQ RESV I6)
  (PUNCH I6)
  (RETURN I6) ))) 155
DEFINe ((
(FIND' (LAMBDA (X Z) (PROG (F'))
   (SETQ F' (REDUCE (FIND' X NIL (COND ((ATOM Z) (LIST (LIST Z))) (T (LIST Z)))))
   (COND ((EQ (CAR X) (QUOTE ANY)) (RETURN (PRUNE (LIST (LIST (QUOTE ANY) Z)) (CDR X)) F'))))
   (RETURN F'))))
)

(PRUNE (LAMBDA (M N) (COND ((NULL N) NIL) ((EQUAL M (CAR N)) (CDR N))
   (T (CONS (CAR N) (PRUNE M (CDR N))))))))
))

SET (CONC (LAMBDA (L M) (PROG (P)
   (COND ((NULL M) (RETURN L))
   ((NULL L) (RETURN M)))
   (SETQ P L)
   (COND ((NULL (CDR P)) (GO B)))
   (SETQ P (CDR P))
   (GO A)
   B
   (SETQ P (RPLACD P M))
   (RETURN L)))))
))

SET (JAY (LAMBDA (X Y) (COND ((NULL X) F) ((MEMBER (CAR X) Y) T)
   (T (JAY (CDR X) Y))))))

SET (RULES (LAMBDA (X) (PROG (J' J'' J''' J'''))
   (COND ((NULL X) (RETURN NIL))
   ((NULL (CDR X)) (RETURN (RULES (CAR X)))
   (NOT (ATOM (CAR X))) (RETURN (APPEND (RULES (CAR X)) (RULES (CDR X))))))
   (SETQ J' (LIST X))
   E
   (COND ((NULL J') (RETURN J'''))
   (SETQ J'' NIL)
   (SETQ J''' (CAR J'))
   (SETQ J' (CDR J'))
   A
   (COND ((NULL J'') (GO D))
   ((ATOM (CAR J'')) (GO B)))
   (SETQ J' (CONS (CAR J'') J'))
   (SETQ J''' (APPEND J''' (LIST (CAAR J'')))
   (SETQ J'' (CDR J''))
   (GO A))
   B
   (SETQ J''' (APPEND J''' (LIST (CAR J'')))
   (GO A))
   D
   (SETQ J''' (CONS J''' J'''))
   (GO E))))))
157
(DEFINE (FIND X Y Z) (PROG (R P P^2 P^3)
  (COND ((GR (NULL X) (NULL Z)) (RETURN NIL)))
  (SEQ R (FIND X Y (BREAKDOWN Z)))
  (COND ((NULL (CDAR Z)) (GO A)))
  (SEQ P^2 (CAR Z)) (GO B)
  (SEQ P^2 (CAAR Z)) (GO C)
  (COND ((NOT (ATOM (CAR X))) (GO J))
    ((AND (EQ (CAR X) (CAAR Z)) (NULL (CDR Z)))
     (GO D))
    ((EQ (CAR X) (CAAR Z)) (GO D))
    ((EQ (CAR X) (QUOTE ANY)) (GO F)))
  (RETURN R))
  (RETURN (CONCIF R (LIST (CONS (REVERSE (CONS P^1 Y)) (LIST (CDR X)))))
  (RETURN (CONCIF R (FIND X (CONS P^1 Y) (CDR Z))))
  (COND ((NULL Y) (GO G)))
  ((ATOM (CAR Y)) (GO G))
  ((ATOM (CAAR Y)) (GO G))
  ((NULL (CDR Z)) (RETURN (CONCIF R
    (LIST (CONS (REVERSE (CONS (CONS P^1 (CAAR Y))
      (CDR Y))) (LIST (CDR X))))))))
  (RETURN (CONCIF R (CONCIF
    (FIND X (CONS (CONS P^1 (CAAR Y)) (CDR Y)))
    (FIND X (CONS (CONS P^1 (CAAR Y)) (CDR Y)) (CDR Z))))
  (COND ((NULL (CDR Z)) (RETURN (CONCIF R (CONCIF
    (FIND X (CONS (QUOTE EMPTY) Y) Z)
    (LIST (CONS (REVERSE (CONS (LIST (LIST P^1)) Y))
      (CDR X))))))))
  (RETURN (CONCIF R (CONCIF
    (FIND X (CONS (QUOTE EMPTY) Y) Z)) (CONCIF
    (FIND X (CONS (CONS P^1 Y) (CDR Z)))
    (FIND X (CONS (CONS P^1 Y) (CDR Z))))
  (COND ((ATOM (CAAR X)) (GO I))
    ((NOT (ATOM (CAAR X))) (RETURN NIL)))
  (SEQ P^3 (CONCIF R (FIND X (CONS (QUOTE EMPTY) Y) Z)))
  (SEQ P^3 (CAR X))
  (COND ((AND (EQ (CAR P^3) (CAAR Z)) (NULL (CDR Z))) (GO C))
    ((EQ (CAR P^3) (CAAR Z)) (GO D))
    ((NULL (CDR P^3)) (RETURN R)))
  (SEQ P^3 (CDR P^3)) (GO K)
  (SEQ P^3 (CAR X)) (GO K)))
  (CONCIF (LAMBDA (X Y)
    (COND ((NULL X) Y) ((NULL Y) X) (T (CONC X Y))))))
  (BREAKDOWN (LAMBDA (Z) (PROG (P NEWTREES)
    (COND ((NULL (CDAR Z)) (RETURN NIL)))
    (SEQ P (CDAR Z))
    (COND (ATOM (CAR P)) (RETURN NIL))
    (SEQ (NEWTREES (CONC NEWTREES (LIST (CAR P)))) (GO G)
      (SEQ P (CDR P)) (GO B)))))

158
DEFINE ((
(REDUCE (LAMBDA (X) (PROG (P^1 P^2 P^3 P^4 P^5)
  (COND ((NULL X) (RETURN NIL)))
    (SETQ P^1 X)
  A
    (COND ((NULL P^1) (GO F)))
    (SETQ P^2 (CAAR P^1))
  B
    (COND ((ATOM (CAR P^2)) (GO C))
      ((ATOM (CAAR P^2)) (GO C)))
    (RPLACA P^2 (CONS (QUOTE ANY) (REVERSE (CAAR P^2)))))
  C
    (SETQ P^2 (CDR P^2))
    (COND ((NULL P^2) (GO D)))
    (GO B))
  D
    (SETQ P^1 (CDR P^1)) (GO A))
  F
    (SETQ P^4 X)
    (SETQ P^3 (CAR P^4)) (SETQ P^4 (CDR P^4))
    (COND ((NULL P^1) (GO I)))
    (SETQ P^3 P^1)
  H
    (COND ((INCLUDED (CAR P^3) (CAAR P^2)) (GO G)))
    (SETQ P^2 (CDR P^2))
    (COND ((NOT (NULL P^2)) (GO H)))
  I
    (COND ((NULL P^4) (RETURN (CONS P^3 P^1))))
    (SETQ P^4 P^4)
  J
    (COND ((INCLUDED (CAR P^3) (CAAR P^5)) (GO G)))
    (SETQ P^5 (CDR P^3))
    (COND ((NOT (NULL P^5)) (GO J)))
    (SETQ P^1 (CONS P^3 P^1))
    (GO G))
  )))

(INCLUDED (LAMBDA (X Z)
  (COND ((AND (NULL X) (NULL Z)) T)
    ((OR (NULL X) (NULL Z)) F)
    ((EQUAL (CAR X) (CAR Z)) (INCLUDED (CDR X) (CDR Z)))
    ((OR (ATOM (CAR X)) (ATOM (CAR Z))) F)
    ((AND (BQ (CAAR X) (QUOTE ANY)) (BQ (CAAR Z) (QUOTE ANY)))
      (COND ((BELOW (CDAR X) (CDAR Z)) (INCLUDED (CDR X) (CDR Z)))
          (T F))))
  (T F))))

(BELOW (LAMBDA (X Z)
  (COND ((AND (NULL X) (NULL Z)) T)
    ((OR (NULL X) (NULL Z)) F)
    ((EQUAL (CAR X) (CAR Z)) (BELOW (CDR X) (CDR Z)))
    ((ATOM (CAR X)) F)
    ((ATOM (CAR X)) (BELOW (CONS (LIST (CAR X)) (CDR X)) Z))
    (T (BELOW X (BREAKDOWN Z)) ))))
DEFINE {
  (PARSE (LAMBDA (INF SWITCH) (PROG (PD HIST PATH TEM SVAR PRODS))
    (SESEQ PD (LIST DESIGN)))
  
  S11 (COND ((AND (NULL INF) (NULL PD)) (GO SUCCEED)))
  S12 (COND ((EQUAL (CAR INF) (CAR PD)) (GO ACT13)))
  S13 (COND ((EQUAL (CAR INF) (QUOTE T)) (GO ACT12)))
  S14 (COND ((NOT (TERMIN (CAR PD))) (GO ACT13)))))
  (GO BACKUP)
  S31 (COND ((NOT (TERMIN (CAR PD))) (NOT (TERMIN (CAR PD))))
  (GO ACT31)))
  (GO BACKUP)
  ACT11 (SESEQ HIST (CONS (LIST INF PD PRODS PATH (QUOTE S13)) HIST))
  (SESEQ PD (CDR PD))
  (SESEQ INF (CDR INF))
  (GO S11)
  ACT12 (SESEQ PD (CDR PD))
  (GO S31)
  ACT13 (SESEQ PRODS (GETRULES (CAR INF)))
  (COND ((NULL PRODS) (GO BACKUP))
    ((NOT (OR (PREC (CAAR PRODS) (CAR PD))
       (EQUAL (CAAR PRODS) (CAR PD)))) (GO ACT13B)))
  (SESEQ HIST (CONS (LIST INF PD (CDR PRODS) PATH (QUOTE ACT13A)) HIST))
  (SESEQ PD (TRANSF PD (CONS (CAAR PRODS) (CONS (QUOTE T)
       (REVERSE (CDAR PRODS))))))
  (SESEQ PATH (CONS (CONS (CAAR PRODS) (CONS (CAR INF) (CDAR PRODS))) PATH))
  (SESEQ INF (CDR INF))
  (GO S11)
  ACT13A (SESEQ PRODS (CDR PRODS))
  (GO ACT13A)
  ACT13B (SESEQ PRODS (CDR PRODS))
  (GO S11)
  ACT31 (SESEQ PRODS (GETRULES (CAR PD)))
  (COND ((NULL PRODS) (GO S32))
    ((NOT (OR (PREC (CAAR PRODS) (CAR PD))
       (EQUAL (CAAR PRODS) (CDAR PD)))) (GO ACT31B)))
  (SESEQ HIST (CONS (LIST INF PD (CDR PRODS) PATH (QUOTE ACT31A)) HIST))
  (SESEQ PATH (CONS (CONS (CAAR PRODS) (CONS (CAR PD) (CDAR PRODS))) PATH))
  (SESEQ PD (TRANSF (CDR PD) (CONS (CAAR PRODS) (CONS (QUOTE T)
       (REVERSE (CDAR PRODS))))))
  (GO S11)
  ACT31A (SESEQ PRODS (CDR PRODS))
  (GO ACT31A)
  ACT32 (SESEQ PD (CDR PD))
  (GO S11)
  BACKUP (COND ((NULL HIST) (RETURN SVAR)))
  (SESEQ TEM (CAR HIST))
  (SESEQ HIST (CDR HIST))
  (SESEQ INF (CAR TEM))
  (SESEQ PD (CAR TEM))
  (SESEQ PRODS (CAR TEM))
  (SESEQ PATH (CAR TEM))
  (COND ((EQ TEM (QUOTE S13)) (GO S13))
     ((EQ TEM (QUOTE S14)) (GO S14))
     ((EQ TEM (QUOTE ACT13A)) (GO ACT13A))
     ((EQ TEM (QUOTE ACT31A)) (GO ACT31A))
     (SUCCEED (SESEQ PATH NIL)
       (COND ((EQ SWITCH (QUOTE ONE)) (RETURN SVAR))
         (T (GO BACKUP))))
     ))
)
DEFINE ((

(SETQ (LAMBDA (RULES DES) (PROG (TERMS NTERMS))
  (SETQ DESIG DES)
  (CLEARBITS)
  (ASSIGN RULES)
  (SETQ TERMS DESCRIM RULES))
  (SETQ NTERMS (CDR TERMS))
  (SETQ TERMS (CAR TERMS))
  (PRECPAIRS RULES)
  (ITERBITS)
  (MAPLIST TERMS (FUNCTION (LAMBDA (J) (RPLACA (CADDR J)
    (QUOTE TERM))))))
  (MAPLIST NTERMS (FUNCTION (LAMBDA (J) (RPLACA (CADDR J)
    (QUOTE NTERM))))))
  (MAPLIST (REVERSE RULES) (FUNCTION (LAMBDA (J) (PROCESS (CAR J))))))
  (RETURN NIL) )))

(DESCRIM (LAMBDA (X) (PROG (TERMS NTERMS))
  (MAPLIST X (FUNCTION (LAMBDA (J) (SETQ NTERMS (UNION
    (LIST (CAR J)) NTERMS))))))
  (MAPLIST X (FUNCTION (LAMBDA (J) (SETQ TERMS (UNION
    (SETDF (CDAR J) NTERMS) TERMS))))))
  (RETURN (CONS TERMS NTERMS)) ))))

(PROCESS (LAMBDA (X) (PROG (P' P")
  (SETQ P' (CONS (CAR X) (CDDR X))
  (SETQ X (CADR X))
  (SETQ P" (CADDR X))
  (RETURN (COND ((MEMBER P' P") NIL)
    (T (RPLACA (CDDD R X) (CONS P' P")))))) )))

(ASSIGN (LAMBDA (X) (PROG (P' P" NUM)
  (SETQ NUM '1)
  A (COND ((NULL X) (RETURN NIL)))
  (SETQ P' (CAR X))
  (SETQ X (CDR X))
  B (COND ((NULL P') (GO A))
    ((MEMBER (CAR P') P") (GO C))
    (SETQ P" (CONS (CAR P') P"))
    (RPLACD (CDAR P') (CONS NIL (ASSIGN' NUM)) (CONS NIL (CDDAR P'))))
  (SETQ NUM (ADD' NUM))
  C (SETQ P' (CDR P'))
  (GO B) ))))

(PRECPAIRS (LAMBDA (X) (PROG NIL
  A (COND ((NULL X) (RETURN NIL))
    (T (DOPREC (CADAR X) (CAAR X))))
  (SETQ X (CDR X))
  (GO A) ))))
))
(SET (CFREC (LAMBDA (X DESIG) (MAPLIST (PARSE X NOOPPARSINGS)
            (FUNCTION (LAMBDA (J) (LIST (TREE (CAAR J)) (CDAR J)))))))))

DEFINE ((

  (TREE (LAMBDA (X) (PROG (NOTS NOPS LS RS RSL TOT SPEC)
                (COND ((NULL (CDDR X)) (RETURN (CAR X))))
            (COND ((AND (NOT (ATOM (CAR X))) (NOT (ATOM (CADR X)))) (GO TESTC))
                ((NULL (CDDR X)) (GO EX1)))
    (SEIQ LS (CONS (CAR X) LS))
    (SEIQ X (CDR X))
    (GO A)
  TESTC (COND ((COUNTUP (CDR X)) (GO SFOUND))
            (GO B))
  SFOUND (SEIQ SPEC (CAR X))
    (SEIQ LS (REVERSE LS))
    (SEIQ X (CDR X))
    (SEIQ NOTS 0)
    (SEIQ NOPS 0)
  LOOP (COND ((AND (ATOM (CAR X)) (NULL (CDR X))) (GO EX2))
              ((ATOM (CAR X)) (SETQ NOTS (ADD1 NOTS)))
            (T (SEIQ NOPS (ADD1 NOPPS))))
    (SEIQ RS (CONS (CAR X) RS))
    (SEIQ X (CDR X))
    (COND ((AND (EQUAL NOPS NOTS) (TERMIN (CADR (NEXTP X)))) (GO D)))
    (GO LOOP)
  D (SEIQ RSL (CONS (REVERSE RS) RSL))
    (SEIQ RS NIL)
    (GO C)
  EX1 (SEIQ SPEC (CAR X))
    (SEIQ TOT (LIST (REVERSE LS)))
    (GO EXIT)
  EX2 (SEIQ TOT (COND ((NULL LS) (REVERSE RSL)) (T (CONS
            LS (REVERSE RSL))))))
  EXIT (RETURN (CONS (CAR SPEC) (TREE (CDR SPEC) (MAPLIST
            TOT (FUNCTION (LAMBDA (J) (TREE (CAR J)))))))))
))
DEFINE ((
    (TREE¹ (LAMBDA (X Y) (COND
      ((NULL X) NIL)
      ((TERM$IN (CAR X)) (CONS (CAR X) (TREE¹ (CDR X) Y)))))
    (T (CONS (CAR Y) (TREE¹ (CDR X) (CDR Y)))))))))

(NEXTP (LAMBDA (X) (COND
  ((NULL X) (QUOTE (X (X X TERM)))))
  ((ATOM (CAR X)) (NEXTP (CDR X)))
  (T (CAR X)))))

(COUNTUP (LAMBDA (X) (PROG (NOTS NOPS)
    (SETQ NOTS 0)
    (SETQ NOPS 0)
    (COND ((AND (ATOM (CAR X)) (NULL (CDR X))) (RETURN T))
      ((ATOM (CAR X)) (SETQ NOTS (ADD¹ NOTS)))
      (T (SETQ NOPS (ADD¹ NOPS))))
    (COND ((GREATERP NOTS NOPS) (RETURN F)))
    (SETQ X (CDR X))
    (GO A))
  ))))

163
((FUNCTION ASSIGN)) (UANDL 7777Q (QUOTE ASSIGN))
ASSIGN
(UANDL o o)
(RJ GEINO)
(J ASSIGN o o 1)

((FUNCTION TERMIN)) (UANDL 7777Q (QUOTE TERMIN))
TERMIN
(UANDL o o)
(RJ (SPECIAL CAADDR) o o 1)
(SBA (QUOTE NTERM))
(J TERMIN o o 1)

((FUNCTION GETRULES)) (UANDL 7777Q (QUOTE GETRULES))
GETRULES
(UANDL o o)
(RJ (SPECIAL CAADDR) o o 1)
(J GETRULES o o 1)

((FUNCTION DOPREC)) (UANDL 7777Q (QUOTE DOPREC))
DOPREC
(UANDL o o)
(STQ TEM o o 1)
(RJ (SPECIAL CDADDR) o o 1)
(STA)
(LDA o 1 o 1)
(RJ (SPECIAL CDADDR) o o 1)
(RJ SEEBIT)
(LDQ TEM o o 1)
(LDA)
(J DOPREC o o 1)

TEM
(UANDL o o)

((FUNCTION PREC)) (UANDL 7777Q (QUOTE PREC))
PREC
(UANDL o o)
(STQ TEM o o 1)
(RJ (SPECIAL CDADDR) o o 1)
(STA)
(LDA o 1 o 1)
(RJ (SPECIAL CDADDR) o o 1)
(RJ LOOKATBIT)
(LDQ TEM o o 1)
(ADA o 5)
(J PREC o o 1)
(LDA (QUOTE T))
(J PREC o o 1)

TEM
(UANDL o o)

164
((FUNCTION CLEARBITS) (UANDL Q (QUOTE CLEARBITS))
CLEARBITS
    (UANDL O O)
    (STQ TEM O O 1)
    (LDA MATLOC O O 1)
    (LDQ MATDIM O O 1)
    (RJ CLEARMTRI)
    (LDQ TEM O O 1)
    (J CLEARBITS O O 1)
TEM
    (UANDL O O)
MATLOC
    (Q ESCO Q)
MATDIM
    (200)
)

((FUNCTION ITERBITS) (UANDL Q (QUOTE ITERBITS))
ITERBITS
    (UANDL O O)
    (STQ TEM O O 1)
    (RJ FINDLIMIT)
    (LDQ TEM O O 1)
    (J ITERBITS O O 1)
TEM
    (UANDL O O)
)
DEFINE ((
  (SETCFG (LAMBDA NIL (SETGR (APPEND TERMINAL (APPEND AUXRULES GRAMMAR))
         (QUOTE S'))) )))

DEFINE ((
  (SETRANS (LAMBDA (N) (PROG (N1 N2 N3 N4 N5 N6 N7 D1 D2 D3)
       (SETQ N1 N)
       (COND ((NULL N1) (RETURN (SETQ TRANS N2))))
       (SETQ N3 (CAR N1))
       (SETQ N1 (CDR N1))
       (COND ((EQ (CAR N3) (QUOTE BINARY)) (GO B)))
       (SETQ N4 (TRANSFORM (NUMBARG (CADR N3)) (CADDR N3)))
       (SETQ N5 (INVERS (NUMBARG (CADR N3)) N4 (CADDR N3)))
       (COND ((NULL N5) (SETQ N2 (CONS (LIST (SIMPLIFY N4) N5 (CAR N3)
             (CADDR (CDDR N3)))) (N2) ) (T
         (SETQ N3 (CONS (LIST (SIMPLIFY N4) N5 (CAR N3) (CADDR
             (CDDR N3)))) N5))))
       (GO A) )
       (SETQ N8 (QUOTE INS))
       (SETQ D2 NIL)
       (SETQ D3 NIL)
       (SETQ D1 (NUMBARG (CADR N3)))
       (COND ((EQ (CAR D1) (QUOTE SENTE)) (GO D)))
       (SETQ D2 (APPEND D2 (LIST (CAR D1))))
       (SETQ D1 (CDR D1))
       (GO C) )
       (SETQ D1 (CDR D1))
       (COND ((EQ (CAR D1) (QUOTE SENTB)) (GO E)))
       (SETQ D3 (APPEND D3 (LIST (CAR D1))))
       (GO D) )
       (SETQ D1 (CDR D1))
       (COND ((EQ (CAR D1) (QUOTE SENTE)) (GO E)))
       (SETQ D3 (APPEND D3 (LIST (QUOTE COMP)) (CDR D1)))
       (GO D) )
       (SETQ N4 (TRANSFORM (NUMBARG (CADR N3)) (CADDR N3)))
       (SETQ N5 (INVERS (APPEND D2 (CONS (LIST (QUOTE COMP)) (CDR D1))) N4 (CADDR N3)))
       (SETQ N7 (INVERS D3 N4 (CADDR N3)))
       (SETQ N2 (APPEND N2 (LIST (LIST (CAR N3) (SIMPLIFY N4) N5 N7))))
       (GO A) ))))

166
SET
(RECOGNIZE (LAMBDA (L) (IMPOSE (LIST L TRANS NIL))))

SET
(BUILD (LAMBDA (X Y) (PROG (B1 B2 B3 B4 B5)
(COND ((AND (NOT (ATOM Y)) (NOT (ATOM (CAR Y))))
  (EQUAL X (CAAR Y))) (RETURN NIL))
  (SETQ B1 Y)
  (COND ((EQUAL X (QUOTE ANY)) (GO E))
  ((EQUAL X (QUOTE COMP)) (GO G)))
A
  (COND ((NULL B1) (GO C))
  ((EQUAL (CAR B1) (QUOTE COMP)) (GO J))
  ((ATOM (CAR B1)) (GO B)))
  (SETQ B2 (CONS (REPL1 (CAAR B1) TERMTABLE) B2))
H
  (SETQ B3 (CONS (CAR B1) B3))
  (SETQ B1 (CDR B1))
  (GO A)
B
  (SETQ B2 (CONS (CAR B1) B2))
  (SETQ B1 (CDR B1))
  (GO A)
C
  (SETQ B4 (CFREC (REVERSE B2) X))
  (GO D)
D
  (COND ((NULL B4) (RETURN B5)))
  (SETQ B3 (CONS (CAR (REPL1 (CAR B4) (REVERSE B3))) B5))
  (SETQ B4 (CDR B4))
  (GO D)
E
  (COND ((NULL B1) (RETURN (CONS (CONS (CONS X B4) B1) B2)))
  (SETQ B2 (CONS (LIST (CONS X B4) B1) B2))
  (SETQ B4 (APPEND B4 (LIST (CAR B1))))
  (SETQ B1 (CDR B1))
  (GO E)
G
  (COND ((NULL B1) (RETURN B2)))
  (SETQ B4 (APPEND B4 (LIST (CAR B1))))
  (SETQ B1 (CDR B1))
  (SETQ B2 (CONS (LIST (CONS X B4) B1) B2))
  (GO C)
J
  (SETQ B2 (CONS (REPL1 (CAR B1) TERMTABLE) B2))
  (GO H)))

167
DEFINE ((
  (INVERS (LAMBDA (X Z Y) (PROG (Q^1 Q^2 Q^3 Q^4))
    (SETOQ Q^2 X)
    A
    (SETOQ Q^1 Z)
    (SETOQ Q^3 1)
    (COND ((NULL Q^2) (RETURN Q^4)))
    B
    (COND ((NULL Q^1) (SETOQ Q^4 (APPEND Q^4 (LIST (CAAR Q^2))))
      ((OR (EQUAL (CAR Q^1) (CAR Q^2)) (AND (EQUAL (CAR Q^1) (CAR Q^2))
        (OR (MEMBER (CONS (CADAR Q^2) (CDAR Q^2)) Y) (MEMBER (CONS
          (CADAR Q^1) (CDAR Q^2)) Y))) (SETOQ Q^4 (APPEND Q^4 (LIST Q^3))))
      (SETOQ Q^2 (CDR Q^2))
      (GO A))
      (SETOQ Q^3 (ADD 1 Q^3))
      (GO B))))
  )))

SET
(TRANSFORM (LAMBDA (X Y) (PROG (N^1 N^2 N^3) (SETOQ N^2 X) A (SETOQ N^1 X) (COND ((NULL N^2) (RETURN N^3)))
    B
    (COND ((NUMBERP (CAR N^2)) (GO C)) ((ATOM (CAR N^2)) (GO G)) (T (SETOQ N^2 (APPEND (CAR N^2) (CDR N^2))))
    (GO B))
    C
    (COND ((ZEROP (CAR N^2)) (GO E)) (T (GO D)))
    D
    (COND ((EQUAL (CAR N^2) (CAR (CDAR N^1))) (SETOQ N^3 (APPEND N^3 (LIST (CAR N^1)))) (T (GO F)))
    E
    (SETOQ N^2 (CDR N^2)) (GO A))
    F (SETOQ N^1 (CDR N^1)) (GO D)
    G
    (SETOQ N^3 (APPEND N^3 (LIST (CAR N^2)))) (GO E))))

SET
(NUMBARG (LAMBDA (X) (PROG (X^1 X^2 X^3) (SETOQ X^1 X) (SETOQ X^2 1) A (COND ((NULL X^1) (RETURN X^3)))
    (SETOQ X^3 (APPEND X^3 (LIST (LIST (CAR X^1) X^2)))) (SETOQ X^2
    (ADD 1 X^2)) (SETOQ X^1 (CDR X^1)) (GO A))))

SET
(SIMPLIFY (LAMBDA (X) (PROG (S^1 S^2)
    (SETOQ S^1 X)
    A
    (COND ((NULL S^1) (RETURN S^2))
      ((ATOM (CAR S^1)) (GO B))
      (T (SETOQ S^2 (APPEND S^2 (LIST (CAAR S^1)))))))
    B
    (SETOQ S^2 (APPEND S^2 (LIST (CAR S^1))))
    C
    (SETOQ S^1 (CDR S^1))
    (GO A))))

168
VALUE (REPL^1)
(LAMBDA (X V) (COND ((EQUAL X (CAAR V)) (CAR (CDAR V))) (T (REPL^1 X (CDR V))))))

VALUE (REPL^2)
(LAMBDA (U V) (PROG (R^1 R^2 R^3 R^4) (SETQ R^1 U) (COND ((NULL R^1) (RETURN (LIST NIL V))) ((ATOM (CAR R^1)) (GO A)) ((MEMBER (CAR R^1) TERMTABLE) (GO B))) (SETQ R^2 (REPL^2 (CAR R^1) V)) (SETQ R^4 (REPL^2 (CDR R^1) (CDR R^2))) (RETURN (LIST (CONS (CAR R^2) (CAR R^4)) (CDR R^4)))) A (COND ((BELONGS (CAR R^1) TERMTABLE) (GO B)) (SETQ R^3 (REPL^2 (CDR R^1) V)) (RETURN (LIST (CONS (CAR R^1) (CAR R^3)) (CDR R^3)))) B (SETQ R^3 (REPL^2 (CDR R^1) (CDR V))) (RETURN (LIST (CONS (CAR V) (CAR R^3)) (CDR R^3)))))

VALUE (BELONGS)
(LAMBDA (X Y) (COND ((NULL Y) F) ((EQUAL X (CAR (CDAR Y))) T) (T (BELONGS X (CDR Y)))))

VALUE (TRANSF)
(LAMBDA (X Y) (PROG NIL A (COND ((NULL Y) (RETURN X))) (SETQ X (CONS (CAR Y) X)) (SETQ Y (CDR Y)) (GO A)))
VALUE (INVTRAN)
(LAMBDA (N P) (PROG (V1 V2) (SETQ V1 (CADR N)) (COND (TRACEINV (PRINT (LIST (QUOTE INVTRANARGS) N P)))) A (COND ((NULL V1) (COND (TRACEINV (RETURN (PRINT V2)))) (T (RETURN V2))))))) (SETQ V2 (APPEND V2 (CHOOSE2 (CAR V1) P)))) (SETQ V1 (CDR V1)) (GO A))

VALUE (CHOOSE2)
(LAMBDA (M P) (COND ((NUMBERP M) (COND ((EQUAL M 1) (COND ((NOT (ATOM (CAR P))) (COND ((EQUAL (CAAR P) (QUOTE ANY)) (CDAR P)) ((EQUAL (CAAR P) (QUOTE COMP)) (CDAR P)) (T (LIST (CAR P)))))))) ((EQUAL (CAR P) (QUOTE EMPTY)) NIL) (T (LIST (CAR P)))))) (T (CHOOSE2 (SUB1 M) (CDR P)))))) (T (LIST M))))

VALUE (PUTFORCOMP)
(LAMBDA (X Y) (COND ((OR (EQUAL (CAR Y) (QUOTE COMP)) (AND (NOT (ATOM (CAR Y)))) (EQUAL (CAAR Y) (QUOTE COMP)))) (CONS (CONS (QUOTE COMP) X) (CDR Y))) (T (CONS (CAR Y) (PUTFORCOMP X (CDR Y)))))

VALUE (INVTRANS1)
(LAMBDA (N P) (PROG (V1 V2) (SETQ V1 (CADR N)) (COND (TRACEINV (PRINT (LIST (QUOTE INVTRANS1ARGS) N P)))) A (COND ((NULL V1) (COND (TRACEINV (RETURN (PRINT V2)))) (T (RETURN V2))))))) (SETQ V2 (APPEND V2 (CHOOSE1 (CAR V1) P)))) (SETQ V1 (CDR V1)) (GO A))

VALUE (CHOOSE1)
(LAMBDA (M P) (COND ((NUMBERP M) (COND ((EQUAL M 1) (COND ((AND (NOT (ATOM (CAR P))) (EQUAL (CAAR P) (QUOTE ANY))) (CDAR P)) ((EQUAL (CAR P) (QUOTE EMPTY)) NIL) (T (LIST (CAR P)))))) (T (CHOOSE1 (SUB1 M) (CDR P)))))) (T (LIST M))))
FOOTNOTES

1. By a generative grammar I mean any device that enumerates a set of sentences, i.e., a language, and assigns to each sentence one or more structural descriptions. The term generative grammar was used by Chomsky "to distinguish it from descriptive statements that merely present the inventory of elements that appear in structural descriptions and their contextual variants" [1].

2. Although most grammatical descriptions are not explicit about underlying models that formalize their assumptions, each of these types of grammars has at least been implicitly utilized as a means of grammatical description.

3. See, for example, references [1, 2, 3, 4, 5, 6].

4. These theorems concern the recognition problem for standard form grammars and the weak equivalence with preservation of ambiguity which holds between standard form and arbitrary context-free grammars.

5. Matthews [8, 9, 10] has given a pure analysis-by-synthesis transformational grammar recognition procedure, and we present in this thesis two distinct preanalysis algorithms for use with an analysis-by-synthesis procedure. Chomsky [11] and Evey [12] have given recognition procedures for context-free grammars (and hence finite state grammars), and numerous other context-free recognition procedures have been given. See reference [13] for an extended bibliography. Oettinger and Kuno [14] have given a recognition procedure for standard form context-free grammars,

6. By a context-sensitive grammar we will denote merely a phrase structure grammar consisting of a finite number of rules each of which rewrites a single symbol as a string of symbols in some neighborhood, stateable by means of a rule of the form $A \rightarrow X \, / \, Y \, _Z$

where $A$ is a single nonterminal symbol and $X$, $Y$, and $Z$ are strings of terminal and nonterminal symbols ($Y$ and $Z$ possibly empty). Garvin is not explicit about what he means by a context-sensitive grammar, so it is possible that he is referring to something other than the indicated modification of context-sensitive grammar as the term was originally defined and is generally used. If this is the case, our arguments are misdirected, but we would then disagree with Garvin's apparent assertion that it is impossible to describe language by any formal grammar consisting of a finite set of rules.

7. When we mentioned this procedure to Matthews he observed that it had already been suggested by Herzberger in the form of a personal communication. A direct method of incorporating context-sensitive restrictions into the recognition procedure has been given by Griffiths [15].

8. Exhaustive recognition of context-free natural language grammars, for example, has shown that this is a serious and so far unavoidable problem for these grammars [24, 25].

9. Our reasons for this include increased economy (which is related to the attainment of linguistic adequacy in better expressing significant
generalizations) and the ability of a tree-based transformational grammar to make available to the phonological and semantic components certain essential structural information.

10. I cannot tell from available reports whether or not Joshi permits the use of ordered rules. He does not preclude them, and hence he may be utilizing ordered semi-Thue productions.

11. It is possible, of course, to consider the establishment of necessary structure for transformational reversals to be one part of step (3), identification of transformations. If we so interpret step (3), and if, moreover, we restrict step (5), phrase structure determination, to finding all structure not previously determined by previous steps, then our stated objections to Fraser's breakdown of steps vanish. Fraser does not make it clear what he wishes to include in each step or how these steps might be carried out in detail.

12. Herzberger's paper was written while he was at MITRE in the summer of 1963, and it is included in the subsequent MITRE report [31] as a supplementary paper. Nevertheless, it suggests an approach which is distinct in several respects from the one which was eventually followed by the Language Processing Techniques Sub-department.

13. In the computer implementation of transformations # is represented by SENTB and X is represented by ANY.

14. In order to depict that a sequence of trees satisfies a term X or Y we have shown such a sequence as being connected to that X or Y.
Our definition of satisfies, of course, establishes no such structure. Our computer program implementation of inverse transformations discussed in Section 3.2, however, does temporarily make such connections in order to facilitate the movement of a sequence of trees as a unit.

15. Matthews reports [personal communication] that he has recently obtained more satisfying results by eliminating the use of permutation transformations in the description of Hidatsa. He does, however, presently make use of both daughter adjunction and daughter substitution elementary transformations.

16. See Section 3.6 for a treatment of how the rules of Q may be obtained.

17. By the semantic component of a transformational grammar we refer to a component of the type described by Katz and Fodor [32] and Katz and Postal [33], suitably modified to apply specifically to a syntactic component of the type that we assume for which the recursive power resides in the base subcomponent.

18. For example, when a level of embedding contains two embedded sub-levels, neither embedded within the other, and when, furthermore, the same binary transformation is performed in collapsing each sub-level, it is immaterial which is collapsed first, and we do not wish to have these two orderings reflected by two different structural descriptions.

19. Obligatory transformations, many whose use is signalled by a phrase structure marker symbol, need not be included in the lists of
transformations to be applied at each level. Hence, to the extent that obligatory transformations are thus included, such specifiers are redundant.

20. Keyser's example was of the following type: In analyzing the tree

\[ \begin{array}{c}
A \\
B \quad C \\
D \quad E \\
e \\
d \\
\end{array} \]

with respect to the structural index \((B, ((D)), X)\) the computer found both the proper analysis \((B, X)\)

\[ \begin{array}{c}
B \\
C \\
D \quad E \\
e \\
d \\
\end{array} \]

and \((B, D, X)\).

\[ \begin{array}{c}
B \\
D \\
E \\
e \\
d \\
\end{array} \]

The convention can well be made that if a proper analysis satisfying the optional term \(D\) is found, the alternative proper analysis which does not satisfy \(D\) is discarded. Using this convention, the ambiguity of analysis vanishes, of course. It is easy for us to incorporate this convention in the synthesis phase of our procedure, but our pre-analysis procedure assumes that both proper analyses are equally valid.

Less trivial examples of more than one proper analysis existing can also be found. Considering, for example, Grammar 2A of Appendix D we observe that the \(WH^1\) transformation can question more than one noun phrase, the ADJ transformation can apply in more than one way (for
example, in such a sentence as the bird that is early catches the worm that is juicy to give the early bird catches the juicy worm or the early bird catches the worm that is juicy), and the binary transformation RELEMB can apply to more than one combination of DET N NU COMP.

21. To give some idea of the magnitude in question, a computer counting once per microsecond (10^6 times per second) would take about 2.3 x 10^{52} centuries to reach this number. The predicted death of the sun is about 10^8 centuries away.

22. In particular, as a result of a rather extensive comparative study [13] we decided to use a CF recognition procedure related to one described by Irons [19] and referenced in the cited comparative study as the SBT (selective bottom to top) method. Evans of the Air Force Cambridge Research Laboratories (AFCRL) wrote the LISP program that accomplishes this CF recognition, and it is documented in Appendix D as the function PARSE and its associated sub-functions.

23. The recognition procedure thus obtained is somewhat closer to the approach suggested by Zwicky et al. [31] than is the procedure of Section 3.4. All of our procedures, however, make use of the inverse transformations of Section 3.2 which map sequences of trees into other sequences of trees, whereas Zwicky suggests using inverse transformations that map one tree into another.

24. For a description of this list processing programming language see the LISP 1.5 Programmer's Manual [32].
25. Fifteen to thirty seconds were required for one version of the program using that variant of the method of Section 3.4 in which the result of each inverse transformation is not immediately built up to \( S^1 \). As is shown in Appendix C, versions of other procedures took up to eight minutes for recognition.

26. The AFCRL M-460 LISP system is a product of T. Evans and T. Hart of the Air Force Cambridge Research Laboratories (AFCRL). It was designed to be used in an on-line fashion, and it contains a wealth of features such as on-line editing, break point control, and selective tracing which are enormously effective aids to debugging LISP programs. These features, coupled with the fortunate extended availability of large blocks of time on the M-460 computer and the existence of a context-free recognition program written by T. Evans, made possible the writing and debugging of a rather crude preliminary version of one of the recognition procedures of this thesis in about three months. Additional modifications and testing of various grammars extended over another nine months. The convenience of the M-460 LISP system was not properly appreciated until efforts were made to transfer the working LISP program to the 7090 LISP system. Operating on a one run per day basis, it took longer to produce a working program than it had originally taken to write and debug the first version of that program. The difficulties encountered consisted chiefly of clerical errors in keypunching and run submission, with a few esoteric incompatibilities between the two LISP systems thrown in for good measure.
27. It is a moot point whether the sentence the book is old on the table is any less acceptable than John likes the old book (when he is) on the table. We have rather arbitrarily taken steps to preclude the former but not the latter. Clearly our primitive grammar is far too crude with respect to the way it fails to restrict certain constructions to allowable environments. The point we wish to stress, however, is that our recognition routine makes explicit the interpretations that a grammar assigns to a sentence and points out inadequacies of the structure assigned by that grammar and inadequacies of the sentences specified by that grammar.
BIBLIOGRAPHY


General Rewriting Systems," Proceedings IEEE, Symp. Switching

Grammar in the Automatic Parsing of Natural Languages," 1965 International Conference on Computational Linguistics,
New York, New York.

Bedford.


60," Comm ACM, 51.

Languages whose Syntax is Expressible in Extended Backus
Normal Form," Proceedings of the Int. Symp. on Symbolic
Languages in Data Processing, Rome, Italy, pp 741-758.


22. Ledley, R., and Wilson, J., (Mar 62). "Automatic-Program-
ing Language Translation through Syntactic Analysis," Comm.
ACM 5, 145.


183
BIographical Note

Stanley R. Petrick was born in Cedar Rapids, Iowa on August 16, 1931. He attended the Roosevelt High School and received a B. S. degree in Mathematics from the Iowa State University in 1953. During 1953 to 1955 he attended MIT while on active duty as an Air Force officer and received the S. M. degree from the Department of Electrical Engineering in 1955. He was elected to Sigma Xi in 1955.

Mr. Petrick has been associated with the Applied Mathematics Branch of the Data Sciences Laboratory at the Air Force Cambridge Research Laboratories since 1955 and his recent studies at MIT have been partially supported by AFCRL. During 1959-1962 he held the position of Lecturer in Mathematics in the Evening Graduate Division of Northeastern University.


His publications include:


On The Use of Boolean Functions in Pattern Recognition, IRE International Symposium on Information Theory, Brussels, Belgium, September 1962.


