ON INDUCING A NON-TRIVIAL, PARSIMONIOUS, HIERARCHICAL GRAMMAR FOR A GIVEN SAMPLE OF SENTENCES

John Reed Koza

January 1973

Technical Report No. 142

with assistance from:

National Science Foundation
Grant No. GJ-29989X
Washington, D.C.
ABSTRACT

ON INDUCING A NON-TRIVIAL, PARSIMONIOUS, HIERARCHICAL GRAMMAR
FOR A GIVEN SAMPLE OF SENTENCES

by
John Reed Koza

Co-Chairpersons: Bruce Arden
                John Holland

The thesis presents an algorithm which, for a given sample of
sentences, induces a grammar that (1) can generate the given sample
of sentences, (2) is non-trivial in the sense that it does not merely
enumerate the sentences of the sample, (3) is hierarchical in the sense
that the sentences of the sample are derived through relatively long
sequences of sentential forms, (4) is, at the same time, parsimonious
in the sense that the grammar contains a relatively small number of
rules of production, (5) contains recursive rules of production under
appropriate specified conditions, and (6) contains disjunctive rules
of production (and generalizations) under appropriate specified condi-
tions.

The thesis formally defines the notion of a regularity in a sample
of sentences, presents a combinatorial algorithm for searching for regu-
larities, and argues that searching for local regularities is an
appropriate basis for a grammar discovery algorithm.

The Grammar Discovery Algorithm presented involves (1) a search
of the given sample of sentences for local regularities—this search
being an exhaustive process within a limited range, (2) the considera-
tion non-exhaustively of various possible transformations of the given sample
via grammatical rules of production that are developed from the local
regularities, (3) a selection process among such transformations,
and (4) the reapplication of these steps until the sample is fully

resolved and a grammar induced.

The thesis contains and describes a computer program implementing the Grammar Discovery Algorithm.

While the emphasis is on grammar discovery, there is also discussion on the equivalent problem of inducing axioms and rules of inference in a formal system.

The Grammar Discovery Algorithm is justified in terms of its internal consistency, its satisfying of various heuristic criteria and by examples.
ACKNOWLEDGMENT

I thank Professor Bruce Arden who, as co-chairperson of this thesis committee, gave of his time, ideas, and information in writing this thesis; Professor John Holland who, as co-chairperson, originally suggested this topic in his seminar in adaptive systems theory and gave aid and comfort to this thesis throughout; Professor Eugene Lawler (now removed to the University of California, Berkeley, and hence not now a member of the committee) who suggested several of the optimizing features of the Grammar Discovery Algorithm; Associate Professor Bernard Zeigler who was helpful at various stages; Professor Walter Reitman who was helpful in the writing of this thesis and particularly the sections justifying the Algorithm; and to Professor Larry Flanigan who, on very little notice, served as a proxy and replacement for Professor Zeigler on the committee, while the latter was on sabbatical in Israel.

I also thank Monna Whipp of the Logic of Computers Group for typing and preparing this thesis for publication as a report of the Logic of Computers Group.

And, in general, I thank the Department and Faculty for tolerating the glacial advance of this research project.

John Koza

Ann Arbor, Michigan
December, 1972
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgment</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Notation</td>
<td>vi</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>A. STATEMENT OF THE PROBLEM</td>
<td>1</td>
</tr>
<tr>
<td>B. REVIEW OF PREVIOUS WORK IN THE FIELD</td>
<td>7</td>
</tr>
<tr>
<td>C. PRINCIPAL CLAIMS OF THIS THESIS</td>
<td>9</td>
</tr>
<tr>
<td>D. OVERVIEW OF THIS THESIS</td>
<td>10</td>
</tr>
<tr>
<td>II. DESCRIPTION OF THE GRAMMAR DISCOVERY ALGORITHM</td>
<td></td>
</tr>
<tr>
<td>A. PRESENTATION OF THE SAMPLE</td>
<td>17</td>
</tr>
<tr>
<td>B. THE SEARCH PHASE -- FINDING REGULARITIES IN A SAMPLE OF SENTENCES</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION TO THE SEARCH PHASE</td>
<td>19</td>
</tr>
<tr>
<td>2. CYLINDER SETS IN A SAMPLE</td>
<td>20</td>
</tr>
<tr>
<td>3. MASKS</td>
<td>23</td>
</tr>
<tr>
<td>4. TYPES OF MASKS</td>
<td>25</td>
</tr>
<tr>
<td>5. CONTEXT-SUB-SEQUENCES, PREDICTED-SUB-SEQUENCES, AND DOMAIN SEQUENCES</td>
<td>27</td>
</tr>
<tr>
<td>6. DEFINITION OF A REGULARITY</td>
<td>28</td>
</tr>
<tr>
<td>7. TYPES OF REGULARITIES</td>
<td>30</td>
</tr>
<tr>
<td>8. RATIONALE FOR SEARCHING FOR LOCAL REGULARITIES</td>
<td>31</td>
</tr>
<tr>
<td>9. ALGORITHM OF THE SEARCH PHASE</td>
<td>35</td>
</tr>
<tr>
<td>10. EXAMPLES</td>
<td>39</td>
</tr>
<tr>
<td>C. THE RECODING PHASE-- DEVELOPING RULES OF PRODUCTION FROM REGULARITIES</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION TO THE RECODING PHASE</td>
<td>45</td>
</tr>
<tr>
<td>2. TRANSFORMATIONS (RECODINGS)</td>
<td>46</td>
</tr>
<tr>
<td>3. DEVELOPMENT OF RULES OF PRODUCTION FROM LOCAL REGULARITIES</td>
<td>48</td>
</tr>
<tr>
<td>4. THE RECODING PROCEDURE</td>
<td>52</td>
</tr>
<tr>
<td>5. EXAMPLE</td>
<td>56</td>
</tr>
<tr>
<td>D. THE SELECTION PHASE -- SELECTING RECODINGS</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION TO THE SELECTION PHASE</td>
<td>60</td>
</tr>
<tr>
<td>2. ENTROPY, PARSIMONY, RECURSIVE PARSIMONY OF A TRANSFORMATION</td>
<td>61</td>
</tr>
<tr>
<td>3. THE $p_{ii}$-GRAPH OF A TRANSFORMATION</td>
<td>68</td>
</tr>
<tr>
<td>4. RESOLVING TRANSFORMATIONS</td>
<td>73</td>
</tr>
<tr>
<td>5. TRIAL RECODINGS AND THE SELECTION OF THE ACTUAL RECODING</td>
<td>75</td>
</tr>
</tbody>
</table>
E. INDUCING RECURSIONS FROM A FINITE SAMPLE
   OF SENTENCES
   1. INTRODUCTION AND MOTIVATION 84
   2. DEFINITION OF A RECURSION 86
   3. APPROACHES TO INDUCING RECURSIONS 87
   4. SENTENCE-ORIENTED METHOD OF INDUCING
      RECURSIONS 90
   5. RULE-ORIENTED METHOD OF INDUCING RECURSIONS 94
F. INDUCING DISJUNCTIONS AND GENERALIZATIONS
   1. INTRODUCTION AND MOTIVATION FOR INDUCING
      GENERALIZATIONS 100
   2. INDUCING GENERALIZATIONS COMBINATORIALLY
      (RULE-ORIENTED METHOD) 101
   3. INTRODUCTION AND MOTIVATION FOR INDUCING
      DISJUNCTIONS 104
   4. INDUCING DISJUNCTIONS USING ENTROPY
      (SENTENCE-ORIENTED METHOD) 106
G. THE TERMINATION PROCEDURE 111
H. CHOICE OF LIMITS ON LENGTH OF REGULARITIES
   IN THE SEARCH PHASE 113
I. CHOICE OF PARAMETERS IN THE RECODING PROCEDURE 116
J. EXTENSION OF THE GRAMMAR DISCOVERY ALGORITHM
   TO SAMPLES OTHER THAN LINEAR SEQUENCES OF
   SENTENCES 119

III. THE SELF-PUNCTUATING NATURE OF THE GRAMMAR DISCOVERY
     ALGORITHM
     A. INTRODUCTION 123
     B. EFFECT OF CHOICE OF LIMITS ON LENGTH OF
        REGULARITIES IN THE SEARCH PHASE 124
        1. INITIAL DISCUSSION 124
        2. EXTENSIONS OF A REGULARITY 125
        3. DEFINITION OF A MAXIMAL REGULARITY 127
        4. PERSISTENCE OF MAXIMAL REGULARITIES
           AFTER RECODINGS 128
     C. EFFECT OF CHOICE OF PARAMETERS IN THE RECODING
        PROCEDURE 134
     D. EFFECT OF PRESENCE OR ABSENCE OF INITIAL
        PUNCTUATION IN THE SAMPLE 136

IV. ADDITIONAL FEATURES OF THE GRAMMAR DISCOVERY ALGORITHM
     A. TERNARY MASKS AND DON'T CARE CONDITIONS 139
     B. ALPHABET-ENLARGING VERSUS HUFFMAN-TYPE RECODING 142
     C. RECODINGS WITH NOISE IN THE SAMPLE 144
     D. CONTEXT-FREE CASE 145

V. EXAMPLES 148

VI. FUTURE DIRECTIONS 180
APPENDIX A: DESCRIPTION OF INPUT TO THE COMPUTER PROGRAM
IMPLEMENTING THE GRAMMAR DISCOVERY ALGORITHM 182

APPENDIX B: LISTING OF THE COMPUTER PROGRAM IMPLEMENTING
THE GRAMMAR DISCOVERY ALGORITHM 193

BIBLIOGRAPHY 245
### TABLE OF NOTATION

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Starting set (axioms) of general derivation system.</td>
</tr>
<tr>
<td>$V_T$</td>
<td>Terminal Alphabet of grammar or general derivation system.</td>
</tr>
<tr>
<td>$V_N$</td>
<td>Non-Terminal Alphabet of grammar or general derivation system.</td>
</tr>
<tr>
<td>R</td>
<td>Rules of Production of grammar or general derivation system.</td>
</tr>
<tr>
<td>$\mathcal{G} = &lt;V_T, V_N, R, A&gt;$</td>
<td>General Derivation System.</td>
</tr>
<tr>
<td>S</td>
<td>Starting Symbol of grammar.</td>
</tr>
<tr>
<td>$\mathcal{G} = &lt;V_T, V_N, R, S&gt;$</td>
<td>A Grammar.</td>
</tr>
<tr>
<td>$C_1$</td>
<td>Number of symbols in alphabet of the Sample ($V_T$).</td>
</tr>
<tr>
<td>$Y$</td>
<td>The sample of sentences. A sequence over $V_T$ or the current alphabet.</td>
</tr>
<tr>
<td>$T$</td>
<td>Indexing parameter for sample.</td>
</tr>
<tr>
<td>N</td>
<td>Length of Sample $Y$.</td>
</tr>
<tr>
<td>$N_{max}$</td>
<td>Length of longest sentence in $Y$. (If there is no initial punctuation in the sample, $N_{max} = N$).</td>
</tr>
<tr>
<td>$V_c$</td>
<td>The current Alphabet. $V_T$ is the current alphabet for level one.</td>
</tr>
<tr>
<td>C</td>
<td>Number of symbols in current alphabet (that is, $C_1$ plus non-terminal symbols added). $C = C_1$ for level one.</td>
</tr>
<tr>
<td>M</td>
<td>Length of Sequence of symbols.</td>
</tr>
<tr>
<td>g,h</td>
<td>Indices typically used to index positions in a regularity, the context part or predicted part of a regularity.</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>The value $1/\varepsilon g$.</td>
</tr>
<tr>
<td>$P, P_{ij}$</td>
<td>Conditional Probabilities.</td>
</tr>
<tr>
<td>$%$</td>
<td>Symbol meaning &quot;predicted&quot;.</td>
</tr>
<tr>
<td>_</td>
<td>Symbol meaning &quot;context&quot;.</td>
</tr>
<tr>
<td>#</td>
<td>Symbol meaning &quot;Don't Care&quot;.</td>
</tr>
</tbody>
</table>
$M(I)$ Mask—Sequence of Length M over symbols "§", "_", and possibly also "#" (if mask is ternary).

$\gamma(I)$ Context-Sub-Sequence—Sequence of Length M over current alphabet augmented by "§" and possibly also "#".

$\phi(I)$ Predicted-Sub-Sequence—Sequence of Length M over current alphabet, augmented by "_" and possibly also "#".

$R = \langle \gamma, \phi, P, Y \rangle$ A regularity in sample Y whose context-sub-sequence is $\gamma$, whose predicted-sub-sequence is $\phi$, and whose conditional probability is $p$.

$\Delta$ The domain sequence of a regularity.

$T$ A transformation.

$\lambda$ Coefficient of parsimony.

$\lambda_r$ Coefficient of Recursive Parsimony.

$n$ Number of Rules of Production.

$n_r$ Number of Recursive Rules of Production.

. The period (Initial punctuation mark).
I. INTRODUCTION

A. STATEMENT OF THE PROBLEM

The processes of deduction and induction are fundamental in mathematics and the axiomatized sciences. These same two processes also appear in the study of generative grammars.

We define a general derivation system $G = \langle V_T, V_N, R, A \rangle$ to be a system in which $V_T$ is the set of terminal symbols, where $V_N$ is the set of non-terminal symbols, where $R$ is a set of rules, and where $A$ is the starting set. The starting set contains strings of terminal and non-terminal symbols. The rules are pairs of symbol strings. The first string, called the antecedent (or simply the left side), is a non-empty string of terminal symbols and/or non-terminal symbols. The second string of the pair, called the consequent (or simply the right side), is a string of terminal symbols and/or non-terminal symbols, or is empty.

A derivation system generates sentences by starting with any one of the strings in the starting set, and by applying the rules, one after the other. Each application of a rule consists of substituting the consequent sub-string for the antecedent sub-string. Each symbol string so produced is, in general, a string of terminal symbols and non-terminal symbols, and is called a sentential form. A sentential form consisting only of terminal symbols is called a sentence.

A generative grammar is a derivation system in which the starting set $A$ consists only of a single distinguished non-terminal of length one. We customarily call this symbol "S", the starting symbol, and we
call the rules of a grammar rules of production.

A formal system consists of two derivation systems. First, there is a set $V_T$ of symbols of discourse about the subject at hand, and a set of formation rules $R_F$ for the formation of well-formed sentences over $V_T$. This derivation system is a grammar and has a starting symbol S and whatever non-terminal symbols are needed to express the rules $R_F$. The second derivation system has a subset A of the well-formed sentences of the first derivation system as its starting set. This subset consists of the accepted truths of the system, called axioms. There is also a set of truth-preserving rules $R_I$, called rules of inference, for transforming one true, well-formed sentence into another. By recursively applying the axioms and the rules of inference, one derives the true sentences $V_T$ of the formal system, called theorems. The non-terminal symbol set of this second derivation system is typically the same as its terminal symbol set. Thus, any sentential form of this second derivation system is a sentence and also a theorem.

The process of generating sentences in a derivation system is called derivation. The process of generating theorems in a formal system is called deduction. In both cases, $V_T, V_N, R$, and A are given. Typically, these two processes present themselves in the setting of finding the derivation, if any, that can generate a particular given sentence. For grammars, this problem is the parsing problem. For formal systems, it is well to remember that all theorem proving is merely a parsing problem. Parsing and theorem-proving algorithms are either bottom-up in the sense that they start with the given sentences and consider possible sentential forms from which these sentences can be immediately derived, or top-down in the sense that the derivation starts with the starting set A.
Induction is the inverse process for deduction. In induction, one begins with an observed, limited set $Y$ of the true sentences $Y_T$. The Copernican doing induction tries to discover a parsimonious set of axioms $A$ and a set of rules of inference $R_I$ which together could have generated the given set $Y$.

Grammar discovery is the inverse process for derivation in a grammar. In the grammar discovery problem, one starts with a set $V_T$ of $C_1$ terminal symbols and a set $Y$ of sentences over this $V_T$. $Y$ may be presented, for example, as a concatenation of the individual sentences, with or without initial punctuation between the sentences to identify the sentences.

The problem we discuss herein is the problem of finding a non-trivial, parsimonious, hierarchical grammar $G = \langle V_T, V_N, R, "S" \rangle$ which could have generated the given sentences $Y$. This problem requires finding a set of rules $R$ of production and a suitable set of non-terminal symbols $V_N$.

Grammar discovery and induction are necessarily bottom-up processes.

In what follows, the main emphasis will be on the grammar discovery problem, since this is the underlying problem common with the induction problem for formal systems.

When we say that we are seeking a non-trivial grammar, we are referring to any grammar except a grammar which merely catalogs the sentences of the sample $Y$—an example of which is the grammar in which the starting symbol "S" may be rewritten as the disjunction of the observed sentences of $Y$. Note that the adjective "non-trivial" applies to this well-defined, definite enumerative situation. When we say that we are seeking a hierarchical grammar, we are referring to a
grammar which, as a minimum, is non-trivial, but which has the additional quality of generating its sentences through relatively long sequences of sentential forms. Note that the adjectives "hierarchical" is less precise and more connotative than the adjective "non-trivial". The connotation is that each small phrase and substructure of a sentence is generated by a separate application of a rule of production—in contrast to a situation in which large phrases are generated by one application of one rule. When we say that we are seeking a parsimonious grammar, we are referring to a grammar with a relatively small number of rules of production. Note that the adjective "parsimonious" is also a connotative word. The connotation is that a few rules of production are each applied many times in the generation of sentences—in contrast to a situation in which many rules are each used only in a few specific cases. A hierarchical grammar without recursions tends to be unparsimonious. A grammar may be parsimonious without being hierarchical—for example, if it is trivial. A grammar may be hierarchical without being parsimonious, as, for example, a grammar which fails to employ recursions, or employs superfluous non-terminal symbols or superfluous rules of production.

Note that we state the grammar discovery problem as a problem of finding a parsimonious and a hierarchical grammar, but not necessarily the most parsimonious and most hierarchical grammar—which are qualities that are not defined here. Similarly, for lack of definition, we do not require the finding of a necessarily interesting grammar, nor a significant grammar, nor a linguist's grammar, nor a grammar conforming to some known or supposed physiological or psychological model, nor the "best" grammar.

The grammar discovery problem, of course, admits of several
variations, and some of these variations have appeared within the very limited literature on grammar discovery and induction. For example, the machine alleging to solve the grammar discovery problem may receive more than a fixed sample of sentences. The machine may, for example, receive, or ask to receive, answers to certain questions in order to assist it in solving the problem. These questions may take the form of asking whether certain new sentences which it constructs are in the language produced by the grammar which is the desired solution to the problem. Or, non-sentences identified as such may be introduced with the sample sentences. Or, the machine may ask for a larger sample of sentences or identified non-sentences if it needs them (Gold, 1967). Then again, the machine may be assumed to be receiving a continuing, infinite input of sample sentences (Goodall, 1962). Another possibility is that the machine can receive, or ask to receive, certain cues, such as the number (or an upper bound on the number) or internal states which an automaton of a certain type would have to have in order to accept the language generated by the grammar which is the solution to the problem (Pao, 1969). Or, the machine may be told that the given sample contains all the sentences of the language (as in Gold's "text" method of presentation) or that the sample contains sentences whose production requires the use of all the transitions in a particular automaton (Pao, 1969). Or, the sample may be assumed to be structured in various ways—for example, the sample may be assumed to contain (or tend to contain) all of the sentences of the language whose length is not longer than the longest sentence of the sample (Feldman, 1969).

We deal here, however, with what seems to be the most basic and general problem—the kind of grammar discovery problem one would
encounter in deciphering an unknown message from outer space, or in
decoding an ancient language from a given scroll, or in finding the
substructures in a coded message—that is: given a fixed, finite
sample of sentences and no other information of any kind (that is,
no other information about the source of the same; no information
about the type of grammar producing the sentences; no information
about the statistical properties or completeness of the sample; no
non-sentences; no cues; and no information in response to queries
or requests) find a non-trivial, hierarchical, parsimonious grammar
that could have generated the sentences of the sample.
B. REVIEW OF PREVIOUS WORK IN THE FIELD

The literature in the field of induction is very small.

Solomonoff (1964) treats the problem as a special case of sequence extrapolation using enumerative methods which are not readily extendable, and which are highly dependent on such artifacts as order of sentences within the sample.

Gold (1967) proposed a theory of language learning in which the learner receives sequences of correct sentences of an unknown language and successively guesses grammars of specified types. If the guessing converges to one correct grammar, the language has been "identified in the limit." Gold proves several decidability results concerning language identifiability in the limit, and proposes several categorizations of language learning models, and methods of presenting information to such learning models.

Feldman (1969) proposes various concepts of grammatical complexity and proposes approaches to inducing a least complex grammar for a given sample. Feldman (1969) includes a heuristic method of merging symbols of the alphabet to achieve recursive rules from a finite sample.

Pao (1969) deals with the induction of finite state languages and "delimited" languages—a subset of context-free languages. Pao specifies conditions on the sample to guarantee a solution; and then, given certain cues, the algorithm constructs an automaton, evaluates it, and modifies it until a solution is attained.

Caianello (1970) discusses the Procrustes Algorithm for feature extraction and form analysis of language, and uses semigroup homomorphisms similar to what are called "transformations" or "recodings" herein.
In a highly suggestive, short paper, Goodall (1962) discusses the problem of induction from the point of view of logical types, and the Godel Incompleteness Theorem. Goodall develops a probabilistic automaton for a given ongoing infinite sequence of input symbols. In this paper, Goodall suggests, without specifying, the idea of recoding the given sample into a new "higher level" sample using "resolving transformations" that optimize the conflicting qualities of "entropy" (information) and "parsimony." This thesis started as an attempt to attribute some consistent and reasonable meaning to these suggestive terms and other metaphorical observations in the Goodall paper, but no claim is made that the definitions and constructions herein of these same terms in any way correspond to that intended by Goodall.

In addition to work in the immediate field of induction, there is, of course, a large literature in the fields of pattern recognition (e.g. Uhr); cryptography (e.g. Smith and Gaines); picture analysis; sequence forecasting and correlation, including continuous functions are numerical sequences.
C. PRINCIPAL CLAIMS OF THIS THESIS

In this thesis, we formalize the notion of a regularity in a sample of sentences, present a combinatorial algorithm for searching for regularities, and argue that searching for local regularities is a suitable beginning for a grammar discovery algorithm—an approach not found in the literature.

We then present the Grammar Discovery Algorithm which, for a given sample of sentences, induces a grammar that (1) can generate the given sample, (2) is non-trivial in the sense that it does not merely enumerate the sentences of the sample, (3) is hierarchical in the sense that the sentences of the sample are derived through relatively long sequences of sentential forms, (4) is, at the same time, parsimonious, in the sense that the grammar contains a relatively small number of rules of production, (5) contains recursive rules of production under appropriate specified conditions, and (6) contains disjunctive rules of production (and generalizations) under appropriate specified conditions. We do not claim that the demonstratably non-trivial, hierarchical, and parsimonious grammar that we seek and find here is an interesting grammar, is a significant grammar (in the sense, perhaps, that it conforms to some known or supposed physiological or psychological model), is a "linguist's" grammar, or is the "best" grammar. Rather, we only claim that (1) the grammars we seek and find here do possess the above three qualities and that these qualities are reasonable objectives for a grammar discovery algorithm; (2) the grammar discovery algorithm herein satisfies certain heuristic criteria; and (3) the Algorithm handles a variety of interesting examples that indicate it is not inherently limited in its capacity.
D. OVERVIEW OF THIS THESIS

Part II of this thesis contains a description of the Grammar Discovery Algorithm.

In II. A., we discuss the method of presenting the sample of sentences of the Grammar Discovery Algorithm. The Grammar Discovery Algorithm begins with a sample of sentences $Y$ produced by an unknown grammar $G = \langle V_T, V_N, R, "S" \rangle$. The sentences are presented as a concatenation of sentences, with or without initial punctuation (for example, periods) between the sentences to identify the sentences. The terminal alphabet $V_T$ is also given—or can be trivially inferred from the sample. The goal is to find a parsimonious, non-trivial, hierarchical grammar for the given sample of sentences—that is, to find a set $R$ of rules of production, and whatever non-terminal alphabet symbols $V_N$ are needed to write the rules $R$. One of the non-terminal symbols in $V_N$ will be designated as the starting symbol ("S") of the grammar.

The Grammar Discovery Algorithm is divided into three parts: the Search Phase, the Recoding Phase, and the Selection Phase.

In II. B., we describe the first main phase of the Grammar Discovery Algorithm—namely, the Search Phase. In II. B. 1 through II. B. 6, we formally develop and define the idea of a regularity in a sample.

Informally, a "regularity" exists whenever the occurrence of certain symbols somewhere in a sentence in the sample implies that certain other symbols occur elsewhere in the sentence with a conditional probability greater than mere chance.

In the Search Phase, one is interested in finding regularities in the given sample of sentences. The search is for local, not
global, regularities. It is not obvious that any grammar discovery algorithm should begin with a search for regularities, but an argument will be given to justify this search (II. B. 8). The regularities will also be categorized by type in a manner suggestive of the various Chomsky types of grammars (e.g. context-sensitive unrestricted rewrite, etc). (II. B. 7) The grammatical types of the rules production that ultimately will be induced will parallel the types assigned to the regularities.

We present the algorithm of the Search Phase in II. B. 9, and give examples (II. B. 10).

The Search Phase is an exhaustive, combinatorial process—within the limits imposed by the principle of searching only for local regularities.

The Search Phase ends with the generation of various tables of statistics about the various regularities discovered.

The second main phase of the Grammar Discovery Algorithm is the Recoding Phase. In the Recoding Phase, the local regularities found in the Search Phase are used to transform (recode) the given sample of sentences into a new set of sentences. This recoding is specified by (1) a set of grammatical rules of production, which specify which substitutions are to be made, and by (2) a Recoding Procedure, which specifies when and where these substitutions are to be made. The grammatical rules of production are developed from the local regularities discovered in the Search Phase. These rules ultimately become the rules of production of the induced grammar. The Recoding Procedure is essentially a punctuating process that partitions the given sample into appropriate phrases and substructures. The Recoding Procedure operates so as to minimize the introduction of rules of pro-
duction into the induced grammar. For example, before a new rule of production is introduced into the induced grammar, the algorithm considers the possibility of instead introducing one recursive rule (to replace one existing rule and the proposed new rules) as well as the possibility of re-using the new rule alone or in conjunction with other existing rules. The Recoding Phase is a non-exhaustive process, in contrast to the Search Phase.

In II. C., we describe this Recoding Phase. We define the notion of a transformation (recoding) in II. C. 2. We show how to develop grammatical rules of production from local regularities in II. C. 3, and show the connection between the grammatical type of the rule and the mask of the local regularity from which the rule was developed. In II. C. 4, we describe the Recoding Procedure—which, with the rules of production, defines the transformation (recoding).

The new set of sentences resulting from a recoding contains most of the information and structure of the given sample of sentences. Because structural regularities found in the original sample are replaced with short markers, what were originally global regularities in the original sample tend to become near-by local regularities in the new set of sentences. These short markers ultimately become the non-terminal symbols of the induced grammar.

The original given sample of sentences is regarded as the set of sentences of the first level in the Grammar Discovery Algorithm. These sentences are composed entirely of terminal symbols. The set of sentences resulting from the recoding then becomes the sentences of the next level of the process. These sentences contain non-terminal symbols as well. These sentences will now be searched for their local regularities, and another recoding performed. The alternate
application of first the Search Phase, and then the Recoding Phase
is what produces the non-trivial, hierarchical quality of the grammar
that is developed.

The third main phase of the Grammar Discovery Algorithm is the
Selection Phase. After the Search Phase is completed, several
(but not all) possible recodings of the sentences are considered.
The Selection Phase is a non-exhaustive process that oversees these
several trial recodings and selects one recoding which is then
actually applied to the sample to produce one new set of sentences.
This selection is made on the basis of three criteria: the entropy,
parsimony, and recursive parsimony of the transformation, the
parsimonious quality of the induced grammar derives from this se-
lection from among the various possible recodings. Upon completion
of this actual recoding, the 3 main phases are repeated, using the
new set of sentences, instead of the original sample. The processes
continue until the conditions of the Termination Procedure are
fulfilled, thus ending the grammar discovery algorithm.

The result is a set of rules of production R, and a non-terminal
alphabet $V_N$ with which to express them, and a starting symbol "S".
Together with the given terminal alphabet $V_T$, these elements will
constitute a grammar $G = \langle V_T, V_N, "S", R \rangle$ that could have produced
the given sample of sentences.

In II. D., we describe this Selection Phase. We define the
entropy, parsimony, and recursive parsimony of a transformation in
II. D. 2. We define the notion of resolving transformation in II. D. 4.
We graph the conditional probabilities of the regularities of the
transformation in II. D. 3. And, in II. D. 5, we describe the pro-
cess of making several trial recodings (each parameterized by the
length of the longest regularity used), and the algorithm for selecting the actual recoding.

In II.E., we describe the process of inducing recursions from a finite sample of sentences. In II.E.1, we first discuss the motivation for inducing recursions—namely, the ability of the induced grammar to generate an infinity of sentences. Without recursions, there can only be a finite number of sentences generated by a grammar. We define recursions in II.E.2, and discuss different possible approaches to inducing recursions in II.E.3. The different approaches can be divided according to whether they operate on the sentences of the sample or on the rules of production of the induced grammar. In II.E.4, we present the sentence oriented method of inducing recursions. This method has limited application. Finally, in II.E.5, we discuss the rule-oriented method of inducing recursions.

In II.F., we discuss the related problems of inducing generalizations and inducing disjunctions. The discussion parallels the discussion of recursions. In II.F.1, we present the motivation for inducing generalized rules when the sample is a formal system. In II.F.2, we present a combinatorial rule-oriented method for inducing generalizations. Then, in II.F.3, we turn to disjunctions. We discuss the motivation for inducing disjunctive (non-deterministic) rules of production in grammar induction—namely, the ability of the induced grammar to generate a variety of essentially different sentence types. Without disjunctions, there can only be essentially one sentence type generated by the grammar. In II.F.4, we present a sentence-oriented method for inducing disjunctions that is based on finding ensembles of substitution instances that have maximal or near-maximal entropy. We conclude this section with a discussion of the similarity of the problem of
inducing generalizations and inducing disjunctions.

In II.G, we discuss the Termination Procedure and Convergence of the Algorithm.

In II.H, we discuss the heuristic considerations for choosing the upper limits on lengths of regularities to be searched for in the Search Phase.

In II.I, we discuss the various parameters of the Recoding Procedure.

In II.J, we elaborate on the idea that the Grammar Discovery Algorithm presented here for linear sequences of symbols is applicable to a broad variety of induction and pattern recognition problems presented in different formats. This section also serves to extend the definitions and concepts of the Grammar Discovery Algorithm by providing illustrations from a simple two-dimensional pattern recognition problem.

In II.K, we present a variety of examples of the Algorithm.

Throughout part II, we point out how the basic features of the Grammar Discovery Algorithm—namely, its Search Phase involving local regularities, the Recoding Phase, and the Selection Phase—work to produce a non-trivial, hierarchical, and parsimonious induced grammar.

In Part III, we discuss the self-punctuating nature of the Grammar Discovery Algorithm. In II.B.2, we define the notion of extending a regularity, of a regularity being preserved under extension, and of a maximal regularity (III.B.2). In III.B.4, we argue that a maximal regularity will usually not be missed or lost by an unfortuitous choice of limits on the length of regularity in the Search Phase.

In III.C, we argue that the choice of parameters in the Recoding
Phase and the combinatorial incompleteness of the Recoding Phase does not usually cause the Algorithm to miss maximal regularities in the sample—unless it finds equally desireable maximal regularities. In III.D., we make the observation (quite contrary to intuition) that the presence or absence of initial punctuation between the individual sentences of a sample does not qualitatively or even quantitatively (combinatorially) complicate grammar induction.

In Part IV, we discuss several additional features of the Grammar Discovery Algorithm, including ternary masks and "don't care" conditions (IV.A); alphabet-enlarging versus the alternative (Huffman) type recodings (IV.V); recodings with noise in the sample and what noise represents (IV.C); and the generation of contest-free rules of production by an algorithm which appears to be inherently context-sensitive (IV.D).

In Part V, we discuss the justification for the Algorithm.

In the appendices, we describe the input to the Fortran IV computer program implementing the Grammar Discovery Algorithm (Appendix A); we exhibit the program in Appendix B; and we present additional examples in Appendix C.
II. DESCRIPTION OF THE GRAMMAR DISCOVERY ALGORITHM

A. PRESENTATION OF THE SAMPLE

Let \( V_T \) be a set of \( C_1^* \) symbols called the \textit{terminal alphabet}.

The Grammar Discovery Algorithm is said to operate in one of two modes depending on whether initial punctuation is present in the sample. We say that the Grammar Discovery Algorithm is operating in the \textit{first mode} if there is no initial punctuation present in the sample—that is, the sample \( Y \) consists of one long sequence of terminal symbols

\[ Y = Y(1), Y(2), \ldots, Y(N) \]

in which the "sentences" of the sample are concatenated together.

We say that the Grammar Discovery Algorithm is operating in the \textit{second mode} if \textit{initial punctuation} is present between the sentences of the sample. This initial punctuation, if present, is conventionally the period. In this mode, the sample appears as a string of symbols (a sentence) followed by a period, another string of symbols (another sentence), another period, etc.

Thus, the \textit{sample} is a sequence of symbols over the terminal alphabet, with or without initial punctuation between the individual

---

*Mathematical variables which are capitalized herein refer to variables that appear in the computer program implementing the Grammar Discovery Algorithm.

**Mathematical vectors which appear in the computer program implementing the Grammar Discovery Algorithm are written herein in the parenthesized style common in computer languages.

***The variable MODE in the computer program specifies whether the Grammar Discovery Algorithm is operating in the first mode or in the second mode.
sentences. The sample consists of only this symbol sequence. No additional information of any kind is given. No assumptions as to the statistical properties or completeness of any other aspect of this sample are made. Note that this method of information presentation is neither Gold's "text" method (in which the sample is presumed to contain every sentence of the language!) nor Gold's "informant" method (in which non-sentences appear) (Gold, 1967).

The presentation of the sample Y as a sequence of symbols, rather than as a square, rectangle, tree, graph, or other structure or raster containing symbols conforms to the predisposition in the study of grammars of natural and artificial languages and in the study of formal systems to consider linear sequences of symbols. However, virtually nothing in what follows is actually dependent on this linear method of presentation. Thus, pattern recognition problems where the sample consists of other structures or rasters of symbols are equally amenable to the methods described herein.
B. THE SEARCH PHASE--FINDING REGULARITIES IN A SAMPLE OF SENTENCES

1. INTRODUCTION TO THE SEARCH PHASE

The first main phase of the Grammar Discovery Algorithm is the Search Phase.

In the Search Phase, we search for local regularities in the given sample of sentences. Informally, a "regularity" exists whenever the occurrence of certain symbols somewhere in a sentence in the sample implies that certain other symbols occur elsewhere in the sentence with a conditional probability greater than mere chance.

In this section, we proceed to define formally the notion of a cylinder set in a sample; to generalize this concept using the idea of a mask; to categorize these masks by grammatical type; to define formally the notion of context-sub-sequence, the predicted-sub-sequence, and the domain sequence; and to formally define and illustrate the idea of a regularity in a sample. We then argue that if a grammar discovery algorithm is to begin with a search for regularities, the length of those regularities must be small—that is, we should only search for local regularities. We present the algorithm of the Search Phase.

The Search Phase is an exhaustive, combinatorial process--within the limited range established by the principle of searching only for local regularities.
2. CYLINDER SETS IN A SAMPLE

Let V be a set of C symbols, \( a_1, a_2, \ldots, a_C \). Call V the alphabet. A sentence over V is a sequence

\[ s = y_1 y_2 \ldots y_n \]

where each \( y_i, 1 \leq i \leq n \), is in V. Our interest here is in infinite sentences, although most of the statements we make would apply equally to infinite sentences.

Let Y be a given sample of sentences. In the terminology of automata theory and information theory (Khinchin, 1956), each sentence in the set of all possible sentences over \( V_T \) is an elementary event of some space of events. Each subset of Y (and Y itself) then is an event. We are interested in a particular kind of event—the cylinder set.

An index sequence

\[ Z = (t_1, t_2, \ldots, t_h) \]

is a sequence of different non-zero positive integers. An allowable index sequence for a given set Y of sentences is an index sequence in which no integer is greater than the length of the shortest sentence in Y. Two index sequences are disjoint if they contain no integers in common. The cardinality of an index sequence X, denoted \( |Z| \), is the number of integers in Z.

A cylinder set \( \xi \), or a cylinder, for a given sample Y of sentences is the subset of all sentences in Y such that

\[ y_{t_i} = a_i \]

for all \( i = 1, \ldots, h \), and where each \( a_i \in V \), and where the sequence \( (t_i)_{t=1}^h \) is an allowable index sequence for the set Y. \( \xi \) may, of course, turn out to be empty. Note that the cylinder set \( \xi \) is defined in terms of 3 parameters: (1) the sample Y of sentences, (2) the allow-
able index sequence $Z$, and (3) a sequence of $a_i$'s from the alphabet $V$.

Consider the cylinder set $\xi$ defined for a sample $Y$ of sentences, and index sequence $Z = (t_i)_{i=1}^{h}$, and $a$'s from the alphabet $V$. Let $W = (w_j)_{j=1}^{g}$ be an index sequence disjoint from $Z$. Let $\sigma$ be the cylinder set defined over the cylinder set $\xi$ (which is a subset of the sample $Y$ of sentences), an index sequence $W$, and $g$ $a$'s from $V$. A positive regularity exists in a sample $Y$ of sentences if, for some non-empty cylinder set $\xi$ defined as above, and for some cylinder set $\sigma$ defined as above, then the conditional probability

$$\frac{|\sigma|}{|\xi|} > \frac{1}{c^g}$$

We denote the quantity $1/c^g$ by $\varepsilon$. Note that this quantity is the probability associated with a "pure chance" event—that is, it is the probability of occurrence of particular string over $C$ symbols of length $g$ if each of the strings occurs equiprobably.

It will be customary herein to number the cylinder sets, such as $\xi$ and $\sigma$, so that we will then be able to index the conditional probabilities (and thereby the positive regularities) with two numerical indices and refer to them as $P_{i,j}$, where $i$ is the index number assigned to cylinder set $\xi$, and $j$ is the index number assigned to $\sigma$.

The $h$ positions of the index sequence $Z$ constitute the context part of the positive regularity; and the $g$ positions of the index sequence $W$ constitute the predicted part of the positive regularity.

Note that a positive regularity in a given sample is specified by 4 parameters: (1) the sample, (2) a context part, (3) a predicted part, and (4) the conditional probability $|\sigma|/|\xi|$.

In what follows, we will assume that the union of the $h+g$ integers in the disjoint index sequences $Z$ and $W$ together constitute $h+g$ consecutive integers—or, equivalently, that the context part
and the predicted part cover \( h+g \) contiguous positions of the sentences of the sample. We will discuss the occasions when this is not the case. separately later in the section of TERNARY MASKS.
3. MASKS

It will prove advantageous to generalize the above approach to cylinder sets and to detach the cylinders from specific positions $t_1, \ldots, t_h$. That is, if a "1" in position 1 and a "1" in position 2 reliably predict a "0" in position 3, and a "1" in position 4 and a "1" in position 5 reliably predict a "0" in position 6, we should be able to generalize these two observations about the sample and identify them as being a regularity applying to sequences of length 3. Indeed, often the same phrase will appear many times in a sample—but it will usually appear in many different relative positions within the sentences. We should therefore be able to generalize these regularities and identify them as one regularity.

Khinchin accomplishes this generalization by introducing a time shift operation. This approach is equivalent to the various "window" schemes in pattern recognition approaches (Uhr, 1963).

However, even with the time shift, all the observations are specific to the symbols that occur in the various positions. That is, if an "a" in the first position (note: not position 1) and a "c" in a third position reliably predict a "b" in the second position, and a "d" in a first position and an "f" in a third position reliably predict an "e" in the fifth position, then we should also be able to generalize these two observed regularities and identify them as one regularity applying to certain sequences of length 3.

Thus, it is desirable to make a further generalization and detach the specific context symbols and the specific predicted context symbols from the regularity, and express the fact that certain positions may constitute a context that reliably predicts certain predicted positions. This leads to the following definition:
A binary mask, $M$, or a mask, is a binary sequence over the symbols "_" and "%"—provided the sequence contains at least one "_" and at least one "%". The symbol "_" denotes "the context." The symbol "%" denotes "the predicted part." An $M$-mask is a mask of length $M$. For example, "__%__" is a 3-mask in which the context is the first and third position, while the predicted part is the second position.

The universe of binary masks is all $2^M - 2$ binary sequences over "_" and "%" provided the sequence contains at least one "_" and at least one "%". By implication, the shortest masks are of length 2.

For example, the universe of binary masks of length 3 consists of the 6 masks "__%", "__%%", "%%_", "%__", "___%", and "%__". Note that "____" (all context) is not considered a mask and has no meaning. Also "% % %" (all predicted part) is not considered a mask at this point in this paper. It can, however, be given a reasonable meaning by interpreting it as the mask which catalogs unconditional probability of occurrence of strings of length 3 in the sample.*

*The control parameter LAWL controls whether this extra mask is used in the computer program implementing the Grammar Discovery Algorithm.
4. TYPES OF MASKS

The universe of M-masks is partitioned by grammatical type as follows:

All masks of the form "\textasciimacron}^M-k\textasciimacron}_{k\neq0}\) are called strictly left-sensitive. All masks of the form "\textasciimacron}^kM-k\textasciimacron}_{h\neq0}\) are called strictly right-sensitive. Both of these two types of masks are considered to be of the same level of grammatical complexity—that is, context-sensitive. "\textasciimacron}'' is an example of a strictly left-sensitive 3-mask, and "\textasciimacron}'' is an example of a strictly right-sensitive 2-mask.

All masks of the form "\textasciimacron}^k_1 \textasciimacron}^k_2M-k_1-k_2\) , where \(k_1\neq0\) and \(k_2\neq0\) and \(k_1+k_2\neq M\), are called strictly context-sensitive. M must be at least 3 to have a strictly context-sensitive mask. "\textasciimacron}'' is an example of a strictly context-sensitive mask.

All other masks are called strictly unrestricted rewrite. These masks are characterized by more than two switches from "\textasciimacron}'' to "\textasciimacron}'' or from "\textasciimacron}'' to "\textasciimacron}'' (if they start with a "\textasciimacron}''), or by two or more such switches (if they start with a "\textasciimacron}''). "\textasciimacron}'' is the shortest strictly unrestricted rewrite mask.

The justification for the choice of these names for these subsets of masks will be seen later, in the section wherein we use the regularities derived from the masks to develop the rules of production of the induced grammar.

Masks can have a radix of either 2 or 3. If the radix of a mask is 3, the mask is called a ternary mask. A ternary mask is a sequence over the symbols "\textasciimacron}'', "\textasciimacron}''", and "\textasciimacron}''". The symbols "\textasciimacron}''" and "\textasciimacron}''" are defined as for binary masks. The symbol "\textasciimacron}''" denotes "don't care."

There are several important differences and effects when one uses ternary masks, instead of binary masks. Ternary masks (1) allow
a separation in positions between the symbols of a regularity; (2) allow a kind of generalization operation (opposite in effect to the substitution rule of inference in logic) over whatever may be in a certain position; and (3) may allow certain cryptographic regularities to be more readily discovered. Ternary masks will be discussed in full later.

For example, of the 6 masks of length 3, "_ % %" and "_ _ %" are strictly left-sensitive; "% _ " and "% _ " are strictly right-sensitive; "_ % _" is strictly context sensitive; and "% _ %" is strictly unrestricted rewrite.
5. CONTEXT SUB-SEQUENCES, PREDICTED SUB-SEQUENCES, AND THE DOMAIN SEQUENCE

The context-sub-sequence $\gamma$ is a sequence of length $M$ over the $C$ symbols of the current alphabet $V_c$ and the predicted symbol "§" defined, for $I = 1, 2, \ldots, M$ as

$$\gamma(I) = \begin{cases} 
Y(T-M+1) & \text{if } M(I) = "\_" \\
"§" & \text{if } M(I) = "§"
\end{cases}$$

The symbol "§" is merely a filler symbol in $\gamma$, and its appearance here does not carry any of the meaning that the symbol otherwise has. Because of this convention, note that the context-sub-sequence $\gamma$ contains in itself all the information of the binary mask $M$ from which it was produced.

The predicted-sub-sequence $\phi$ is a sequence of length $M$ over the $C$ symbols of the current alphabet $V_c$ and the context symbol "\_" defined, for $I = 1, 2, \ldots, M$ as

$$\phi(I) = \begin{cases} 
Y(T-M+1) & \text{if } M(I) = "§" \\
"\_" & \text{if } M(I) = "\_"
\end{cases}$$

Here again "\_" is a filler symbol in $\phi$. Note that the predicted-sub-sequence contains in itself all the information of the binary mask $M$ from which it was produced.

If the mask is a ternary mask, then $\gamma(I)$ and $\phi(I)$ are "#" whenever $M(I)$ is "#".

Finally, we define the domain sequence as the sequence of length $M$ over the current alphabet $V_c$ obtained as follows:

$$\Delta(I) = \begin{cases} 
\gamma(I) & \text{if } \phi(I) = "§" \\
\phi(I) & \text{if } \gamma(I) = "\_"
\end{cases}$$

for $I = 1, \ldots, M$. Note that $\Delta$ consists of the symbols from the current alphabet $V_c$ (and not "§" or "\_") occurring position-by-position in either $\gamma$ or $\phi$, and represents the symbols actually appearing in $Y$. 
6. DEFINITION OF A REGULARITY

A regularity \( \mathcal{R} = \langle \gamma, \phi, p, Y \rangle \) is defined as a quadruple consisting of a context-sub-sequence \( \gamma \) (a sequence of length \( M \) over the symbols of the current alphabet \( V_c \) and the symbol "%"), a predicted-sub-sequence \( \phi \) (a sequence of length \( M \) over the symbols of the current alphabet \( V_c \) and the symbol "_"), a real number \( p \) representing the conditional probability that the occurrence of \( \gamma \) in the sample \( Y \) predicts the occurrence of the predicted-sub-sequence \( \gamma \) in \( Y \). When the sample \( Y \) intended is clear, we may write \( \mathcal{R} \) as \( \langle \gamma, \phi, p \rangle \). The length of a regularity is \( M \), which is the common length of \( \gamma \) and \( \phi \) and \( \Delta \).

In English text, for example, one regularity is \( \mathcal{R} = \langle "Q\%", '_U", 1.0 \rangle \). This regularity is of length 2. This regularity expresses the fact that the letter "Q" is always followed by the letter "U". The binary mask \( \mathcal{M} \) from which this regularity is derived is "_%", and is of length 2. This mask is a strictly left-sensitive mask. The context-sub-sequence \( \gamma \) of this regularity is "Q%", and the predicted-sub-sequence \( \phi \) is "_U". Both, of course, are also of length 2. The conditional probability \( p \) that the context-sub-sequence predicts the occurrence of the predicted-sub-sequence in the sample is 1.0. The domain sequence \( \Delta \) for any occurrence of this regularity is "QU"—that is, "QU" are the symbols that actually occur in the sample. \( \Delta \) is also of length 2.

Note that the frequency of occurrence of a given context-sub-sequence (the "Q" in the above case) does not enter into the definition of the regularity since conditional probabilities are used. It should be noted, however, that the notion of a mask can be extended in order to accomodate the measurement of probabilities of occurrence of single symbols, 2-grams, 3-grams, etc. in the sample. This extention is
accomplished by allowing a mask to consist entirely of the predicted symbol ( %). With a vacuous context, the conditional probability $P_{ij}$ now becomes the probability of occurrence of the predicted-sub-sequence $. The probability of occurrence is not, however, used anywhere in the Grammar Discovery Algorithm.
7. TYPES OF REGULARITIES

We classify regularities and $P_{ij}$'s by the magnitude of the conditional probability $P_{ij}$. If the $P_{ij}$ is exactly 1.0, we say that the regularity (or, the $P_{ij}$ itself) is of type 1, and we call it structural. The regularity that "U" follows "Q" in English is a structural regularity. If $P_{ij}$ is near 1.0; that is, if $P_{ij}$ is less than 1.0 but greater than or equal to $1.0 - 1/c^g$, then we say that the regularity (and the $P_{ij}$) is of type 2. If the $P_{ij}$ is less than $1.0 - 1/c^g$ but greater than or equal to 1/2, then we say that the regularity (and the $P_{ij}$) is of type 3. If the $P_{ij}$ is less than 1/2 but greater than or equal to $\epsilon = 1/c^g$ then we say that the regularity (and the $P_{ij}$) is of type 4. The regularities of these 4 types are the positive regularities. If the $P_{ij}$ is less than $\epsilon$, but greater than 0.0, we say that the $P_{ij}$ is of type 5, and we call it noise. If the $P_{ij}$ is exactly 0.0, we say that the $P_{ij}$ is of type 6. We are interested, of course, in the $P_{ij}$ that are 1.0 or near 1.0.
8. RATIONALE FOR SEARCHING FOR LOCAL REGULARITIES

It is not obvious that searching for local regularities in a sample of sentences is the way to begin a grammar discovery algorithm. Indeed, both practical and theoretical considerations seem to advise against this kind of approach.

First of all, it would appear that, in searching for regularities in a sample of sentences produced by an unknown grammar, one would be obliged to consider dependencies between symbols widely separated from one another in the sentence. This point is made in Chomsky (1956, p. 286) in discussing the distance in the "relations of dependencies" that may occur as a result, for example, of self-embedding in natural languages such as English. In this regard, consider the sentence

$$++p+pp$$

which is a well-formed sentence of the propositional calculus (in Polish Notation), and which is generated from the single recursive rule of production "p \rightarrow +pp". When one parses this sentence, the first symbol (i.e. the "*") is in fact generated with the same application of the rule of production as the last symbol (i.e. the final "p").

Now the number of possible regularities observable in M positions ($M \geq 2$) of sentences over an alphabet of C symbols is

$$\sum_{h=1}^{m-h} c^{h \cdot \binom{m}{h}} \cdot c^{m-h}$$

(where h is, as before the number of positions in the context part of the regularity). This is equal to

$$(2^m - 2) c^m \leq f_m$$

which is very large for anything but small m. Indeed, in the simplest case, that of the binary alphabet (C=2), we have
Moreover, we are naturally interested in the possibility of sentences of indefinite length; that is, in sentences that are produced by automata with cyclic state transition diagrams, and in sentences produced by grammars with at least one recursive rule of production. A finite sample of such sentences—which is, of course, the only situation we would encounter—contains a longest sentence; but, still, that sentence could be very long. Thus, there would seem to be practical combinatorial obstacles to any algorithm which is based on searching for regularities over \( m \) symbols, for even modestly large \( m \), much less the \( m \) one would encounter in practice.

Even without the practical combinatorial obstacles that seem to surround analysis of sentences of non-small length \( m \), there is a theoretical problem in searching for regularities among distant symbols. Consider a finite sample of sentences over an alphabet \( V \) with \( C \) symbols, wherein the longest sentence is of length \( m \). Unless the sample contains at least one instance of each of the \( C^m \) possible sequences of length \( m \), (which would make the grammar very uninteresting, indeed) there will be at least one, and in general very many, conditional probabilities that are greater than \( \frac{1}{C^g} \), for each \( g = 1, 2, 3, \ldots, m-1 \). Thus, there will be at least one positive regularity in any such sample. The fact that positive regularities do appear will be used later. But for the moment, it will be used to make another argument.

Consider again all the well-formed sentences of the propositional calculus generated by the single recursive rule of production "\( P \rightarrow PP \)". The sentences produced by this rule are all of length \( m = 3, 5, 7, 9, \ldots \). There are \( 2^{\frac{m-3}{2}} \) sentences of length \( m \). And, in particular, there are

<table>
<thead>
<tr>
<th>( m )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 4 )</th>
<th>( 5 )</th>
<th>( 6 )</th>
<th>( 7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_m )</td>
<td>8</td>
<td>48</td>
<td>224</td>
<td>960</td>
<td>3968</td>
<td>16,128</td>
</tr>
</tbody>
</table>
8 sentences of length 9; namely,

+P+P++PPP
++P++PPP
+P+++PPP
++++PPP
+P+P+P+P
++P+P+PPP
+P+++P+PP
++++P+PPP

Suppose further, we are given a sample of sentences in which no sentence is of length greater than 9, and that only 7 of the 8 sentences of length 9 are included in the sample. Suppose it is the last sentence which is not present. Then the conditional probability that there is a "P" in position 6, given that there is a "++++P" in positions 1 through 5, respectively, and given that there is a "PPP" in positions 7 through 9, respectively is 1.0. Since c=2 and g=1 here, this is greater than 1/c^g = 1/2, and indeed therefore indicates a "positive" regularity. This discovered structural regularity is, of course, an artifact of the sample. If the sample were understood to be complete, in the sense of containing all of the sentences of the grammar, then this regularity would be faithful to the grammar. However, this sample of sentences is typical in that it is admittedly an incomplete sample arising from a grammar which is capable of producing other sentences. In a larger sample, it is possible that the missing sentence would occur as a sentence. Or, it may very well appear as a sub-sentence (which we can call phrase or clause) of a longer sentence. Thus, when we search for regularities of length equal to the length of the longest sentences of the sample, the regularities we find are likely to be artifacts of the sample size, and therefore highly dependent and variable on the sample size. Any grammar discovery algorithm based on finding regularities of this quality would be unstable. Our goal, of course, is that the grammar
produced not depend on sample size, after a certain point. A grammar
discovery algorithm should converge, and thereafter be stable.

A final objection to searching for regularities of long length is
that their discovery may indeed tell something about each itemized
long sentence present in the sample, but tell us little about the
sub-structure of the sentence. Thus, even in the absence of other
objections, the regularities produced would tend to lead to trivial
information (i.e. cataloging the long sentences)--rather than informa-
tion about the sub-structure of the sentences.

Thus, if a grammar discovery algorithm is to begin with a search
for regularities, the length of regularities considered must be
small--both absolutely, to avoid practical combinatorial problems--
and relatively, to avoid instability and triviality problems. For
these, and other reasons that will be discussed later, it is necessary
(and also sufficient) to consider only small $M$ in searching for regu-
larities in a grammar discovery algorithm. The way to find global
properties is to look for local properties.
9. ALGORITHM OF THE SEARCH PHASE

In the grammar discovery algorithm, we consider masks of length
M = 2, 3, 4, ..., up to some pre-determined, small upper limit on
M.* The selection of this upper limit M2 will be discussed later.

For each such M, we consider all possible masks $\mathcal{M}$ of length
M which are of grammatical type up to some pre-determined limit**—
perhaps, for example—all masks no more complex than strictly context
sensitive. The masks are considered in order of increasing grammatical
complexity—up to the pre-determined limit. That is, context sensi-
tive masks are considered, for example, before unrestricted rewrite
masks. This pre-determined limit is selected on the basis of what
type of grammatical rules one is seeking. It does not matter whether
one is too generous in admitting rules that are more complex, because
the rules are not only generated in order of increasing grammatical
complexity, but also are adopted into the grammar in that order. Thus,
all the M-1 strictly right sensitive masks of length M, and all M-1
strictly left sensitive masks of length M are generated and considered
first. They, in turn, are followed by the $M(M+2)/2$ strictly context
sensitive masks. These masks are then followed by the strictly unre-
stricted rewrite masks of length M, of which there are

$$2^M - 2 - 2M - 2 - M(M+2)/2$$

Suppose we denote, for the moment, the symbols of a sentence
in the sample as

$$Y(1), Y(2), ..., Y(d),$$

where d is the length of the sentence. Then, given the current M we
are considering, and the current mask $\mathcal{M}$, we examine the sentence,

*The variable M2 is this upper limit in the computer program.

**The variable GRAM controls this in the computer program.
symbol by symbol. We consider the sub-strings of length $M$ terminating in position $T=M$, $M+1$, ..., $d$ ($d \geq M$), and match each with the current mask—that is, we match the symbols $Y(T-M+1)$, ..., $Y(T-M)$ with mask positions 1, 2, ..., $M$. A context-sub-sequence $\gamma$ and a predicted-sub-sequence $\phi$ are then developed from each such matching, as indicated by the definition of $\gamma$ and $\phi$.

Each distinct context-sub-sequence $\gamma$ that is produced is given a distinct index number. Each distinct predicted-sub-sequence $\phi$ that is produced is also given a distinct index number. This allows us to refer to the conditional probability $P_{ij}$ that predicted-sub-sequence $j$ occurs, given that context-sub-sequence $i$ occurred, by using the two indices $i$ and $j$. In the computer program implementing the Grammar Discovery Algorithm, the index number assigned is the natural number equivalent to the $M$ symbols of $\gamma$ and $\phi$ modulo $(C+2)+2$.** These two natural numbers together thus abbreviate all the information needed to identify the sub-sequence as either a predicted sub-sequence or a context-sub-sequence—because a context-sub-sequence contains symbols from the alphabet and the "§", but never the "_", while a predicted-sub-sequence contains symbols of the alphabet and the "_", but never the "§".

**It should be noted that many of the operations of the Search Phase, as well as later phases such as the Recoding Phase, are amenable to being performed by parallel computers—either by general purpose parallel computers which will represent the next generation of computers, or by a special purpose parallel computer. The combinatorial obstacles to performing grammar induction on large samples of unknown text will thus be greatly reduced in practice.

**The natural number corresponding to the context-sub-sequence is called SQCTXT in the computer program. The natural number corresponding to the predicted-sub-sequence is called SQPRED. The reason for the first "+2" is the inclusion of the symbols "_" and "§". The reason for the second "+2" is the inclusion of the symbols "#" and the initial punctuation mask.
The pair of 2 natural numbers together are sufficient to recreate the context-sub-sequence $\gamma$, the predicted-sub-sequence $\phi$, and the mask $M$.

The procedure for doing the above depends somewhat on the mode in which the Grammar Discovery Algorithm is operating. If initial punctuation is present in the sample (the second mode), the examination above proceeds through each separate sentence. Naturally, if $T+M-1>d$, then masks of that length $M$ are not considered for that sentence. (We can, if we like, allow the initial punctuation mask appearing at both ends of a given sentence to be considered as part of that sentence; however, we never allow inter-sentence examinations). If there is no initial punctuation in the sample, the only restriction is that $T+M-1>d$.

In either mode, the entire sample $Y$ is examined under the current mask $M$.

Upon completion of this examination, it is now possible to compute the conditional probability $P$ that a given predicted-sub-sequence $\phi$ arising from a particular mask $M$ of length $M$ occurred, given the occurrence of a particular context-sub-sequence $\gamma$ arising from that same mask.

This process is then repeated for each mask being considered.

It is convenient to sort these conditional probabilities into descending numerical order, from 1.0 on down. The $P_{ij}$'s are then classified into types (I, II, III, IV, and V) as previously described.

Note that for a given context-sub-sequence with $h$ context positions, one would expect that most of the $(M-h)^C = g^C$ possible predicted-sub-sequences will not actually occur in the sample. In a non-pathological sample, there are relatively few context-sub-sequence
and predicted-sub-sequence pairs that will actually occur.*

The result of the Search Phase is the production of tables of regularities that occur in the sample.

*In the computer program implementing the Grammar Discovery Algorithm, the possibility of an excessive number of small, insignificant $P_{ij}$'s is dealt with by providing for the early culling of the tables of regularities. This early elimination of $P_{ij}$'s is controlled by the parameter EARLY.
10. EXAMPLES

Example A: To illustrate the ideas developed above, consider the unpunctuated (first mode) symbol string Y as follows: THE-BIG-CAT

The terminal alphabet for this sample Y of length N = 64 over C1 = 12 symbols is A = \{A,B,C,D,E,G,H,I,N,R,T,\cdot\}, where "\cdot" is an undistinguished, but visible symbol to denote the blank.

The table below illustrates the kind of masks, context sequences, predicted sequences, and conditional probabilities that one obtains in analyzing the sample.

SOME REGULARITIES FOR LEVEL 1

<table>
<thead>
<tr>
<th>LENGTH M</th>
<th>CONDITIONAL PROBABILITY Pij</th>
<th>TYPE</th>
<th>MASK</th>
<th>CONTEXT Sub-Sequence M</th>
<th>PREDICTED Sub-Sequence \delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.000000</td>
<td>1</td>
<td>_%</td>
<td>C_%</td>
<td>_AT</td>
</tr>
<tr>
<td>3</td>
<td>1.000000</td>
<td>1</td>
<td>_%</td>
<td>TH%</td>
<td>_E</td>
</tr>
<tr>
<td>3</td>
<td>1.000000</td>
<td>1</td>
<td>_%</td>
<td>%G%</td>
<td>_B_I</td>
</tr>
<tr>
<td>3</td>
<td>1.000000</td>
<td>1</td>
<td>_%</td>
<td>%AN%</td>
<td>_R_</td>
</tr>
<tr>
<td>3</td>
<td>1.000000</td>
<td>1</td>
<td>_%</td>
<td>R_T%</td>
<td>_A_</td>
</tr>
<tr>
<td>3</td>
<td>0.500000</td>
<td>3</td>
<td>_%</td>
<td>B_%%</td>
<td>_I_G</td>
</tr>
<tr>
<td>3</td>
<td>0.500000</td>
<td>3</td>
<td>_%</td>
<td>B_%%</td>
<td>_A_D</td>
</tr>
</tbody>
</table>

The first line of this table, for example, indicates that the letter C is always followed by the letters AT in the sample. This forms the "word" CAT. The second line of this table indicates that the letter pair TH is sufficient to determine the following letter which is E. These 2 lines illustrate left-sensitive rules.

Line 3 of this table indicates that the letter G is sufficient context on the right to predict, with 100% reliability, the preceding two letters, namely B and I. Line 4 indicates that the context of R on the left and the context of T on the right is sufficient to determine
the one intervening letter A, making the word RAT. This is an example of a strictly context sensitive rule.

The only mask not illustrated for length 3 above is the unrestricted rewrite mask %-%. This is the only unrestricted rewrite mask of length 3.

The last two lines of this table indicate that the letter B is followed by IG and AD equally often (i.e. with probability .50).

Using the terminology we have developed, the information on line 1 represents the type I regularity $R = \langle"C\%", \_AT", 1.0,Y\rangle$ of length $M = 3$.

Example B: Now let us consider an example based on the following sequence of 272 bits found in the memory of an IBM 360 computer:

(1) "1110 0110 1100 1000 1100 0101 1101 0101 0100 0000 1100 1001 1101 0101 0100 0000 1110 0011 1100 1000. 1100 0101 0100 0000 1100 0011 1101 0110 1100 0100 1100 0010 1100 0101 0100 0000 1101 0110 1100 0110. 0100 0000 1100 1000 1110 0100 1101 0100 1100 0001 1101 0101 0100 0000 1100 0010 1110 0101 1110 0101 1100 0101. 1101 0101 1110 0011 1110 0010 0100 0000"

Here the alphabet $A = \{0,1\}$ and $N = 272$ and $C_1 = 2$.

When the memory of the computer is dumped, the sequence of 272 bits would conventionally be presented as 68 half-bytes (4 bit sequences). The name conventionally attached to each 4 bit sequence is the hexadecimal equivalent of the 4 bits in binary. Using these names, the dump might read:

(2) "E6C8C5D5 40 C9D5 40 E3C8C5 40 C3D6E4E2C5 40 D6C6 40 C8E4D4C1D5 40 C5 E5 C5D5E3 E2 40"
When the 68 half bytes are then combined into 34 full bytes and presented as EBCDIC characters (as is the case in the EBCDIC dump), we see

(3) "WHEN IN THE COURSE OF HUMAN EVENTS"

If we now search the sample (1) for regularities, we find for a regularity length M of 2 the following regularities:

LEVEL = 1  M = 2

| NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) | 0 |
| NUMBER OF TYPE 2 SEQUENCES (MESSAGE)    | 0 |
| NUMBER OF TYPE 3 SEQUENCES (MESSAGE)    | 4 |
| NUMBER OF TYPE 4 SEQUENCES (MESSAGE)    | 4 |
| NUMBER OF TYPE 5 SEQUENCES (NOISE)      | 4 |
| NUMBER OF P(I)                          | 12 |
| NUMBER OF POSSIBLE P(I)                 | 8 |
| SEPARATION VALUE BETWEEN TYPES 3 AND 4  | 0.50000 |
| EPSILON FOR DEFINING TYPES 2 AND 3      | 0.34000 |

<table>
<thead>
<tr>
<th>#</th>
<th>LEVEL</th>
<th>LENGTH</th>
<th>PROB</th>
<th>TYPE</th>
<th>MASK</th>
<th>CONTEXT</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>_%</td>
<td>1%</td>
<td>_0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>0%</td>
<td>_0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>0.55</td>
<td>3</td>
<td>_%</td>
<td>0%</td>
<td>_0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0.55</td>
<td>3</td>
<td>%</td>
<td>0%</td>
<td>_0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0.45</td>
<td>4</td>
<td>%</td>
<td>0%</td>
<td>_1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>0.45</td>
<td>4</td>
<td>%</td>
<td>0%</td>
<td>_1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.43</td>
<td>4</td>
<td>%</td>
<td>1%</td>
<td>_1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0.42</td>
<td>4</td>
<td>%</td>
<td>1%</td>
<td>_1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>0.31</td>
<td>5</td>
<td>%</td>
<td>2%</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>0.25</td>
<td>5</td>
<td>%</td>
<td>2%</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>0.25</td>
<td>5</td>
<td>%</td>
<td>2%</td>
<td>01</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>0.19</td>
<td>5</td>
<td>%</td>
<td>2%</td>
<td>11</td>
</tr>
</tbody>
</table>

Note that the mask "%%" is included in the above table and that it catalogs the probabilities of occurrence of the various sequences of symbols appearing in the sample.
Similarly, if we search the sample for regularities of length no greater than 3, we obtain the following table of regularities:

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>M</th>
<th>3</th>
</tr>
</thead>
</table>

| NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) | 0 |
| NUMBER OF TYPE 2 SEQUENCES (MESSAGE) | 0 |
| NUMBER OF TYPE 3 SEQUENCES (MESSAGE) | 8 |
| NUMBER OF TYPE 4 SEQUENCES (MESSAGE) | 8 |
| NUMBER OF TYPE 5 SEQUENCES (NOISE) | 24 |
| NUMBER OF P(1) | 40 |
| NUMBER OF POSSIBLE P(1) | 32 |
| SEPARATION VALUE BETWEEN TYPES 3 AND 4 | 0.5000 |
| EPSILON FOR DEFINING TYPES 2 AND 3 | 0.3333 |

<table>
<thead>
<tr>
<th>#</th>
<th>LEVEL</th>
<th>LENGTH</th>
<th>PROB</th>
<th>TYPE</th>
<th>MASK</th>
<th>CONTEXT</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>1</td>
<td>3</td>
<td>0.61</td>
<td>3</td>
<td>%</td>
<td>11%</td>
<td>___0</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>3</td>
<td>0.52</td>
<td>3</td>
<td>%</td>
<td>10%</td>
<td>___0</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>3</td>
<td>0.49</td>
<td>4</td>
<td>%</td>
<td>10%</td>
<td>___1</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>3</td>
<td>0.43</td>
<td>4</td>
<td>%</td>
<td>10%</td>
<td>___1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___0</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>2</td>
<td>0.57</td>
<td>3</td>
<td>%</td>
<td>1%</td>
<td>___1</td>
</tr>
</tbody>
</table>
Some of the regularities we obtain with an $M$ of 4 are

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>M</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>4</td>
<td>0.70</td>
</tr>
<tr>
<td>87</td>
<td>4</td>
<td>0.68</td>
</tr>
<tr>
<td>96</td>
<td>4</td>
<td>0.67</td>
</tr>
<tr>
<td>85</td>
<td>4</td>
<td>0.65</td>
</tr>
<tr>
<td>139</td>
<td>4</td>
<td>0.63</td>
</tr>
<tr>
<td>147</td>
<td>4</td>
<td>0.63</td>
</tr>
<tr>
<td>90</td>
<td>4</td>
<td>0.61</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>0.61</td>
</tr>
<tr>
<td>98</td>
<td>4</td>
<td>0.61</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>0.60</td>
</tr>
<tr>
<td>93</td>
<td>4</td>
<td>0.60</td>
</tr>
<tr>
<td>138</td>
<td>4</td>
<td>0.60</td>
</tr>
<tr>
<td>135</td>
<td>4</td>
<td>0.58</td>
</tr>
<tr>
<td>134</td>
<td>4</td>
<td>0.58</td>
</tr>
<tr>
<td>144</td>
<td>4</td>
<td>0.58</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.57</td>
</tr>
<tr>
<td>26</td>
<td>3</td>
<td>0.57</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.57</td>
</tr>
<tr>
<td>42</td>
<td>1</td>
<td>0.57</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>0.56</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>0.55</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.55</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.55</td>
</tr>
</tbody>
</table>

**Number of Type 1 Sequences (Structural):** 0

**Number of Type 2 Sequences (Message):** 0

**Number of Type 3 Sequences (Message):** 16

**Number of Type 4 Sequences (Message):** 32

**Number of Type 5 Sequences (Noise):** 64

**Number of Possible $P(I)$:** 112

**Separation Value Between Types 3 and 4:** 0.50000

**Epsilon for Defining Types 2 and 3:** 0.25000
And, some of the regularities we obtain with an M of 5 are shown below:

\[ \text{LEVEL} = 1 \quad \text{M} = 5 \]

| NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) | C |
| NUMBER OF TYPE 2 SEQUENCES (MESSAGE) | 2 |
| NUMBER OF TYPE 3 SEQUENCES (MESSAGE) | 33 |
| NUMBER OF TYPE 4 SEQUENCES (MESSAGE) | 79 |
| NUMBER OF TYPE 5 SEQUENCES (NOISE) | 174 |
| NUMBER OF P(I) | 288 |
| NUMBER OF POSSIBLE P(I) | 256 |
| SEPARATION VALUE BETWEEN TYPES 3 AND 4 | 0.50000 |
| EPSILON FOR DEFINING TYPES 2 AND 3 | 0.20000 |

<table>
<thead>
<tr>
<th>#</th>
<th>LEVEL</th>
<th>LENGTH</th>
<th>PROB</th>
<th>TYPE</th>
<th>MASK</th>
<th>CONTEXT</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>1</td>
<td>5</td>
<td>0.86</td>
<td>2</td>
<td>_____</td>
<td>1111%</td>
<td>0</td>
</tr>
<tr>
<td>419</td>
<td>1</td>
<td>5</td>
<td>0.86</td>
<td>2</td>
<td>%____</td>
<td>%1111</td>
<td>0</td>
</tr>
<tr>
<td>394</td>
<td>1</td>
<td>5</td>
<td>0.74</td>
<td>3</td>
<td>%____</td>
<td>%0011</td>
<td>0</td>
</tr>
<tr>
<td>287</td>
<td>1</td>
<td>5</td>
<td>0.73</td>
<td>3</td>
<td>_____</td>
<td>1010%</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
<td>5</td>
<td>0.73</td>
<td>3</td>
<td>%____</td>
<td>%1011</td>
<td>0</td>
</tr>
<tr>
<td>270</td>
<td>1</td>
<td>5</td>
<td>0.72</td>
<td>3</td>
<td>_____</td>
<td>0110%</td>
<td>0</td>
</tr>
<tr>
<td>272</td>
<td>1</td>
<td>5</td>
<td>0.70</td>
<td>3</td>
<td>_____</td>
<td>1101%</td>
<td>0</td>
</tr>
<tr>
<td>142</td>
<td>1</td>
<td>4</td>
<td>0.70</td>
<td>3</td>
<td>%____</td>
<td>%101</td>
<td>0</td>
</tr>
<tr>
<td>410</td>
<td>1</td>
<td>5</td>
<td>0.70</td>
<td>3</td>
<td>%____</td>
<td>%0101</td>
<td>1</td>
</tr>
</tbody>
</table>

In later sections, we will refer to these tables as part of other examples.
C. THE RECODING PHASE---DEVELOPING RULES OF PRODUCTION FROM REGULARITIES

1. INTRODUCTION TO THE RECODING PHASE

The Recoding Phase is the second main phase of the Grammar Discovery Algorithm.

In the Recoding Phase, the local regularities found during the Search Phase are used to develop grammatical rules of production which, in turn, are used to transform the given set of sentences into a new set of sentences. The new set of sentences contains most of the information and structure of the original set of sentences. This transformation (recoding) process works by replacing contiguous predicted symbols of local regularities occurring in the sample with single symbols. Thus, what were originally global regularities in the given sample tend to become local regularities in the new set of sentences. The recoding that is done is defined, in part, in terms of grammatical rules of production; and these rules of production ultimately become the rules of production of the grammar being induced. The resulting new sentences can later be treated anew as a sample and this new sample can then be searched for regularities, reusing the methods of the Search Phase. And, this new sample can then be recoded at this higher level, yielding additional rules of production and a newer set of sentences. The alternate application of first the Search Phase, and then the Recoding Phase is what produces the hierarchical (and non-trivial) quality in the grammar being induced.
2. TRANSFORMATIONS (RECODINGS)

We are concerned here with mappings from a sequence Y of length N over Cl symbols to a new sequence Y' of length N' over Cl' symbols. A transformation (or recoding) T as used here will refer to such a mapping. If Cl' > Cl, we say that the mapping is alphabet-enlarging.

A transformation is presented in terms of a set of (1) grammatical rules of production, and (2) a Recoding Procedure. Both are necessary to define the transformation. Each grammatical rule of production consists of an antecedent (left) side, which is a symbol string over the alphabet of Y'; an arrow; and a consequent (right) side, which is a symbol string over the alphabet of Y. An example of a rule of production is

$$OAO \rightarrow 01110$$

We read this as "replace A by 111 whenever A is found in context with a zero on both sides of it."

Note that the set of rules of production is not sufficient, in general, to uniquely define the mapping from Y to Y'. There will often be many overlapping occurrences of the context-sub-sequences of the local regularities. Hence there are many different overlapping opportunities for applying the rules of production. It is therefore necessary to establish an order of precedence for developing the rules of production. The Recoding Procedure is an algorithm for scanning the sequence Y for occurrences of the context-sub-sequences of the various local regularities and for specifying in what order the rule of production should be developed from these regularities.

The sample Y, as initially presented, is said to be the sample of the first level. Each transformation, when applied to a sample, maps that sample from its current level to a sample of one higher level. It should be noted that each level above the first represents
one sentential form in the derivation of the terminal string (which is the sample at the first level). Thus, each level above the first will contain some non-terminal symbols.

We adopt the convention here that a transformation of a given sample $Y$ will always be over the complete domain $Y$. This will in general necessitate the application of the identity transformation on certain substrings of $Y$. This convention guarantees that the entire domain $Y$ is "covered" by each transformation, and this completeness of covering facilitates later comparisons between transformations.
3. DEVELOPMENT OF RULES OF PRODUCTION FROM LOCAL REGULARITIES

In the Search Phase, certain local regularities were discovered. Recall that a regularity $R = \langle \gamma, \phi, P, Y \rangle$ consists of a context-sub-sequence $\gamma$, a predicted-sub-sequence $\phi$, and an associated conditional probability $P$. The local regularities are now the basis for developing the rules of production of the grammar.

Suppose we are given a context-sub-sequence $\gamma$ from some regularity $R$ of length $M$. Suppose further that for some integral $T > 0$,

$$Y(T-M+1) = \gamma(I)$$

for each $I = 1, 2, \ldots, M$ for which $\gamma(I)$ is not "§" or "#"*—that is, suppose the context specified by the context-sub-sequence $\phi$ is present in the sample $Y$; and, that its terminal position is position $T$ of $Y$. Suppose the Recoding Procedures indicates it is appropriate that a rule of production be developed using this occurrence of the context part of this regularity, and suppose further that we are generating context-dependent rules of production and that we are using only alphabet-enlarging recordings. Then the rule of production would be produced as follows:

1. The consequent (right) side of the rule consists of the domain sequence

   $$\Delta = \Delta(I), \ldots, \Delta(M)$$

   —which is of length $M$.

2. The antecedent (left) side consists of a sequence of symbols constructed as follows: Begin with the context-sub-sequence $\gamma$ and consider each sub-string of contiguous predicted symbols "§" in $\gamma$. Enlarge the current alphabet $V_C$ by a single new symbol, say "N".

* "#" is the "don't care" symbol and is discussed more fully later.
This new symbol will be a non-terminal symbol in the induced grammar. Now, insert this single new symbol \(N\) into the context-sub-sequence \(\gamma\) in place of the entire substring of contiguous predicted symbols. If there is more than one sub-sequence of contiguous predicted symbols in \(\gamma\), introduce another new non-terminal symbol into \(V_C\) for each such sub-sequence. Note that there can be more than one such sub-sequence only when the mask from which the regularity was derived was of strictly unrestricted rewrite type. Note that the antecedent (left) side thus produced is of length \(M\) or less. Note that this decrease in length is the result of replacing the entire contiguous predicted substring by a single new symbol. Note also that the symbols of the current alphabet \(V_C\) occurring in context or don't care positions in the sample are left unchanged. Finally, note that the grammatical type of the rule of production developed in this way is exactly the grammatical type of the mark from which the regularity \(R\) was derived.

Suppose, for example, that \(R = \langle"11\%", ", 00", 1.0, Y\rangle\) is a regularity in a given sample \(Y\)—that is, in \(Y\), the symbols 11 are invariably followed by 00. The context-sub-sequence \(\gamma\) for this structural regularity is "11\%". The predicted-sub-sequence \(\phi\) is "\(-00". The conditional probability \(P\) is 1.0. The mask "\(-\%" gave rise to this regularity, and this mask is a left-sensitive mask. The domain sequence \(\Delta\) is "1100". The sequence \(Y\) begins 0100110001..., so that the context part of the context-sub-sequence "11\%" occurs in \(Y\) so that its final symbol (i.e. the second "\%") is at position \(T = 8\) of \(Y\). The rule of production developed from this regularity of length 4 in \(Y\) will be the left-sensitive rule

\[11N \rightarrow 1100,\]

where \(N\) is understood to be a non-terminal. In this example, "N"
replaces the two contiguous predicted symbols, which in this case, are the two 0's.

The alternative to generating context-dependent rules of production is to generate context-free rules. If context-free rules are to be generated, the antecedent (left) side will consist solely of a single new non-terminal symbol. In this event, the antecedent (left) side will, of course, be of length 1. If we are considering only M ≥ 2, then it will be shorter than the consequent side. The subject of generating context-free rules will be discussed in detail later.

An alternative to an alphabet-enlarging code is a recoding using a Huffman code, or a similar scheme for optimal recoding. In this approach, the alphabet is not enlarged. Instead, a distinctive symbol string in the original alphabet is inserted in place of the symbols in Y. The details of possible non-alphabet-enlarging recordings and the advantages and disadvantages of this kind of recoding will be discussed later.

Thus, as we have seen, the length of the regularities searched for in the Search Phase limits the length of the consequent (right) side of the rules of production developed for the induced grammar. Thus, if M is the length of the longest regularity searched for in the Search Phase, M is also the length of the longest consequent side of a rule of production in the induced grammar. This in turn, means that if the sample Y is of length N, then regardless of how the rules of production are applied to accomplish the recoding of Y, the recoded version of Y cannot be of length shorter than N/M. Since M is generally fairly small, the image of the sample Y under a Recoding is
of a length relatively close to the length of Y itself. This means that Y is not recoded by any one Recoding into an extremely short completely-resolved image. In other words, if we look at the way the sample Y is derived from the rules of production using the ultimate induced grammar, we see that the derivation is indeed a hierarchical derivation—that is, a derivation requiring relatively many intermediate sentential forms (levels). Note that it is the smallness of M which results in this hierarchical characteristic. Note also that the hierarchical grammar thus induced is also not trivial. Thus, both the hierarchical and non-trivial character of the induced grammar follow from the decision to search only for local regularities in the Search Phase.

Finally, note that while certain Unrestricted Rewrite masks can be accommodated in the above procedure (for example, the mask "%%_%%"), there is no provision in general for the development of length-increasing rules of production (that is, rules in which the consequent (right) side is longer than the antecedent (left) side). (See Ginsburg and Greibach [1966]). However, since every rudimentary event (Smullyan [1959]) can be accepted by some linear bounded automaton (Myhill [1960]), and hence context-sensitive language, the formal systems of Smullyan are within the scope of this Algorithm.
4. THE RECODING PROCEDURE:

We noted earlier that a transformation is specified in terms of both (1) a set of grammatical rules of production, and (2) a Recoding Procedure. The Recoding Procedure is an algorithm for scanning the sequence $Y$ for occurrences of the context-sub-sequences of various local regularities, and for specifying in what order the rules of production should be developed from those regularities.

The Recoding Procedure implemented in the computer program for the Grammar Discovery Algorithm admits of numerous variations. These variations will be described in detail as the reason for each possible variation arises. For now, we will consider merely the basic features of Recoding Procedure.

In the Search Phase, masks of various possible lengths, from a length of $M_1$ to a length of $M_2$, were used to find local regularities. We take these same $M_1$ and $M_2$ from the Search Phase for use in the Recoding Procedure. If $M_2$ is the mask of longest length considered in the Search Phase, then the first occurrence of a context-sub-sequence associated with that mask cannot be earlier than position $T = M_2$ in the sequence $Y$. Therefore, the Recoding Procedure starts scanning the sequence $Y$ at position $T = M_2$. The sequence $Y$ is examined for the presence of a sub-sequence of $Y$ of length $M$ which terminates at position $T$ such that

$$Y(T-M+I) = \gamma(I)$$

for each $I = 1, 2, \ldots, M$ for which $\gamma(I)$ is not "%" or "#". In general, context-sub-sequences may be identified with various sub-sequences of $Y$. The purpose of the Recoding Procedure is to select one context-sub-sequence from among the several possibilities. The sub-sequence of $Y$ that is selected is the one which meets the following conditions:
(1) The sub-sequence selected is of longest length, of the lengths between M1 and a parameter MBE which is less than or equal to M2. This range of lengths is further constrained by 2 considerations. First, there must indeed be a sub-sequence in Y of the length L being considered. This means that \( T \geq L \), for each L considered. Also, if the sequence Y has initial punctuation, then the sub-sequence in Y of length L must not include an initial punctuation mark. Note that an initial punctuation mark is always assumed to precede any sequence Y, and the first case is merely a special case of the second.

The second constraint is that none of the positions occupied by the sub-sequence of length L have been recoded as yet on this level. That is, our aim is to find regularities which "cover" the sequence Y no more than once at any position. If multiple covering were allowed at any one level, then (i) the resulting grammar would be ambiguous, and (ii) it might be necessary to establish a precedence ordering among the rules of production of the grammar induced.

(2) The regularity selected is the one whose conditional probability is of the highest type, among those types which are allowed in the Recoding. The lowest allowable type* is specified in advance, and is typically type I, II, or occasionally III.

(3) The regularity selected has highest conditional probability, given condition (2). This is accomplished by the sorting of the conditional probabilities \( P(I) \) associated with the regularity.

(4) The regularity selected is the regularity whose mask is of simplest grammatical type. That is, a regularity arising from

*The variable ALLOW in the computer program specifies the lowest allowable type of regularity to be used in recoding.
a left-sensitive mask will be preferred to a regularity arising from a strictly-context-sensitive mask, etc. Naturally, the type of masks that can be selected are limited by the types of masks used in the Search Phase.

After this local regularity is selected, a rule of production may be developed from it, in the manner described in the preceding section.

The sample Y is then recoded, as specified by that rule of production. Before a new rule of production is introduced however, the possibility of instead introducing a recursive rule of production is considered. The procedure for inducing recursions works with rules of production developed at previous levels of recoding. Thus, if the grammar discovery algorithm is operating on its first level, no test for recursion is performed, and the rule of production developed from the regularity is automatically added to the induced grammar. If the algorithm is already on the second level, or on a higher level, the test for recursion will be performed. This test is fully described in a later section. If a recursion is then induced, one recursive rule of production is added to the induced grammar. This recursive rule replaces one non-recursive rule of production which is already in the induced grammar.

The Recoding Procedure operates so as to minimize the introduction of new rules of production. Thus, when a new non-recursive rule is to be introduced, the entire sequence Y is immediately searched for additional possibilities of applying that rule. These recodings are then performed in preference to any other possible recoding. Similarly, whenever a new recursive rule of production is introduced, a similar back-tracking search is made through all previous levels to see if the recursive rule can be applied. Moreover, with recursive rules,
the Recoding Procedure considers the possibility of immediately re-applying recursive rules (even of the same positions of Y which have just been recoded), so that as much of Y as possible is recoded using the newly induced recursive rule.

Also, the Recoding Procedure operates in such a way that it first scans Y looking only for the possibility of inducing recursive rules of production. After this scan is done, the sequence Y is rescanned for the possibility of inducing non-recursive rules. This double scanning approach given preference to inducing a recursive rule. Because a recursive rule replaces an existing rule already in the grammar, this approach helps minimize the number of rules in the induced grammar.

The Recoding Procedure operates in such a way that the degree of "covering" of the sample is maximized—that is, as many as possible of the symbols of Y are either in the context part or predicted part of some regularity and some rule of production.

Once a non-recursive rule is introduced into the induced grammar, opportunities to reapply that rule are always considered before any further rules are added to the grammar.
5. EXAMPLE

In this section, we illustrate the development of rules of productions from regularities, and illustrate the application of the rules using a recoding procedure.

We begin with an illustration of a context-sensitive rule of production.

Suppose the given sample of sentences is as follows:

```
LEVEL 1

SYMBOL STRING OF LENGTH 137 AND USING ALPHABET OF SIZE 14


THE-BAD-DOG-SAT--
```

Note that the terminal alphabet for the sentences of this sample include 14 Roman alphabet symbols and the period (as initial punctuation).

Suppose further that the following regularity has been identified in the sample:

<"%E", "TH", p>

This regularity is a left-sensitive context-sensitive regularity because the mask from which the regularity was derived is a left sensitive mask (i.e. "%_").
The development of a rule of production from this regularity proceeds as follows: The two contiguous predicted positions (namely positions 1 and 2) are consolidated and replaced by the new non-terminal symbol ("O" in this case). Together with the two context positions, these three symbols become the antecedent (left) side of the rule of production. The consequent (right) side of the rule consists of the four terminal symbols from the regularity. The rule of production is thus

\[ 0E\ldots \rightarrow \text{THE} \ldots \]

In a similar manner, the following 7 rules of production might be developed:

**TENTATIVE RULES OF PRODUCTION FOR M OF 4**

- \[ 0E \rightarrow \text{THE} \ldots \]
- \[ 1G \rightarrow \text{B} \ldots \]
- \[ 2AT \rightarrow \text{C} \ldots \]
- \[ 3AN \rightarrow \text{R} \ldots \]
- \[ 4D \rightarrow \text{R} \ldots \]
- \[ 5G \rightarrow \text{D} \ldots \]
- \[ 6AT \rightarrow \text{S} \ldots \]

Applying these rules of production to the sample, one recodes the sample and obtains the following:

**NEW STRING**

- \[ 0E \ldots 1G \ldots 2AT \ldots ?A\ldots 0E \ldots 4D \ldots 2AT \ldots 3AN \ldots 0E \ldots 1G \ldots 5G \ldots 3AN \ldots 0 \]
- \[ E \ldots 1G \ldots 2AT \ldots 6AT \ldots 0E \ldots 4D \ldots 2AT \ldots 6AT \ldots 0E \ldots 1G \ldots 5G \ldots 6AT \ldots 0 \]
Note that the rules of production in this example were context-sensitive because the mask of the regularity from which the rules were derived were context-sensitive masks.

It would seem that only context-sensitive rules of production can be derived. Such is not the case. Later, after motivating and defining the concept of maximal regularity (in section III.B.3 and IV.D), we will develop context-free rules of production. However, it should be noted now that if there is a procedure for developing context-free rules of production, then their application to the given sample, using the recoding procedure, is exactly the same as with context-sensitive rules of production.

To illustrate this statement, recall example B from the previous section. The sample in Example B was the following binary sequence:

```
LEVEL 1
---------------------------
SYMBOL STRING OF LENGTH 272 AND USING ALPHABET OF SIZE 2

11100110110010001100010111010100:CO00CC011100100111101010100
000011100111100100011010100000C11C001111011011011000100
111C001011000111101011011100011100C1000000110010001110
01001101010011000011101010100000110001011110010111100101
11010101111001111001001000010001111111
```
Suppose there is a justification for developing context-free rules of production in the Grammar Discovery Algorithm. Suppose that the following four context-free rules are developed:

\[
\text{TENTATIVE RULES OF PRODUCTION FOR } M = 2
\]

\[
\begin{align*}
A & \quad \longrightarrow \quad 11 \\
B & \quad \longrightarrow \quad 10 \\
C & \quad \longrightarrow \quad 01 \\
D & \quad \longrightarrow \quad 00
\end{align*}
\]

Then the recoding of the given sample, using these four context-free rules of production and the recoding procedure, would yield the following new sample of sentences:

\[
\text{NEW STRING}
\]

\[
\text{ABCBADBDACCACCDCCDDDBACADCDACDDDAADDAACCBADCAADBADCD}
\]

\[
\text{ADDABCCDDACBCBAACCDCDDEACBACDDACCDADDCCACCCDDDDDCCABCCABCC}
\]

\[
\text{ACCCACCAABBCDCCD}
\]

In the examples that accompany several of the following sections of this paper, we will use context-free rules of production in many cases, although the rationale for developing these rules of production will not be fully developed until sections III.B.3 and IV.D.
D. THE SELECTION PHASE—SELECTING RECODINGS

1. INTRODUCTION

The third main phase of the Grammar Discovery Algorithm is the Selection Phase. The Selection Phase oversees several possible trial recodings and selects one which is then actually applied to the sample of sentences to produce one new set of sentences. This selection is made on the basis of three criteria: the entropy, the parsimony, and the recursive parsimony of the transformation.

In this section, we define the concepts of entropy of a transformation, parsimony of a transformation, and recursive parsimony of a transformation. We define a resolving transformation, and we present the algorithm for trying recodings, evaluating them, and then selecting one recoding as the actual recoding to be performed on the sample. The idea of the $P_{ij}$-graph is developed.
2. ENTROPY, PARSIMONY, RECURSIVE PARSIMONY OF A TRANSFORMATION

Typically, there is more than one rule of production in each transformation. Each rule of production is derived from a local regularity. Recall that each regularity $\mathcal{R} = \langle \gamma, \phi, p_{ij} \rangle$ consists of a context-sub-sequence $\gamma$, a predicted-sub-sequence $\phi$, and an associated conditional probability $p_{ij}$. We define the entropy of a transformation to be

$$- \sum p_{ij} \log_2 p_{ij}$$

(using the convention that $0 \log_2 0 = 0$), where the $p_{ij}$ are the conditional probabilities associated with the regularities from which the rules of production of the transformation were derived. Note that the sum here is taken over the (reduced) set of rules of production. Note that this measure depends only on knowing the set of rules of production of the transformation.

Note, for example, that if each local regularity from which the rules of production of the transformation was developed is a structural regularity (i.e. the context-sub-sequence predicts the predicted-sub-sequence with 100% reliability), then the entropy of the transformation will be zero. As an illustration of this, consider the string $Y$

$$\ldots \text{abcdefghefghabcdefgh} \ldots$$

This string might be recoded using the two strictly-context-sensitive rewrite rules

$$\text{aNd} \to \text{abcd}$$

$$\text{eMh} \to \text{efgh}$$

which each have 100% reliability. The resulting string $Y'$ would then be

$$\ldots \text{aNdMheMh}aNdMh \ldots$$
Note that no information is lost by this transformation for the positions actually recoded.

On the other hand, if the local regularities used to develop the rules of production of a transformation all had conditional probabilities of $\epsilon = 1/c^g$ (i.e. where the occurrence of the context tells us "nothing" about the "predicted" part), then the entropy of such a transformation would be maximal. Information is lost by such a transformation; the original sequence $Y$ cannot be recovered by applying any kind of inverse of production of this transformation to the encoded string $Y'$. The information lost would be maximal for the positions actually encoded. For example, suppose 001 is followed with 00 or with 11 with probability 1/2, and that the rule of production is

$$001A \rightarrow 00100.$$ 

$Y$ may be the sequence $...00100,00111,00100,00111...$ (the commas are added here only as a visual aid) and $Y'$ may be $...001A,00111,001A,001A...$. Here $Y$ cannot be recovered from $Y'$.

The weighted entropy of a transformation is the same sum as the entropy of a transformation, except that the sum is taken over each application of the rules of production. Thus, rules that are used more often in the actual recoding count more in this sum. Note that this measure is dependent on both the set of rules of production and also the Recoding Procedure—which determines how many times each rule is applied.

The fewer rules of production in a grammar, the more parsimonious it is.

Accordingly, we define the parsimony of a transformation as $\lambda \cdot n$, where $n$ is the number of (reduced) rules of production, and
where \( \lambda \) is a positive real number called the *coefficient of parsimony.*

Note that \( n \) is a measure of the complexity of the transformation, and indirectly of the induced grammar (Goodall).

To illustrate the idea of entropy and parsimony, consider the following trivial grammar which can be inferred from any given sample of sentences \( a_1, a_2, \ldots, a_n \) over an alphabet \( V_T \). The grammar is \( G = \langle V_T, S, S, P \rangle \) where the rules of production \( P \) are the \( w \) rules:

\[
\begin{align*}
S & \rightarrow a_1 \\
S & \rightarrow a_2 \\
& \quad \ldots \\
S & \rightarrow a_w
\end{align*}
\]

In this case, the entropy of the transformation here is zero, because all transformations are 100% reliable. However, the parsimony here is high — namely \( \lambda w \).

We define the *combined entropy-parsimony measure* of a transformation to be

\[
- \sum P_{ij} \log_2 P_{ij} + \lambda \cdot n
\]

Note that there is a trade-off between the entropy and the parsimony of a transformation. The combined entropy-parsimony measure for 2 different transformations can be the same if the one with more rules of production has correspondingly less entropy (i.e. if the regularities used to develop the rules of production of the transformation have greater reliability).

Note also that while probability of occurrence of the different contexts is not explicitly considered in the Grammar Discovery Algorithm (because conditional probabilities are used throughout), probabilities of occurrence are reflected implicitly in the following

*The variable \( \text{LAMBDA} \) in the computer program is the coefficient of parsimony.*
way: A rule of production developed from a regularity whose context part that occurs rarely in the sample will not be applied often in transforming the sample, and therefore other rules of production will be needed to transform the remainder of the sample, and therefore parsimony will be less. The ideas of "entropy" and "parsimony" were suggested by a paper of Goodall (1962). However, it is likely that Goodall's entropy was a measure of information (and therefore was dependent on probabilities of occurrence and not conditional probabilities). The idea of a combined measure and a trade-off between these two quantities appears in Goodall (1962).

One recursive rule of production generally replaces many (indeed, an infinity) of non-recursive rules of production.

The recursive parsimony of a transformation is defined as

$$\lambda_r \cdot n_r$$

where $n_r$ is the number of recursive (reduced) rules of production, and where $\lambda_r$ is a positive real number, called the coefficient of recursive parsimony.*

The combined entropy-parsimony-recursive-parsimony measure of a transformation is defined to be

$$- \sum p_{ij} \log_2 p_{ij} + \lambda_r \cdot n_r + \lambda_r n_r$$

Typically, recursive rules of production are more desirable than non-recursive rules. Thus, $\lambda_r$ would be chosen so that

$$\lambda_r << \lambda$$

so as to give relatively greater weight in the measure to the less desirable non-recursive rules. One might even assign a value of zero to recursive parsimony.

*The variable LAMR in the computer program is the coefficient of recursive parsimony.
Note that in all of the foregoing cases, the identity transformation would have no entropy at all, since the application of an identity transformation is a completely reliable recoding. The combined measure of entropy, parsimony, and recursive parsimony actually used in the computer program implementing the Grammar Discovery Algorithm is an *ad hoc* combination of the above features. We have not attempted to derive the particular function chosen from a series of desired axiomatized properties, or to show that the function (or its class) is the only function (or only class of functions) that satisfies the axioms. The properties that the function does, however, satisfy are as follows:

(1) A rule of production developed from a structural (Type I) regularity should have zero entropy. Thus, rules developed from either type I regularities or identify transformations will contribute zero to entropy.

(2) Each rule of production should count equally against parsimony. Certainly, measuring grammatical complexity can be done in other ways (and better ways) than counting rules of production; however, we will use this simple approach suggested by Goodall (1962) here. One important implication of this requirement is that each different identity transformation should count as a rule in determining parsimony. Thus, we do not treat the identity transformation as one transformation any more that we treat any other rule of production having different predicted parts as being the same. Note that if identity transformation did not count against parsimony, they would not contribute to the H measure at all (since their entropy is zero); and, if that were the case, the transformation with minimal H would always be the identity transformation on the whole sample, and the initial sample would then be perpetuated from
level to level.

(3) Each recursive rule of production should count equally against recursive parsimony.

(4) The entropy for each application of each rule should be normalized to reflect the number of symbols recoded with it. If this normalization were not done, longer rules would tend to be favored over shorter rules. Carried to the extreme, the best recoding would be accomplished with very long rules—rules so long that they are 100% reliable solely by virtue of their capturing artifacts of the sample.

(5) The relative weights to be given to the 3 attributes of entropy, parsimony, and recursive parsimony should be determined by a weighted sum. The trade-off should be between entropy on the one hand and the two kinds of parsimony on the other.

The weighted, normalized, combined entropy-parsimony-recursive-parsimony used in the computer program is therefore

\[ -M_{ij}\sum P_{ij} \log P_{ij} + \lambda_n n + \lambda_r n_r \]

where this sum is taken over each application of each rule of production, and where \( M_{ij} \) is the length of the regularity associated with \( P_{ij} \). Note that

\[ \sum M_{ij} = N, \]

where \( N \) is the length of \( Y \).

The idea of reliably recoding, or "bunching together," of portions of a sample of data is found in a variety of settings. It is well known in psychology and physiology [Miller, 1956] that human beings are well able to classify gradations in pitch, hue, loudness, count, smell, taste, and a variety of sensory information into about 7 categories. If faced with much more than 7 categories, humans lose the ability to distinguish reliably and must "bunch" the data together so
that items within the bunch may first be distinguished and gradated. Then, at the next higher level, representatives of the bunches may then be distinguished and gradated.

The pervasiveness of this number 7 (plus or minus 2) seems also to extend into the social and political behavior of humans. As C. Northcote Parkinson (1957) has noted in an examination of the English Privy Council and other cabinet bodies, human decision-making bodies tend to function best with about 8 members. When more than this number are added to a committee, an inner committee of smaller size tends to form to do the actual decision-making and to integrate the views of factional representatives from the larger body.

Although not within the scope of this study, it seems possible that, for humans, the tradeoff between parsimony and entropy comes when more than 7 items must be integrated. This level (of parsimony) in turn implicitly defines the degree of faithfulness of representation (entropy) which necessarily must be accepted by humans in a variety of sensory and social and political settings.
3. THE $p_{ij}$-GRAPH OF A TRANSFORMATION

Whenever we have a collection of probabilities from a set of regularities abstracted from a given sample, we may not only classify them as to type (i.e. type I, II, III, IV, and V), but we may sort them into descending order by magnitude. Such a graph, which we call the $p_{ij}$-graph has the general appearance seen below:
In the figure, \( \sigma \) denotes the permutation of the \( F_m p_{ij} \)'s which sort them into descending order. \( \sigma(i) \), where \( i \) is an integer from 1 to \( F_m \), is the ordered pair \((i,j)\) that identifies the \( p_{ij} \). Goodall (1962) presented such a graph, although apparently erroneously using the actual probabilities of occurrence. These probabilities, of course, are rarely near 1.0. However, if we instead consider conditional probabilities (as we do herein) or even relative probabilities (perhaps normalizing all probabilities of occurrence of an M-gram by dividing by the largest), there are several provocative and valid points in Goodall. As always, the Goodall terminology is highly metaphorical and suggestive. The high probabilities at the left end of the graph represent the "structure" in the sample. The structure is that portion of the sample that is reliably present. Structure is associated with the type I or II regularities in the sample. Structure may be such facts as Q's are always followed by U's in English; that periods are reliably followed by spaces; that most birthdays in the social security files have a "19" in them; or that certain self-synchronizing codes have a certain punctuating symbol at the beginning of each phrase (e.g. the peak of a sawtooth wave at the beginning of a television line image). Since the structure is reliable, all of the structure may be abbreviated and replaced by a shorter marker (or even removed altogether!). What is structure at one level is structure at the next level--regardless of how it is recoded, abbreviated, or deleted.

The message portion of a sample is the part that is unpredictably variable at this level of analysis. The entropy of this portion of the sample is high because this is the part of the sample which contains the information. A key point is that what is "unresolvable message"
at one level is not necessarily unresolvable at a higher level. Unlike structure, message is not always message. Of course, message cannot be recoded (except by an identity transformation) because it has no as-yet-identified internal structure? Message corresponds to the type III and IV regularities in the sample.

The noise portion of the sample is that part of the sample which occurs with very low probability. Noise corresponds to the type V regularities—indeed, the "irregularities." Noise represents all the exceptions, contradictions, error, and randomness of the sample—things which occur so rarely that their non-occurrence in the sample is so predictable that they cannot be considered message (which is unpredictable variability). Whether noise represents "error" (as, for example, would be the case in an incoming encoded message) or "exceptions" (that is, information which indeed is correct, but rare) depends on the nature of the sample. Accordingly, one would want to preserve exceptions for analysis at a higher level and discard error as information that is incompatible with the sample. This is an external decision.

On the $p_{ij}$-graph, structure, message, and noise appear as one proceeds from left to right.

The $p_{ij}$-graph described earlier is of greatest interest when we plot only the $p_{ij}$'s used to develop rules of production that are actually used in a transformation. We call this graph the $p_{ij}$-graph of a transformation.

*There is one possible exception to this statement which is discussed in the sentence-oriented method for inducing disjunctions. The exception occurs when a message sequence is one of an ensemble of possible substitution instances for a predicted sub-sequence that has maximal or near maximal entropy. See II.4.
For example, if our initial sample is

```
LEVEL 1
SYMBOL STRING OF LENGTH 222 AND USING ALPHABET OF SIZE 2
1100110110010011001011010101010100000110110011010101010100
0001110011110100011010011011000010011010000110011101
110101011011000111011010010100000110001011110010111110010
1101010111100111100101010100000110001011110010111110010
```

and we use the following 4 rules of production using an $M$ of 2

```
TENTATIVE RULES OF PRODUCTION FOR M OF 2
A -----> 11
B -----> 10
C -----> 01
D -----> 00
```

we might find the following values for entropy and parsimony:

```
ENTROPY TERM 65.18735
PARSIMONY TERM 4.00000
RECURSIVE PARSIMONY TERM 0.0
VALUE OF H FOR THIS RECODING 69.18735
NUMBER OF RULES OF PRODUCTION 4
NUMBER OF RECURSIVE RULES 0
NUMBER OF TIMES RULES ARE APPLIED 136
```

If we then apply the rules of production, using a standard recoding procedure, we would obtain a $p_{ij}$-graph such as is found on the next page. This recoding results in the following new string:

```
ABCDEBADCCACCCDDDBABDAADBADCCDDDBADDAACCBADCD
ADDBADCCDDDBACBACBDDCABDDAABACDCAACCCDDDBADCCABCCABCC
ACCCABDAAABBCDDDD
```
Graph of p(I) used in reccding for level 1 and M of 2

1.00  T+++ 
0.98  | I |  
0.96  | I |  
0.94  | I |  
0.92  | I |  
0.90  | I |  
0.88  | I |  
0.86  | I |  
0.84  | I |  
0.82  | I |  
0.80  | I |  
0.78  | I |  
0.76  | I |  
0.74  | I |  
0.72  | I |  
0.70  | I |  
0.68  | I |  
0.66  | I |  
0.64  | I |  
0.62  | I |  
0.60  | I |  
0.58  | I |  
0.56  | I |  
0.54  | I |  
0.52  | I |  
0.50  | I |  
0.48  | I |  
0.46  | I |  
0.44  | I |  
0.42  | I |  
0.40  | I |  
0.38  | I |  
0.36  | I |  
0.34  | I |  
0.32  | I |  
0.30  | I |  
0.28  | I |  
0.26  | I |  
0.24  | I |  
0.22  | I |  
0.20  | I |  
0.18  | I |  
0.16  | I |  
0.14  | I |  
0.12  | I |  
0.10  | I |  
0.08  | I |  
0.06  | I |  
0.04  | I |  
0.02  | I |  
0.0  | I |  

3334

Number of type 1 sequences (structural) 0
Number of type 2 sequences (message) 0
Number of type 3 sequences (message) 3
Number of type 4 sequences (message) 1
Number of type 5 sequences (noise) 0
Number of p(I) 4
4. RESOLVING TRANSFORMATIONS

If the entropy is not large for a transformation, it must be that \( p_{ij} \)'s are mostly near 1.0 or near 0.0.

Let \( g \) be the largest number of predicted positions for any mask in a transformation. This number is always less than or equal to \( M-1 \) for the largest \( M \) tried in the Search Phase. The criterion for a transformation is the value

\[
- \frac{1}{c^g} \log_2 \frac{1}{c^g}
\]

A resolving transformation is a transformation for which the entropy is less than the criterion.

This suggestive term is found in Goodall (1962). The idea there that when a recoding satisfies the criterion, the underlying structure that generated the sample has been identified; and, therefore, the sample is "resolved."

Note that if one of our measures which combine entropy and parsimony is used, some \( p_{ij} \)'s may be smaller than \( 1 - \epsilon \); or, if a transformation which is not a resolving transformation is considered, some \( p_{ij} \)'s may be smaller than \( 1 - \epsilon \). In both these cases, however, most \( p_{ij} \)'s will tend to be near 1, and exceptions will occur only because the introduction of that \( p_{ij} \) leads to such a substantial reduction in the number of rules or production that the increase in entropy is justified. In all cases, however, most of the \( p_{ij} \)'s will be bunched near 1.0.

In the \( p_{ij} \)-graph of a transformation actually used in recoding there will be no \( p_{ij} \)'s that are less than \( \epsilon \) because we never use regularities so unreliable in a transformation.
In the case of a resolving transformation, there can be no $p_{ij}$'s equal to $\varepsilon$ because one such $p_{ij}$ would alone make the entropy of the transformation larger than the criterion.

These last 2 observations define the features of any $p_{ij}$-graph of a resolving transformation.
5. TRIAL RECODINGS AND THE SELECTION OF THE ACTUAL RECODING

M1 and M2 are the lower and upper limits, respectively, on the length of context-sub-sequences and predicted-sub-sequences considered in the Search Phase in the search for local regularities. For each M between M1 and M2, a different recoding results, in general, if we allow only regularities of length up to M to be used in developing rules of production in the recoding. Thus, M2-M1+1 recodings are defined. These recodings are parameterized by the index MBE.

In the Selection Phase, these M2-M1+1 different trial recodings are each attempted, and the entropy, parsimony, and recursive parsimony of each computed. The recoding which is selected to be the actual recoding is the one which is associated with the first M (considering the direction of considering the M's*) which is a resolving transformation**, or the M which has the best combined entropy-parsimony-recursive-parsimony measure (if it is not actually a resolving transformation). The best combined measure is the least combined measure.

The strings resulting from the application of the actual recoding constitute the strings of the next level of the process. The Search Phase, the Recoding Phase, and the Selection Phase of the Grammar Discovery Algorithm are then applied to the strings of this level, until the Algorithm terminates.

*The variable VDIR specifies this order in the computer program.

**The variable WDF in the computer program determines whether the first resolving transformation discovered is used for the actual recoding, or whether (if there is more than one resolving transformation) the best resolving transformation is used.
For example, if we again refer to the example started on page 68, we find that for an \( M \) of 3, we develop the following 8 rules of production:

\[
\begin{align*}
A & \rightarrow 111 \\
B & \rightarrow 001 \\
C & \rightarrow 101 \\
D & \rightarrow 100 \\
E & \rightarrow 011 \\
F & \rightarrow 000 \\
G & \rightarrow 110 \\
H & \rightarrow 010
\end{align*}
\]

The application of these 8 rules to the sample yields the following new string:

\[
ABCDDEFGCHDDEGBGCHDDEGEHBDHCCFDFGACEEFDFCD+CFGCCDBFCCGHB
\]

The entropy and parsimony for this transformation are as follows:

\[
\begin{align*}
\text{ENTROPY TERM} & \quad 42.55614 \\
\text{PARSIMONY TERM} & \quad 6.00000 \\
\text{RECURSIVE PARSIMONY TERM} & \quad 0.0
\end{align*}
\]

\[
\text{VALUE OF H FOR THIS RECODING} \quad 50.55614
\]

\[
\begin{align*}
\text{NUMBER OF RULES OF PRODUCTION} & \quad 8 \\
\text{NUMBER OF RECURSIVE RULES} & \quad 0 \\
\text{NUMBER OF TIMES RULES ARE APPLIED} & \quad 90
\end{align*}
\]

And, the \( p_{ij} \)-graph for the transformation is shown on the next page.
1.00 ———— t
0.98  |  |
0.96  |  |
0.94  |  |
0.92  |  |
0.90  |  |
0.88  |  |
0.86  |  |
0.84  |  |
0.82  |  |
0.80  |  |
0.78  |  |
0.76  |  |
0.74  |  |
0.72  |  |
0.70  |  |
0.68  |  |
0.66  |  |
0.64  |  |
0.62  |  |
0.60 *  |  |
0.58 *  |  |
0.56 *  |  |
0.54 *  |  |
0.52 *  |  |
0.50 ————++
0.48  |  |
0.46  |  |
0.44  |  |
0.42  |  |
0.40  |  |
0.38 *  |  |
0.36 *  |  |
0.34 *  |  |
0.32  |  |
0.30  |  |
0.28  |  |
0.26  |  |
0.24  |  |
0.22  |  |
0.20  |  |
0.18  |  |
0.16  |  |
0.14  |  |
0.12  |  |
0.10  |  |
0.08  |  |
0.06  |  |
0.04  |  |
0.02  |  |
0.0  ————++
333333344

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) 0
NUMBER OF TYPE 2 SEQUENCES (MESSAGE) 0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE) 6
NUMBER OF TYPE 4 SEQUENCES (MESSAGE) 2
NUMBER OF TYPE 5 SEQUENCES (NOISE) 0
NUMBER OF P(I) 8
Similarly, for an M of 4, we find the following rules:

TENTATIVE RULES OF PRODUCTION FOR M OF 4

A ----> 1110
B ----> 0110
C ----> 1100
D ----> 1000
E ----> 0101
F ----> 1101
G ----> 0100
H ----> 0000
I ----> 1001
J ----> 0011
K ----> 0010
L ----> 0001

and the following evaluation:

<table>
<thead>
<tr>
<th>Term</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entropy Term</td>
<td>30.02759</td>
</tr>
<tr>
<td>Parsimony Term</td>
<td>12.60000</td>
</tr>
<tr>
<td>Recursive Parsimony Term</td>
<td>0.0</td>
</tr>
<tr>
<td>Value of H for this recoding</td>
<td>42.02759</td>
</tr>
</tbody>
</table>

and the following new string:

ABCDCEFEHGCHIFEGHAIJDCDEGHCF8CGCKCEGHBCCAGFLFEHGCEAEAE
FEAJAKGH

The $p_{ij}$-graph is found on the next page.
GRAPH OF P(I) USED IN RECODING FOR LEVEL 1 AND M OF 4

1.00 +---------+
0.98 |       |
0.96 |       |
0.94 |       |
0.92 |       |
0.90 |       |
0.88 |       |
0.86 |       |
0.84 |       |
0.82 |       |
0.80 |       |
0.78 |       |
0.76 |       |
0.74 |       |
0.72 |       |
0.70 |       |
0.68 |       |
0.66 |       |
0.64 |       |
0.62 |       |
0.60 |       |
0.58 |       |
0.56 |       |
0.54 |       |
0.52 |       |
0.50 +---------+
0.48 |       |
0.46 |       |
0.44 |       |
0.42 |       |
0.40 |       |
0.38 |       |
0.36 |       |
0.34 |       |
0.32 |       |
0.30 |       |
0.28 |       |
0.26 |       |
0.24 |       |
0.22 |       |
0.20 |       |
0.18 |       |
0.16 |       |
0.14 |       |
0.12 |       |
0.10 |       |
0.08 |       |
0.06 |       |
0.04 |       |
0.02 |       |
0.00 +---------+

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) 0
NUMBER OF TYPE 2 SEQUENCES (MESSAGE) 0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE) 8
NUMBER OF TYPE 4 SEQUENCES (MESSAGE) 4
NUMBER OF TYPE 5 SEQUENCES (NOISE) 0
NUMBER OF P(I) SEQUENCES (NOISE) 12
If it seems that the combined entropy-parsimony measure \( H \) is merely a monotonic function of \( M \), consider what happens with an \( M \) of 5. Here we get an extraordinary number of rules---27 in all---as shown below:

**TENTATIVE RULES OF PRODUCTION FOR \( M \) OF 5**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>01000</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>11100</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>01100</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>11010</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>01010</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>11001</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>00111</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>10100</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>00001</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>11000</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>11110</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>01000</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>10110</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>00010</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>00110</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>00011</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>01110</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>01001</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>11010</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>01111</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>00101</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>10111</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>10001</td>
<td></td>
</tr>
<tr>
<td>( \langle AA \rangle )</td>
<td>10000</td>
<td></td>
</tr>
</tbody>
</table>
For M of 5, the parsimony is therefore quite large. However, the entropy is quite small. With an M as large as 5, there is sufficient context to guarantee a more faithful (and hence reliable, and hence lower entropy) transformation. However, this greater faithfulness is the result of the proliferation of rules of production. There are more rules, each used less often, and each used more reliably. The following is the calculation of H:

<table>
<thead>
<tr>
<th>Term</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entropy Term</td>
<td>22.21033</td>
</tr>
<tr>
<td>Parsimony Term</td>
<td>27.00000</td>
</tr>
<tr>
<td>Recursive Parsimony Term</td>
<td>0.0</td>
</tr>
<tr>
<td>Value of H for this recoding</td>
<td>49.21033</td>
</tr>
</tbody>
</table>

And, the following is the new string that results:

ABCD EFGHIJKLMNOPQRSTUVWXYZ

(Note that when we exhaust the non-terminal alphabet provided, we use brackets as in Backus-Naur Form to indicate non-terminals).

The $p_{ij}$-graph for an M of 5 is found on the following page.
Note that for $M$ of 5, we have some type II regularities in the transformation.

Reviewing this example, note that the smallest $M$ is attained with an $M$ of 4. Thus, after considering trial recodings for $M$ of 2, 3, 4, and 5, we would select the recoding based on an $M$ of 4 for the actual recoding. We then would use the new string obtained by using the recoding based on an $M$ of 4 as the string for input to level 2 of the Algorithm.
E. INDUCING RECURSIONS FROM A FINITE SAMPLE OF SENTENCES

1. INTRODUCTION AND MOTIVATION

Every sample of sentences we encounter is finite; and every finite sample of sentences can be generated by a finite state grammar. In fact, a finite state grammar can be trivially found which generates the sentences in the sample and only those sentences—thus producing the best possible fit between the sample and the grammar induced. A grammar which generates only a finite number of sentences is a finite cardinality grammar, and a language with only a finite number of sentences is a finite cardinality language. A finite cardinality grammar need not be a finite state (regular) grammar, and a finite state (regular) grammar need not be a finite cardinality grammar.

Of course, for reasons of economy and to satisfy our own intuitive requirements, we insist that the grammar induced by a grammar discovery algorithm not always be either a finite state grammar or a finite cardinality grammar. Thus, we allow the induction of non-finite-state-grammars—that is of left-sensitive, right-sensitive, strictly context-sensitive, and strictly unrestricted rewrite grammars—all from a finite sample (which is, of course, a finite cardinality language itself). Similarly, we allow the induction of non-finite-cardinality-grammars from a finite sample. The only way for a grammar (by which we mean, of course, a finite grammar—i.e. one with a finite number of rules) to generate more than a finite number of sentences is for the grammar to have a recursion. Allowing the induction of recursions from a finite sample is as important as allowing the induction of non-finite-state rules of production from a finite sample.

Indeed, without recursion, many simple relations must be expressed
by numerous different rules of production. Moreover, if the sample of sentences is extended—say by application of a recursion—and we are not allowed to induce grammars with recursions, then new rules of production must then be added to explain the extension. Each extension leads to a proliferation of additional rules of production. In the case of this kind of extension of the sample, a grammar which did not contain a recursion would be neither economical, nor stable (under the extension).
2. DEFINITION OF A RECURSION

Given a grammar \( G = \langle V_N, V_T, P, S \rangle \). Let \( V = V_N \cup V_T \). Let \( w \) and \( y \) be strings over \( V^* \). We say that \( w \) immediately derives \( y \), which we write as
\[ w \Rightarrow y, \]
if there exists a \( z_1, z_2, u, \) and \( v \), all in \( V^* \), such that \( w = z_1 u z_2 \)
and \( y = z_1 v z_2 \), and such that \( u \rightarrow v \) is a rule of production in \( P \).

We say that \( w \) derives \( y \), which we write as \( w \Rightarrow^* y \), if there exists strings \( w_0, w_1, \ldots, w_n \) such that \( w = w_0 \), and such that \( y = w_n \),
and such that \( w_i \Rightarrow w_{i+1} \) for each \( i \) from 0 to \( n-1 \). A sentential
form is a string \( w \in V^* \) such that \( S \Rightarrow w \). A sentence is a string
\( w \in V_T^* \) such that \( S \Rightarrow w \).

A grammar is said to contain a recursion if there is a non-terminal \( N \in V_N \) such that
\[ \alpha N \beta \Rightarrow^* \gamma N s \]
where \( \alpha, \beta, \gamma, s \in V^* \), provided that it is not the case that \( \alpha = \gamma \)
and \( \beta = s \).

Note that rules such as \( N \rightarrow N \) and \( N \rightarrow N \) and \( NM \rightarrow MN \) do not
count as recursions in the above definition. We use only the noun
"recursion" in referring to the above idea. An individual rule of
production \( \alpha N \beta \rightarrow \gamma N s \) is called a recursive rule if the same non-termi-
inal \( N \) appears in both the antecedent (left) side and consequent
(right) side of the rule, and provided that it is not the case that
\( \alpha = \gamma \) and \( \beta = s \), and provided that \( |\alpha N \beta| \neq |\gamma N s| \).
3. APPROACHES TO INDUCING RECURSIONS

There are several possible approaches to inducing recursions from a finite sample.

Feldman (1966, 1967) proposed an algorithm for inferring an unambiguous, recursive regular grammar from a given sample. This algorithm begins with the construction of a non-recursive, regular, grammar that generates exactly the given sentences, then merges non-terminals in this grammar to get a recursive regular grammar which seems to be a "reasonable generalization" of the sample. The algorithm ends with a simplification process.

To generate the intermediate grammar which exactly generates the given sentences, Feldman processes the strings of the sample in order of decreasing length. To the extent that the sample does not have different sentences of equal length, this procedure eliminates the effect of order of the sentences within the sample. Rules of production are developed one-by-one, as they are needed to generate each sentence in the sample in turn. Specifically, starting with a (the) longest sentence $a_1 a_2 \ldots a_n$ of length $n$, Feldman would generate $n-1$ rules as follows: The first rule is $S \rightarrow a_1 A_1$. The next $n-3$ rules are $A_i \rightarrow a_{i+1} A_{i+1}$, for $i = 2, \ldots, n-2$. The $n-1$-th rule is $A_{n-3} \rightarrow a_{n-1} a_{n-2}$ --which is a rule which is not in the form of a rule of a regular grammar, and which he calls a "residue" rule. "Residue" rules in general are rules of the form $A \rightarrow w$, where $V_T, /w/ \geq 2$. New rules are added as necessary as additional sentences from the sample are considered.

As each sentence is considered, new rules of production are added only to the extent required to guarantee the generation of that sentence. These new rules may even be terminating rules—which are like
"residue" rules except that their consequent (right) side is of length 1. Terminating rules come about when all but the last symbol of the sentence under consideration can be generated with the previously developed rules. The rules so developed may be combined and written non-deterministically.

To obtain recursion, rules of production are now merged. Each residue rule is merged with a non-residue rules, thus eliminating the "residue" rules. The general principle is that after such merging, the resulting grammar must still generate all the sample, plus as few new short sentences as possible. This merging is accomplished as follows: Whenever the non-terminal on the antecedent (left) side of a residue rule occurs on the consequent (right) side of a non-residue rule, the non-terminal of the left side of this non-residue rule is substituted for the non-terminal on its consequent (right) side which was in common with the residue rule. This eliminates the residue rule, and makes the non-residue rule involved into a recursive non-residue rule. Note that the resulting grammar is now entirely in the form of a regular grammar. Also, note that this procedure guarantees that the longest sentence of the sample is generated by a recursive rule. Shorter sentences of the sample may be generated either with the aid of a recursive rule, or only by non-recursive rules. The tendency is that the shorter the sentence, the more non-recursive rules have been constructed; hence, it is more likely that it can be generated by a sequence of non-recursive rules, followed by one terminating rule.

The non-residue recursive grammar thus produced is now simplified to remove equivalent productions. This simplication results in no change in the language generated.
A second algorithm by Feldman (1969) infers a "pivot" grammar for a given set of sentences. A pivot grammar is an operator grammar in which a terminal symbol which separates non-terminals in a production appears in no other way. Linear grammars are a special case of pivot grammars, but the class of pivot grammars is much broader than the linear grammars.

The algorithm begins with the sample of sentences and the knowledge of which terminal symbol(s) are the pivot terminal symbols. The algorithm produces a pivot grammar.

The main strategy of the algorithm is to find self-embeddings. Each sentence is examined to see if it has a proper substring which is also a sentence of the sample. If it does, a "loop non-terminal" is substituted for the longest such substring. This results in a new sentence. This new sentence begins part of the sample under consideration. If no sentences have such substrings, the sample is scanned to see if all sentences have the same first symbol, or the same last symbol. If this is the case, the common symbol is trimmed off, and the process described above is then applied to the trimmed sentences.
4. SENTENCE-ORIENTED METHOD OF INDUCING RECURSION

The methods for inducing recursions can be divided into two categories. First, there are methods that operate on the sentences of the sample—that is, the methods that look for some indication in the sample that a recursive rule of production is justified. Second, there are methods that operate on the rules of production developed at each level of the grammar discovery algorithm.*

We examine the sentence-oriented approaches first.

A theorem of Bar-Hillel et al. (1962) states that for every context-free language L, there exist constants p and q, depending only on L, such that if there is a sentence z in L of length greater than p, then z may be written as a concatenation uvwxy, where

\[ |vwx| \leq q, \text{ with } |vw| > 0, \text{ and further that each sentence } \]

\[ u^i v^i w x^i y \]

is in L for \( i \geq 0 \). Thus, if L contains one sufficiently long sentence, an infinity of other long sentences are stated to also be in L.

A proof of this theorem appears in Hopcroft and Ullman (1969) under the name "uvwxy theorem."

The special case of the uvwxy theorem for regular languages is a well-known result, usually presented in discussions of whether a given regular language is finite or infinite. This result is the theorem stating that if a regular language contains a sentence of length greater than the number of non-terminal symbols in the grammar (which is the same as the number of states in the finite automaton associated with the grammar), then the language is infinite, and in fact contains an infinity of sentences of the form

*The variable WCV specifies which approach is to be used in the computer program, only the rule-oriented approach. The variable KRECUR specifies whether any attempt to induce recursions should be made.
The uvwxy theorem, and its proof, suggest a sentence-oriented approach for inducing the presence of a recursion in a given sample of sentences. If the sample of sentences is sufficiently rich and varied, there should be some instances in the sample of substrings of the form $v^iwy^i$. We should attempt to recognize these instances. In a sentence, a substring such as $v^i$ is the product of recursion. This sub-string can be generated by the application of some recursive rule $i$ times.

The detection of $v^i$ can be accomplished with masks of even length $M$, wherein the first $M/2$ positions of the mask are context positions, and the last $M/2$ positions are predicted positions—

\[
\frac{M}{2} \mod \frac{M}{2}
\]

(A mask with the context and predicted symbols reversed works equally well). The given sample of sentences at the current level is searched for regularities associated with this series of masks of even length. The only occurrence of regularities associated with this mask can be over those strings where there is a repeated sub-string. The conditional probability associated with this regularity should be near 1.0 for a recursion to exist. That is, most occurrences of the sub-string should be followed by another occurrence of the same sub-string—the only exception being the last occurrence within the sentence. Note that the conditional probability cannot be exactly 1, unless the entire sample consists of repetitions of the sub-string.

If the recursive rule is a non-self-embedding rule, then we would find only $v^i$ in a sentence. If the recursive rule is a self-embedding one then there would also be an $x^i$ sub-string.
Whenever a repeated substring $\alpha$ is discovered, a new non-terminal can be introduced into the non-terminal vocabulary of the induced grammar, and the recursive rule

$$N \rightarrow \alpha N$$

can be added to the induced grammar. Each occurrence of $\alpha$ in the sample is then replaced with the non-terminal "$N$". The derivation tree for this is as follows:

```
...cabcabcabcd...
```

The recoded sample is then

```
...cNNNd....
```

There is no good procedure for deciding whether to write the rule as $N \rightarrow abcN$ or $N \rightarrow cabN$, when a terminal such as "c" appears at the beginning of the repeated substrings. There is no good criterion for deciding whether to make the recursive rule left recursive or right recursive. There is no good way to decide whether the sequence immediately following the repeated substring should be written as a disjunctive alternative to the recursive symbol—that is,

$$N \rightarrow \text{abcN} \mid d$$

or whether another formulation is more desirable.

Moreover, an obvious self-embedding such as $N \rightarrow \text{abcNdef}$ with the derivation tree

```
...cabcabcabcdefdefdef....
```
may yield the pair of recursive rules

   N \rightarrow abcN

   M \rightarrow defM

and a recoded sample of

   ...cNNNMMM... 

Moreover, this method has no obvious extension to context-sensitive or unrestricted rewrite rules; and the method also requires a rather large sample size in general.
5. RULE-ORIENTED METHOD OF INDUCING RECURSION

In the rule-oriented method of inducing recursion, we examine the rules of production generated at each successive level of the grammar discovery process, and try to induce the presence of recursion.

Consider, for example, a language generated by the grammar with "p" and "+" as the terminals and with the single strictly-context-free recursive rule

\[ p \rightarrow +pp \]

A sample of sentences (with initial punctuation) from this language might be


In practice, the Grammar Discovery Algorithm would discover the single non-recursive rule

\[ A \rightarrow +pp \]

at level 1. The original sample would then be recoded using this rule of production. The result would be a set of sentences containing the original terminal symbols "p" and "+", and also the non-terminal "A". The substring "+pp" will not, of course, appear in the encoded strings. The result would be


Note that substrings such as "+pA", "+Ap", as well as "+AA" do appear in the image.

The rules of production developed at level 2 include

\[ B \rightarrow +AA, \]

which is the obvious non-recursive rule resulting from the recursive structure of the language. However, rules such as

\[ C \rightarrow +pA \]

and
D \rightarrow +Ap

would also be developed since "+pA" and "+Ap" are cohesive sub-strings at this level. Note the resulting exponential growth of the number of rules of production, and of the non-terminals. This growth is typical when the more economical recursive rule (in this case $p \rightarrow +pp$) is not developed.

We now introduce some terminology.

Recursive rules of production for context-free grammars takes the following forms, where $A$ is a non-terminal, and where $\alpha$ and $\beta$ are Non-null strings:

first, $A \rightarrow A\alpha$, which is called a \textit{left recursion};

second, $A \rightarrow \alpha A$, which is called a \textit{right recursion};

and finally $A \rightarrow \alpha A\beta$, which is called a \textit{self-embedding} ("middle" recursion).

The first two rules are \textit{left regular} and \textit{right regular} rules, respectively, when the length of $\alpha$ is only 1.

If $M$ is large enough (that is, the length of the consequent (right) side of the rule of production is large enough), then recursions will manifest themselves between adjacent levels of the grammar discovery process. Suppose the rule of production

$$A \rightarrow \alpha B \phi$$

appears in the induced grammar at level $i$; and suppose the rule

$$B \rightarrow \delta C \epsilon$$

appears at level $i+1$; and suppose $A$ and $C$ are ultimately going to be identified with one another to make the recursive rule

$$A \rightarrow \alpha \delta A \phi \epsilon$$

then if $M$ had been large enough to encompass the length of $\alpha \delta A \phi \epsilon$,
the rule

\[ A \rightarrow a\delta B\phi \epsilon \]

could have appeared at level \( i \), so that the rule

\[ A \rightarrow a\delta A\phi \epsilon \]

would have been induced at that level.

Thus, it is sufficient to examine rules of production occurring at adjacent levels of the induced grammar, if \( M \) is generous enough. (Alternatively, if \( M \) is not generous, it will be necessary to expand our examination of rules to non-adjacent levels).\(^*\)

We now develop a procedure for identifying recursions.

Let \( R_1 \) be a rule of production induced at level \( i-1 \) of the grammar discovery process, and let \( R_2 \) be a proposed new rule of production that is about to be added to the induced grammar at level \( i \). Before any proposed new rule is added at level \( i \) (i.e., \( i \geq 2 \)), that rule is considered in relation to each rule already in the grammar from level \( i-1 \) to see if one recursive rule could replace both. Note that it is here that we limit the detection of recursion to adjacent levels.

The following are the conditions for inducing a recursion:

1. Rules \( R_1 \) and \( R_2 \) must be isomorphic in the following sense:
   a. The precondition for isomorphism is that the consequent (right) side of rule \( R_1 \) and the consequent (right) side of rule \( R_2 \) be of the same length.
   b. Secondly, there must exist a one-to-one mapping \( \xi \) from the set of symbols appearing in rule \( R_1 \) to the set

\(^*\)This latter feature is not now implemented in the computer program for the Grammar Discovery Algorithm.
of symbols appearing in rule $R_2$, such that if symbol $f$ appears in the consequent (right) side of rule $R_1$ at position $t$, then symbol $\xi(f)$ must appear in the consequent side of rule $R_2$ at position $t$, for all $t$ between 1 and the common length of the consequent sides of the two rules.

(2) There must exist a symbol $\sigma$ such that the following two conditions hold:

(a) The symbol $\sigma$ is a predicted symbol on the antecedent (left) side of rule $R_1$. (To determine this, the mask from which rule $R_1$ was developed must be considered).

(b) The symbol $\sigma$ appears on the consequent (right) side of rule $R_2$.

The symbol $\sigma$ is the recursive symbol—the symbol linking the two rules. If no recursive rule were identified, there would instead be a chain of rules at successive levels—each linked between adjacent levels by one such symbol.

If the above conditions hold, we have a recursion. The symbol $\sigma$ is now an extraneous symbol in the induced grammar. Let $T$ be the non-terminal symbol appearing in the same position of rule $R_1$ at level $i-1$ as $T$ occurs in rule $R_2$ at level $i$. The rule $R_2$ is extraneous, and should not be added to the induced grammar. The symbol $\sigma$ is extraneous and should now be deleted from the current non-terminal vocabulary of the induced grammar. Rule $R_1$ should be made into a recursive rule $R_1'$ by replacing the symbol $\sigma$ on its antecedent (left) side by the symbol $T$, so that $T$ occurs now on both the antecedent (left) side and the consequent (right) side of this modified rule $R_1'$. Thus, this modified rule $R_1'$ is now a recursive rule of production.
Moreover, $T$ should replace every occurrence of $\sigma$ in all existing rules as well. Rules which are then identical should then be deleted from the induced grammar. This Rewriting process simplifies the induced grammar.

Finally, the recursive rule just induced should be recursively applied wherever possible to the existing set of sentences. There will, in general, be many opportunities to apply the recursive rule which did not exist before. This back-tracking process has the effect of applying each rule maximally before going on.

Because of these processes of Rewriting all existing rules, and the back-tracking process of recursively applying any new recursive rule wherever possible, it is simpler from the point of view of writing the computer program for this algorithm to induce only one recursive rule at each level. The addition of additional levels to the grammar discovery algorithm in no way complicates the induced grammar.

Thus, in the example, the rule

$$A \rightarrow +pp$$

was a rule at level 1; and (if no search for recursion is made) the rules

$$B \rightarrow +AA$$

$$C \rightarrow +pA$$

$$D \rightarrow +Ap$$

were rules induced at level 2. Level 2 is the first level at which the inter-level search for recursion can be applied. Before accepting
a rule such as

\[ B \rightarrow +AA \]

we would discover that it is isomorphic to the rule \( A \rightarrow +pp \). That is, the consequent (right) side are both of length 3, and there exists a mapping \( \xi \) taking "+" into "+", and "p" into "A" and this mapping is one-one. The symbol "A" is the recursive symbol \( \sigma \). \( T \) is "p".

Replacing "A" with "p", we get the recursive rule

\[ p \rightarrow +pp \]

If we now apply this rule whenever possible to the original sample (1), we get a new sample which has a "p" wherever there is an "A" in (2). However, by applying this rule in every possible way, the sample (1) in fact is completely reduced to


It should be noted that every recursive rule of production is inherently a disjunctive rule as well. In particular, any recursive rule can be written in the form

\[ \alpha NB \rightarrow \delta N \delta | \phi \]

where \( \phi \) represents the empty string, where \( \alpha, B, \phi \) and \( \delta \) are strings, and where "N" is a non-terminal (recursive) symbol. The recursive rules induced above are all of this form. Thus, the rule induced in the above example is, if written completely,

\[ p \rightarrow +pp | \phi . \]
F. INDUCING DISJUNCTIONS AND GENERALIZATIONS

1. INTRODUCTION AND MOTIVATION FOR INDUCING GENERALIZATIONS

One specific application of the Grammar Discovery Algorithm is the induction problem for formal systems. In general, the rules and meta-rules of formal systems are written in terms of quantified variables. For example, a given property, say commutativity of some binary operation "\cdot", may hold "for all" variables taken from a certain set \( Z = \{ z_1, \ldots, z_k \} \). The commutative rule would not be stated as \( k^2 \) separate commutative rules (i.e. \( z_1 \cdot z_2 = z_2 \cdot z_1, z_1 \cdot z_3 = z_3 \cdot z_1, z_2 \cdot z_3 = z_3 \cdot z_2 \), etc.), but rather as one rule stated in terms of a meta-variable which ranges over the set \( Z \) (i.e. \( \forall x_1, x_2 \in Z \quad x_1 x_2 = x_2 x_1 \)). Of course, whenever the set \( Z \) is not finite, the set of meta-variables is not merely a convenience and economy, but a necessity.

The process of inducing the required meta-rule (which is stated in terms of a quantified meta-variables) from specific instances of the rule is called \textit{generalization}. Generalization in induction is the counterpart of substitution in derivation.

In doing induction in formal systems, it is desireable to have a facility for generalization.
2. INDUCING GENERALIZATIONS COMBINATORIALLY (RULE-ORIENTED METHOD)

The induction of generalized rules can be done combinatorially in a manner similar in concept to the induction of recursions. We begin by allowing the Grammar Discovery Algorithm to proceed in its development of rules level by level. But before a new rule of production \( R \) is admitted into the induced grammar (at a level greater than level one), the possibility of instead introducing a generalized rule is considered. The generalized rule, as it exists, would replace the rule \( R \) about to be admitted as well as certain rule(s) already in the induced grammar.

The following are the conditions required to induce a generalization:

1. First, there must be a non-empty set of rules \( R_1, \ldots, R_k \) in the induced grammar which are all isomorphic to the rule \( R \). (Isomorphism of rules is defined during the discussion of inducing recursion).

2. Second, for the \( h \) context positions of the consequent (right) side of rule \( R \), suppose only a particular subset of symbols (say, \( Z_1, Z_2, \ldots, Z_d \)) appear in those \( h \) positions in both \( R \) and \( R_1, \ldots, R_k \). Then all \( d^h \) combinations of these \( d \) symbols in those \( h \) positions must occur. (Note, then, that a minimum requirement is \( k \geq d^h \)).

3. Third, for all but the \( h \) positions, the rules \( R_1, \ldots, R_k \) and \( R \) must be identical.

Now define \( Z \) as the set of \( d \) symbols above. \( Z = \{Z_1, Z_2, \ldots, Z_d\} \). Let \( x_1, \ldots, x_d \) be \( d \) meta-variables. Let \( R^* \) be the rule obtained from \( R \) by substituting \( x_i \) for each occurrence of \( Z_i \) in rule \( R \) \((1 \leq i \leq d)\). Now the rule \( R^* \) replaces \( R_1, \ldots, R_k \) and \( R \) in the induced grammar. Rule
R* is the generalized rule; it is written in terms of the d meta-variables $x_1, \ldots, x_d$. For example, suppose a sample from a formal system contains rules of inference such as

$$
Z_1Z_2 \rightarrow Z_2Z_1 \\
Z_1Z_3 \rightarrow Z_3Z_1 \\
Z_2Z_3 \rightarrow Z_3Z_2 \\
Z_2Z_1 \rightarrow Z_1Z_2 \\
Z_3Z_1 \rightarrow Z_1Z_3 \\
Z_3Z_2 \rightarrow Z_2Z_3 \\
Z_1Z_1 \rightarrow Z_1Z_1 \quad \text{(identity rules)} \\
Z_2Z_2 \rightarrow Z_2Z_2 \quad ("""
Z_3Z_3 \rightarrow Z_3Z_3 \quad ("""

where

$$Z = \{Z_1, Z_2, Z_3\}$$

$$d = 3$$

$$h = 2$$

the $d^h = 9$ rules of inference are isomorphic. The $d^h$ rules of inference can be consolidated into one meta-rule expressed in terms of the meta-variables $x_1$ and $x_2$ universally quantified over $Z$ — namely, $\forall x_1, x_2 \in Z \quad x_1x_2 \rightarrow x_2x_1$.

The above induction of generalized meta-rules is a combinatorial process. Of course, the requirement for the appearance of all $d^k$ variations can be relaxed in practice—with the attendant risk of over-generalizing.*

Finally, it should be noted that the Generalization Process includes a process which might be called the process of finding "negative" regu-

*This combinatorial generalization process is not implemented in the computer program.
larities. Suppose, for example, in a non-binary alphabet that a given symbol of the alphabet does not appear in a particular context in the sample. Then a "negative" regularity can be envisioned which states that fact about the sample. In fact, one can envision the masks in the Search Phase as containing "negative" predicted positions—that is, which record the absence of particular symbols in particular contexts in the sample. However, it will be seen that the Generalization Process described above subsumes this process—and indeed, is more general in the sense that pairs, triplets, etc. of absent symbols in particular contexts can be expressed easily.

Let us now digress briefly and discuss the process of inducing disjunctive rules.
3. INTRODUCTION AND MOTIVATION FOR INDUCING DISJUNCTIONS

An explicit disjunctive rule is a rule of production in which the consequent (right) side consists of two or more different strings, any one of which can be substituted for the antecedent (left) side. A typical explicit disjunctive rule might be

\[ A \rightarrow aB \mid bB \mid c \]

where "A" and "B" are non-terminal symbols, and "a", "b", and "c" are terminal symbols. An implicit disjunctive rule is said to exist in a grammar whenever the antecedent (left) sides of two or more rules of production are identical, but the respective consequent (right) sides are not. Obviously implicit disjunctive rules can be collected and written as one explicit disjunctive rules.

Disjunctions already arise in the Grammar Discovery Algorithm—but only in the course of inducing recursions. Each recursive rule of production inherently is a disjunctive rule. In particular, any recursive rule can be written in the form

\[ \alpha \beta \gamma \delta \rightarrow \gamma \delta \phi \]

where \( \phi \) represents the empty string, where \( \alpha, \beta, \gamma, \) and \( \delta \) are strings, and where "N" is a non-terminal symbol. The recursive rules induced by the Grammar Discovery Algorithm are all in this form.

But so far, there is no other facility in the Grammar Discovery Algorithm for inducing disjunctions. Indeed, the bottom-up character of the Algorithm guarantees that no implicit disjunctive rule (and therefore no explicit disjunctive rule) can result because different substrings in the sample at any level will never be encoded in the same way. A facility for inducing disjunctions is particularly necessary when the sample has initial punctuation. In that event, the encoding can never proceed beyond a string such as
$S_1 S_2 S_3$. (and so forth),
where the $S_i$ are non-terminal symbols. A facility for inducing disjunctions in any level of the process is needed in order to avoid always writing final rules such as

$$S \rightarrow S_1 \mid S_2 \mid S_3 \mid \ldots$$

where "S" is the starting (non-terminal) symbol of the grammar.

This same unattractive situation (i.e. of having to write one final trivial disjunctive rule having a large set of disjunctive choices) is possible, of course, even in the absence of initial punctuation. In any case, this situation can be avoided if the Grammar Discovery Algorithm has a facility to induce disjunctive (non-deterministic) rules of production at any level (that is, not merely at the final level).

Finally, in the absence of a facility for inducing disjunctions (i.e. if all rules were deterministic), only one possible structure (except for variations caused by the disjunctions inherent in recursions) can be represented by the grammar. Thus, while recursions provide the means for generating infinities of sentences, disjunctions provide the means for generating different varieties of sentence.
4. INDUCING DISJUNCTIONS USING ENTROPY (SENTENCE-ORIENTED METHOD)

Up to now, only non-disjunctive (deterministic) rules of production have been induced. These rules of production have been induced as the result of the existence of highly reliable regularities in the sample. For example, the fact that the symbol "a" appears in a position 1 of the sample, and a "e" appears in position 4 of the sample may imply, with 99% reliability, that symbols "bb" appear in positions 2 and 3. Perhaps "cc" appears in positions 2 and 3 with conditional probability .5%. In this situation, a rule of production

\[ aN_e \rightarrow abbe \]

may be induced, where N is a non-terminal symbol. This rule faithfully represents the sequence of symbols in the sample 99% of the time. Actually, three "regularities" are involved in the above recoding. First, there is the highly reliable Type II regularity

\[ R_1 = \langle "a\%e", "_bb_", .99, Y \rangle. \]

This regularity is the basis for the recoding. Then, there are the two Type V regularities (or "non-regularities", if you prefer)

\[ R_2 = \langle "a\%e", "_cc_", .005, Y \rangle \]

and

\[ R_3 = \langle "a\%e", "_dd_", .005, Y \rangle. \]

These two regularities are in effect ignored because they represent a relationship among the symbols of the sample which occurs only rarely. Note that the entropy of the ensemble of probabilities \(<.99, .005, .005>\) is very nearly zero. This near-zero entropy is associated with deterministic rules of production that are induced by the Algorithm.

Now suppose that, in a given sample Y, we have the following three regularities:

\[ R_1 = \langle "a\%e", "_bb_", .33, Y \rangle, \]
\[ \mathcal{R}_2 = \langle "a\%e", ",_cc_", .33, Y \rangle \]
and
\[ \mathcal{R}_3 = \langle "a\%e", ",_dd_", .33, Y \rangle \]

Now, the three possible substitution instances (namely, "bb", "cc", or "dd" in positions 2 and 3) occur with equal probability. Certainly, one could not reasonably induce a rule of production that favored any one of the three possibilities (e.g. aNe \rightarrow abbe). The reason is that the context here (i.e. an "a" in a first position, and a "e" in the fourth position) does not reliably predict the intervening 2 symbols. Indeed, of the three possibilities that occur, the three occur equally often. On the other hand, there are only three substitution instances—that is, the symbols "ae", "ab", etc. are not possibilities. Note that, in this situation the ensemble of probabilities is \(<.33, .33, .33\) here, and that the entropy of this ensemble is large (and indeed maximal, if only the three substitution instances that actually occur are considered).

The above suggests a criterion for inducing disjunctive rules, starting at the very first level of the Grammar Discovery Algorithm.

Let \( W \) be a set of \( w \) regularities \( \mathcal{R}_i = \langle \gamma, \phi_i, P_i, Y \rangle \)\(^{w}_{i=1} \)

having the same context-sub-sequence \( \gamma \) and having non-zero conditional probabilities \( \{ P_i \} \)^{W}_{i=1} . (In general, \( W << C^h \), where \( h \) is the number of predicted positions in the mask associated with the regularities).

Suppose the entropy of the ensemble of these \( w \) non-zero probabilities is maximal, or nearly maximal—that is,

\[ \log_2 w - P_i \sum_{i=1}^{w} \log_2 P_i \]

is zero, or small. In this situation, we write the disjunctive (non-deterministic) rules of production
\[ \alpha \rightarrow \Delta_1 | \Delta_2 | \ldots | \Delta_w \]

where \( \alpha \) is the (common) antecedent (left) side as may be obtained from any of the \( \mathcal{R}_1 \), and where \( \Delta_i \) is the \( \Delta \)-sequence for \( \mathcal{R}_1 \), \( 1 \leq i \leq w \).

Note that we induce deterministic rules of production when the reliability of a single regularity is high (the entropy over the possible substitution instances is zero, or small), and that we induce disjunctive (non-deterministic) rules of production when there are several equally-likely (or nearly equally-likely) substitution instances (i.e. the entropy over the possible substitution instances is maximal or nearly maximal). With intermediate entropy, we do no encoding, and we defer resolution to a higher level. Note also that with either approach, the goal of minimizing the number of rules of production and maximizing the parsimony of the induced grammar is furthered, as is the goal of maximizing the non-trivial and hierarchical quality of the grammar.

In certain samples from formal systems, there is an equivalence between generalized rules and disjunctions. Suppose string \( \alpha, B, \) and \( \gamma \) can be disjunctively substituted for a non-terminal \( N \) appearing in some string \( \delta \) (using the rule \( N \rightarrow \alpha | B | \gamma \)). The string \( \delta \) may be written as \( \delta = \delta_1 \mathcal{N} \delta_2 \), where \( \delta_1 \) and \( \delta_2 \) are strings. Define \( Z \) as the set containing \( \alpha, B, \) and \( \gamma, \) and then define \( z \) as a metavariable ranging over \( Z \). If it is appropriate to interpret the sample as containing theorems of a formal system, we can then rewrite the disjunctive rule

\[ N \rightarrow \alpha | B | \gamma \]

as the universally quantified generalization

\[ \forall z \in Z \exists \delta_1 \exists \delta_2 \]

for that formal system.

Conversely, if we have a universally quantified generalization
\[ \forall z \in Z \quad F(z) \]

where \( F(z) \) is a rule in which \( z \) appears, then we can write

\[ N \to z_1 \mid z_2 \mid \ldots \mid z_n, \]

where \( Z = \{z_1, z_2, \ldots, z_n\} \).

As mentioned earlier, implicit disjunctive rules in a grammar can be gathered together and written as one explicit disjunctive rule. We can therefore define the multiplicity of an explicit disjunction as the number of different disjunctive alternatives stated in the rule.

Rules of production are then of three types: (1) explicit disjunctive rules, (2) recursions, and (3) non-disjunctive, non-recursive rules. If a grammar consists only of the third type of rules, at most one sentence can be generated. It is necessary to have disjunctions to attain a variety of sentence structures. If the grammar consists only of rules of the first and third type, there are no more sentences than the product of the multiplicities of the explicit disjunctions. This number is, of course, finite. This product is an upper limit on the number of essentially different structural types for sentences generated by the grammar (of course, through ambiguity and other reasons, this upper limit may not be attained). Indeed, variety in structural types arises only from disjunctions.

If there are any recursions in the grammar, the number of sentences that may be generated by the grammar becomes infinite (assuming the rule can be used).
Note, however, that the number of essentially different structural types of sentences is still bounded. Recall also that it was noted earlier that every recursive rule is inherently a binary disjunctive rule---one of the disjunctive possibilities being the recursive part of the rule, and the other being the so-called "base" string. Note also that the recursive rule can always be written so that this base string is the empty string.

A normal-form grammar here is a grammar such that all of the rules of production are rules of one of the following types:

(1) explicit disjunctive rules,

(2) recursions with an empty base string, and

(3) non-disjunctive, non-recursive rules.

For every grammar, there is a normal-form grammar that generates exactly the same language---that is, there is an equivalent normal-form grammar.

In general, for a normal-form grammar, the number of essentially different sentence structures is bounded above by the product of multiplicities of the explicit disjunctive rules and \( 2^{n_r} \), where \( n_r \) is the number of recursive rules.

Note, from the description of the Grammar Discovery Algorithm, that every induced grammar produced by the Algorithm is a normal-form grammar.
G. TERMINATION PROCEDURE

The Grammar Discovery Algorithm terminates when the sample \( Y \) of the current level consists only of one symbol. This symbol becomes the starting symbol of the induced grammar. The Grammar Discovery Algorithm arrives at this situation in one of two ways. If the Grammar Discovery Algorithm is operating in the first mode (i.e. there is no initial punctuation between sentences), then this situation is arrived at naturally by the reduction of \( Y \) to a length of one.

If the Algorithm is operating in the second mode (i.e. there is initial punctuation between the sentences of the sample), then the sample at the next-to-last level may have consisted of a concatenation of single non-terminal sentences and initial punctuation marks (i.e. \( s_1.s_2. \) and so forth). In that case, the reduction of the sample from this level to the final level may be accomplished by (a) removing repetitions of the same non-terminal from the sample, and (b) Inducing Disjunctions over the single symbols. Typically this disjunction occurs at an intermediate level, and process (a) then applies to the repetitions at the final level. Given a finite sample \( Y \), the Grammar Discovery Algorithm converges to some induced grammar after a finite amount of time.

If \( M \geq 2 \) (as is always the case) and if context-free rules only are being generated (as described in a later section), then each application of each context-free rule reduced the length of the sample by at least one symbol. Even if each transformation involves only one application of one binary constituent rule (\( M \) of 2), the Algorithm would terminate after either \( N-1 \) levels (\( N \) being the length of the sample, if there is no initial punctuation in the sample) or \( N_{\text{max}}-1 \) levels (\( N_{\text{max}} \) being the length of the longest sentence in the sample, if there is initial punctuation).
If context-sentitive rules are being generated, the reduction in the length of the sample \( Y \) from level to level depends on their being more than one contiguous predicted symbol in the mask of at least one regularity used at least once in the recoding. One can assure this by considering only masks with this property, or by requiring that there be at least one application of such a rule as part of each transformation (or, of course, by having at least one context-free rule in each transformation).

In practice, of course, one usually "covers" the entire sample \( Y \) or nearly all of it) so that these requirements are virtually automatically satisfied by any transformation.
H. CHOICE OF LIMITS ON LENGTH OF REGULARITIES IN THE SEARCH PHASE

In the Search Phase of the Grammar Discovery Algorithm, masks of length \( M = M_1, \ldots, M_2 \) are considered. The lower limit \( M_1 \) and the upper limit \( M_2^* \) on the length of mask to be considered are determined from the following considerations:

(1) Clearly one cannot scan the sample \( Y \) with an \( M \) greater than the length \( N_{\text{max}} \) of the longest sentence in \( Y \). If the Grammar Discovery Algorithm is operating in the first mode (i.e. with no initial punctuation in the sample), then \( M_2 \) cannot be larger than \( N \)—the sample length. If the Grammar Discovery Algorithm is operating in the second mode (i.e. with initial punctuation), \( M_2 \) is limited above by \( N_{\text{max}} \)—the length of the longest individual sentence in the sample \( Y \).

(2) An \( M \) of 1 is appropriate only in the case where each symbol in the sequence \( Y \) is statistically independent from the others, or in the case where \( Y \) consists entirely of 1 symbol repeated endlessly. In the case of one symbol repeated endlessly, an \( H \) (information rate) of zero is attained. In the case of statistical independence among the single (different) symbols, an information rate greater than zero is attained; and indeed \( H \) is maximal (given the alphabet size) when the symbols occur equally often. These two cases are so uninteresting that we specify \( M_1 \geq 2 \) to avoid them.

(3) If \( M \) is large, the number of different sequences of length \( M \) over \( C \) symbols is large. Thus, the number of appearances of any particular sequence in the sample is relatively small. Certainly, if a sequence only occurs once, twice, or 3 times, the statistics about these sequences will not be meaningful. Thus, an \( M \) should be tried which is not too large relative to the total length \( N \) of the sample. Also,

\(^*\)\( M_1 \) and \( M_2 \) are parameters in the computer program.
if $\sum N^M$, it is not even possible for all possible sequences of length $M$ to appear, even if, in these $N$ symbols, no sequences appeared more than once. Thus, $M$ should be chosen so that

$$\log_c N \gg M.$$  

(4) In attempting to induce recursions, the point was made that if there is a recursion, it will manifest itself in the tentative rules induced at consecutive levels, provided $M$ is large enough. Thus, either $M$ should be large enough to encompass any recursion or the search for recursions should extend between non-consecutive levels. This could be done in the same fashion as the search for recursions between consecutive levels.

It should always be remembered that the entire approach to grammar discovery described here is based on the idea of using relatively small $M$ in searching for regularities. If a regularity is missed at one level because $M$ was too small, then we claim the regularity will be detected at a higher level. Thus, if the maximum length of $M$ is limited to say 5, and a Type I (100% reliable) regularity of length 10 exists in the sample, that regularity will be detected at a higher level of the process. In the simplest case the first 5 symbols may be encoded as a second new symbol. If the first 5 symbols at the first level reliably predict the five succeeding symbols, then this regularity involving 10 symbols at the first level manifests itself as a regularity involving two symbols at the second level—namely, the first new symbol reliably predicts the second new symbol as its immediate successor.

It should be emphasized that finding two regularities of length 5 at the first level of the process, and then finding one regularity of length 2 at the second level is not at all the same as finding the one regularity of length 10 at the first level. This point is made in
Goodall (1962). Combinatorics aside, the important difference is that the first approach discovered a hierarchical relation in the sample, and offers a significant insight into the sub-structure of the phrase of length 10, while the other approach merely cataloged the occurrence of a gross event.

The difference is between

\[
\begin{align*}
\text{A} &
\quad
\begin{align*}
\text{B} &
\quad
\begin{align*}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10
\end{align*}
\end{align*}
\text{C}
\end{align*}
\]

and

\[
\begin{align*}
\text{A} &
\quad
\begin{align*}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{align*}
\end{align*}
\]

Indeed, if M is large enough, the subsequences occurring in any finite sample become quite regular and unique—the limiting case of this being a completely trivial, non-hierarchical, and non-parsimonious grammar.
I. CHOICE OF PARAMETERS IN THE RECODING PROCEDURE

The Recoding Procedure described earlier is only one possible recoding procedure. Several variations in this Recoding Procedure are possible. For example, the sample Y may be scanned from right to left, instead of from left to right, in the search for opportunities for developing rules of production. Or, the scan could be started at some intermediate symbol. Or, the positions could be scanned in some order determined by some other considerations—such as the value of the conditional probabilities encountered, etc.

Similarly, the regularities could be considered in ascending, rather than descending, order of length*—thereby giving preference to longer regularities.

Also, the position for starting the scan is variable.** If the length of the regularity varies between M1 and M2, and the scan of Y is started at position T< M2, then obviously a regularity of length less than or equal to T may well be found to recode some or all of these first T positions, to the exclusion of a regularity of length M2.

The Recoding Procedure can also operate so that whenever recoding is attempted using phrases no longer than M that this recoding is actually done using only phrases of length M. This is called a strict recoding***, and is appropriate only when the sample is believed to come from a "uniform" code source (Fano). This variation in the Recoding Procedure would be only rarely appropriate.

Another variation in the Recoding Procedure concerns the allowed range in values of the conditional probabilities of the regularities

---

*This variation is under the control of the variable VDIR in the computer program implemented in the Grammar Discovery Algorithm.

**This variation is under the control of the variable VSTART.

***This variation is under the control of the variable STRICT.
used in recoding. The conditional probabilities associated with each regularity are categorized by type (i.e. Type I, Type II, etc.), and must specify whether the recoding procedure can use only absolutely reliable (Type I) regularities, or whether Type II, or Type III regularities can also be used.****

Finally, Recoding Procedures can differ according to the order in which the various criteria for selecting regularities are applied. For example, one can consider all the Type I regularities which are also of length M, and then the other Type I regularities of different length. Alternately, one can consider all the regularities of length M which are also of Type I, and then consider other regularities of length M which are of different type.

The impact of these variations will be discussed in a later section.

Finally, it should be noted that the Recoding Procedure, even with the variations noted above, is only an abbreviation itself of a more combinatorially complete recoding procedure—that is, a recoding procedure which is exhaustive in the sense that it includes all possible variations in recoding. This complete recoding procedure would not scan the sample from left to right, or right to left, or from the middle out. In this complete procedure, the procedure would start by considering all possible ways of partitioning the given sample Y of length N into parts, with no part containing more than M symbols. The number of such partitions is the same as the number of partitions of the integer N into parts no greater than M (Riordan, 1958), and this number is very large. Then all the possible regularities that might be applied to each part of each partition must be considered. And for each partition, and each such part, each possible regularity

****This variation is under the control of the variable ALLOW.
must be substituted into the part, and a recoding attempted, and entropy, parsimony, and recursive parsimony of that encoding computed. And, when a recoding is finally completed at one level, all the possible different recodings at each successive level must be considered before that recoding is finally selected to be the recoding at the present level.
J. EXTENSIONS OF THE GRAMMAR DISCOVERY ALGORITHM TO SAMPLES OTHER THAN LINEAR SEQUENCES OF SYMBOLS

Earlier we made the statement that the methods of induction and grammar discovery that we described herein are not particular to the format for presentation of the sample. We then proceeded to describe a specific Grammar Discovery Algorithm dealing with the problem of inducing a grammar to generate a given sample of sentences which is presented as a linear sequence of symbols. In this section, we return to this point and argue for the general applicability of the methods of induction and Grammar Discovery Algorithm in terms of a two dimensional pattern recognition problem. It will be seen that this rephrasing is not particular either to the Grammar Discovery problem for sentences, or to the two dimensional pattern recognition problem, and that the main features and insights of the Grammar Discovery Algorithm apply to other problems, as well as similar problems presented in other formats.

Let us consider the simplest kind of two dimensional pattern recognition problem. The sample will consist of a two dimensional raster (matrix) of digital symbols. The raster may, for example, consist of binary digital symbols representing the digitalized image of certain items to be recognized—perhaps printed letters of the arabic alphabet. The alphabet for this sample is the set of whatever symbols appear in the raster—perhaps 0 and 1 in the case of a binary raster. A raster is said to have initial punctuation if certain submatrices in the raster are separated in some way—for example, if there were $11 \times 7$ sub-matrices outlined such that each submatrix is presumed to contain one letter of the arabic alphabet. In contrast, a raster would be unpunctuated if no such initial punctuation of the raster were specified. A mask is a submatrix over the context symbol "_", the predicted symbol
"%", and—if the mask is ternary—the don't care symbol '>'. $\delta$ and $\phi$ are submatrices defined in a manner analogous to the subsequences $\delta$ and $\phi$ of the Grammar Discovery problem. A regularity is defined exactly as for the Grammar Discovery problem except that $\delta$ is the context-sub-matrix and $\phi$ is the predicted-sub-matrix. In the Search Phase, various masks are considered, and regularities are catalogued and characterized by type according to their conditional probability.

In the recoding phase, the domain is partitioned ("covered") by sub-matrices. Whenever the context-sub-matrix of a regularity of allowably high type occurs in one of the parts of the domain, this part is recoded. Rules of production are developed from regularities to express this recoding in a way exactly analogous to the 1-dimensional case. For example, the letter "L" occurring as in a 12 x 12 raster amongst a large sample of Arabic letters might be recoded as shown below.

Here an 8 x 3 pillar of 1's is recognized as a feature of many letters, and recoded as "V" (vertical line); the 3 x 3 vertical-right intersection sub-matrix is recognized and recoded as "I"; and the 3 x 4 hori-
Zontal line might be recognized and recoded as "H". The 9 x 5 blank area might be recoded as "B". Thus, the letter "L" is recoded as a 2 x 2 sub-matrix containing a "V", "I", "H", and "B". Indeed, this description is precisely what an "L" is—namely a vertical line segment intersecting at its end with the end of a horizontal line segment. Note that both the vertical and horizontal segment may be represented by recursive rules. If L's occur often in the sample, this 2 x 2 sub-matrix might be given a name and recoded.

One should note in passing that this entire paper is about induction and pattern recognition of samples of discrete symbols. Suppose instead of doing a basic symbol-by-symbol matching operation, one does a correlation between time segments of continuous signals. It may be possible to extend the notion of regularity to continuous symbols. The time shift operation inherent in the matching of symbols has an immediate analog for continuous signals. Defining the context part, the predicted part, and the don't care part of a signal merely involves considering the portion of a given signal restricted to a particular time domain with the correlation coefficient playing the role of the conditional probability. The idea of a regularity amongst a sample of continuous signals may be defined in the obvious way.

The universe of possible masks is infinite, but it is possible to imagine some discrete time intervals being used to make this universe finite.

The concepts of entropy, parsimony, and recoding all have obvious analogs.

It should be noted that the operation of digitalizing a continuous signal is itself a recoding, or punctuating process, in exactly the sense we have been discussing throughout this paper. In digitalizing
data, a *fidelity criterion* (Chomsky & Miller) is used to associate almost identical continuous signal segments and to encode them under a common discrete digital name.
III. THE SELF-PUNCTUATING NATURE OF THE GRAMMAR DISCOVERY ALGORITHM

A. INTRODUCTION

The choices of the upper and lower limits on the length of regularities searched for in the Search Phase and the choice of parameters in the Recoding Procedure affect the partitioning (punctuating) of the sample into the short parts which are in turn recoded. Since the choice of parameters in the Recoding Procedure and in the values of M1 and M2 are specified externally according to heuristic considerations, the grammar that is ultimately induced would seem to be the product, not of the Grammar Discovery Algorithm, but rather of these external choices. These external choices would seem to lead to two principle types of effects: First, there would seem to be the effect of choosing M2 too small in the Search Phase and thereby "missing" longer regularities in the sample. Second, there would seem to be numerous effects of partitioning the sample for purposes of recoding—perhaps of missing regularities because the regularity encompasses symbols that end up in different parts of this partition—or, perhaps, of losing the "synchronization" of the symbols in the sample (as, for example, in a sample produced by a uniform code source).

It will be seen that both the choice of M2 and the choice of parameters in the Recoding Procedure ultimately concern the partitioning, or "punctuating", the sample into parts. In fact, the Grammar Discovery Algorithm is essentially a punctuating process (or a "selective punctuation" as Goodall would call it).

More importantly, the Grammar Discovery Algorithm is "self-punctuating," and recovers from the presumed predetermining effects of the choice of M2 and of synchronization.
B. EFFECT OF CHOICE OF LIMITS ON LENGTH OF REGULARITIES IN THE SEARCH PHASE

1. INITIAL DISCUSSION

Let us first consider the possible limiting and predetermining effects of the choice M2. (M1 is normally 2 and is not an issue). Suppose that the maximum length of regularity searched for in the Search Phase is M2, but that a regularity of length M, with M>M2, exists in the sample Y, say in positions $t_{h+1}, \ldots, t_{h+M}$. The regularity that might be missed might for example, be that symbol $Y(h+1)$ in position 1 of the regularity, and the symbol $Y(h+M)$ in position M together reliably determine all the intermediate symbols $Y(h+2), \ldots, Y(h+M-1)$. If M is, say 8, the regularity would be characterized by a context sub-sequence of "a%%%h" and a predicted sequence of "_bcdefg_" and a mask of "_%%_%" and a conditional probability of 100% (Type I - absolutely reliable). That is, the regularity is <"a%%%h", "_bcdefg_", 1.0>. If M2 is only 6, this regularity would appear to be missed. We claim that if a regularity is missed at one level because M2 was too small, then the regularity will be detected as a regularity at a higher level. We now proceed to make this statement more precise.
2. EXTENSIONS OF A REGULARITY

First, it is necessary to know that the regularity which may be "missed" is the longest regularity encompassing the symbols involved. This does not weaken the statement; it does not sacrifice generality; indeed, it would seem to make the statement harder to verify.

Recall that a regularity \( R = \langle \gamma, \phi, P, Y \rangle \) is characterized by its context-sub-sequence \( \gamma \) (a sequence of length \( M \) over the current alphabet \( V_C \) augmented by the predicted symbol "\( \% \)"; the predicted-sub-sequence \( \phi \) (which is a sequence of length \( M \) over the current alphabet \( V_C \) augmented by the context symbol "\( _\)"), and the value \( P \) (which expresses the conditional probability that the context-sub-sequence \( \gamma \) predicts the predicted-sub-sequence \( \phi \) in the sample \( Y \)).

A regularity \( R = \langle \gamma, \phi, P, Y \rangle \) may be \( p \)-extended to the left (right) to a new regularity \( R' = \langle \gamma, \phi, P, Y \rangle \) by the addition of a predicted position—that is, by increasing the length of \( \gamma \) and \( \phi \) by one, by annexing a "\( \% \)" to the context-sub-sequence \( \gamma \) on the left (right) thus forming \( \gamma' \), by annexing a specified symbol from the current alphabet \( V_C \) to the corresponding position of the predicted-sub-sequence \( \phi \) thus forming \( \phi' \), and by changing the conditional probability \( P \) to \( P' \) to express the conditional probability that \( \gamma' \) predicts \( \phi' \) in the sample \( Y \). Note that there are \( 2 \cdot C \) possible \( p \)-extensions to the left (or right) for a given regularity. Note also that \( P' \leq P \).

Similarly, a regularity \( R = \langle \gamma, \phi, P, Y \rangle \) may be \( \sigma \)-extended to the left (right) to a new regularity \( R' = \langle \gamma', \phi', P', Y \rangle \) by the addition of a context position—that is, by increasing the length of \( \gamma \) and \( \phi \) by one, by annexing a "\( _\)" to the predicted-sub-sequence \( \phi \)
on the left (right) thus forming $\phi'$, by annexing a specified symbol from the current alphabet $V_c$ to the corresponding position of the context-sub-sequence $\gamma$ thus forming $\gamma'$, and by changing the conditional probability $P$ to $P'$ to express the conditional probability that $\gamma'$ predicts $\phi'$ in the sample $Y$. Note that there are $2^C$ possible c-extensions to the left (or right) for a given regularity. Note also that $P' \geq P$.

Of course, the type of a regularity may change under extension—p-extensions being in general of lower type (less reliable) than the regularity from which they were derived, and c-extensions being in general of higher type (more reliable) than the regularity from which they were derived.

In our discussions, if it does not matter whether an extension is a c-extension or a p-extension, it will be called simply an extension.
3. DEFINITION OF A MAXIMAL REGULARITY

A regularity \( R = \langle \gamma, \phi, P, Y \rangle \) of types I, II, or III is said to be preserved (p-preserved) (c-preserved) under extention provided that \( P' \supseteq P \) — or the extention is of the same type as \( R \).

A regularity \( R \) is said to be maximal (p-maximal) (c-maximal) provided it is not preserved under extention (p-extentions) (c-extention) for any possible extentions (p-extentions) (c-extentions) of it.

The existence of a maximal regularity means that a particular sequence appears as a unit in the sample \( Y \) but that the symbols surrounding this unit vary widely. For example, if \( R = \langle 'ab%'', '_c''', 1.0, Y \rangle \) is a regularity, it means that whenever "ab" appears in \( Y \), the symbol "c" always follows. If \( R \) is a maximal regularity, it means that the sequence "abc" appears in the sample \( Y \) embedded in all variety of different environments, perhaps as "dabce", "fabcg", etc. If \( R' = \langle 'a%'', '_b''', 1.0, Y \rangle \) is also a regularity in \( Y \), it is not maximal because \( R' \) may be extended to \( R \). Equivalently, "ab" does not appear in a variety of different contexts in \( Y \), but rather also appears with a "c" to its right.
4. PERSISTENCE OF MAXIMAL REGULARITIES AFTER RECODINGS

We now claim that a maximal regularity of length $M$ will usually not be missed by the Grammar Discovery Algorithm because $M_2$ (where $M > M_2$) was chosen too small. If a maximal regularity exists at one level of the sample, it will usually persist and continue to exist after the recoding at the next higher level. Thus, an unfortuitously small choice of $M_2$ will not cause the maximal regularity to be lost. "usually" is our "almost always" and it means "in all cases except possible for a case whose occurrence requires the coincidence of several independent events each of very small probability "— in effect, "usually" refers to second order (and lower) effects.

Let us assume that a maximal regularity $\mathcal{R} = \langle Y, \phi, p, Y \rangle$ of specified type (Type I, II, or III) exists in the sample $Y$, and that the regularity is of length $M > M_2$. Suppose that the context sequence appears $q$ times in the sample $Y$. Let $\Delta$ be the domain sequence of $\mathcal{R}$ in $Y$. Say that $\Delta$ appears $q'$ times in $Y$. Note that $p = q'/q$.

Let $\Delta_j$ denote the $j$-th occurrence of the sequence $\Delta$ in $Y$ ($j = 1, \ldots, q$). Let $\Delta_j(i)$ denote the $i$-th symbol in the $j$-th occurrence of $\Delta$ in $Y$ where $i = 1, \ldots, M$, and $j = 1, \ldots, q$.

Let $a_j$ be the symbol appearing in $Y$ to the left of the left-most symbol of the $j$-th occurrence of $\Delta$ in $Y$ ($j = 1, \ldots, q$), and let $b_j$ be the symbol appearing in $Y$ to the right of the right-most symbol of the $j$-th occurrence of $\Delta$ in $Y$.

Note that because $\mathcal{R}$ is maximal, no symbols in the sequence $\langle a_1, \ldots, a_q \rangle$ or $\langle b_1, \ldots, b_q \rangle$ or in the sequence of ordered pairs $\langle a_1 b_1, \ldots, a_q b_q \rangle$ will appear more than $\frac{q}{2}$ times (since $\mathcal{R}$ is of type III or better).

There are apparently three cases to consider:
(1) None of the strings $a_j^\Delta b_j$ of length $q+2$ are encoded by the transformation operating at this level of the Grammar Discovery Algorithm.

(2) some of the symbols in $\Delta j$ are encoded by a transformational rule, but neither the symbol $a_j$ nor the symbol $b_j$ is also encoded by the same application of this rule—that is, only symbols interior to $\Delta j$ are encoded.

(3) Some symbols of $\Delta j$ are encoded by a transformational rule, and either the symbol $a_j$ or the symbol $b_j$ is also encoded by the same application of this rule. (Note, that both $a_j$ and $b_j$ cannot be encoded by one application of one rule, since $M=M2$).

Case (1) poses no problem as to "missing" regularities, since no encoding occurs. At a higher level some encoding may occur, but even then one of the cases below will then apply. It should be noted, however, that the Grammar Discovery Algorithm operates in practice so that most symbols of $Y$ do in fact get "covered" by the rules at each level. When we say that the symbols are "covered", we do not necessarily mean that they are changed, but that they will at least be part of a context-sub-sequence. In summary, this case does not occur often, and is no problem when it does occur.

In case (2), some symbols entirely interior to $\Delta j$ are encoded.

Suppose that the $k$ symbols $\Delta j(h), \ldots, \Delta j(h+k-1)$, $k \ll M$. are the symbols interior to $\Delta j$ which are encoded—that is, are the domain of the transformation $\mathcal{T}$ being applied to the sample $Y$. Note that this transformation $\mathcal{T}$ is made only because there exists some regularity $R' = \langle \gamma', \phi', P', Y \rangle$ of length $k$ of allowably* high type in the sample. As a matter of notation, note that the $\Delta$ of $R'$

*The variable ALLOW controls this in the computer program.
is \( \Delta_j(h), \ldots, \Delta_j(h+k-1) \), and we call this substring \( \Delta' \). Note also that by \( \mathcal{T}(\Delta)(I) \), we mean the I-th symbol in the image under \( \mathcal{T} \) of \( \Delta \).

In a transformation, contiguous symbols of the domain \( \Delta' \) are encoded as new non-terminal symbols. If the image of \( \Delta_j \), that is \( \mathcal{T}(\Delta_j) \), is of the same length or longer than \( \Delta \) (the latter occurring only when Unrestricted Rewrite Rules are being generated), no regularity \( \mathcal{R} \) is "missed" because at some higher level the reduction in length finally occurs, and one of the cases here will then apply. If \( \mathcal{T}(\Delta_j) \) is shorter than \( \Delta_j \), then \( \mathcal{T}(\Delta') \) was shorter than \( \Delta' \). Say the length of \( \mathcal{T}(\Delta') \) is \( f \), where \( f < k \). Then the image of \( \Delta_j \) is the substring

\[
\mathcal{T}(\Delta_j) = \Delta_j(1), \ldots, \Delta_j(h-1), \mathcal{T}(\Delta')(1), \ldots, \mathcal{T}(\Delta')(f), \Delta_j(h+k), \ldots, \Delta_j(M)
\]

which is of length \( M-(k-f) \).

There are now two sub-cases. In the first sub-case, \( M-(k-f) \) is still greater than \( M_2 \). In this case, the regularity \( \mathcal{R} \) is not "missed," because the second sub-case below will apply at a higher level when the necessary reduction in length finally occurs.

The second sub-case is that \( M-(k-f) \leq M_2 \). The original maximal regularity \( \mathcal{R} \) of length \( M \) now manifests itself as a regularity \( \mathcal{R}'' = \langle \gamma'', \phi'', p'', \gamma \rangle \). We show that \( \mathcal{R}'' \) exists by constructing it.

First we extend the transformation \( \mathcal{T} \) (which is a semigroup homomorphism) to the domain \( V_c \cup \{ "", "\%" \} \), by saying that the extension \( \mathcal{T}^* \) is

\[
\mathcal{T}^*(x) = \begin{cases} 
(x) & \text{if } x \in V_c \\
 x & \text{if } x \in \{ "", "\%" \}
\end{cases}
\]

Now we have, for \( I=1, \ldots, M-(k-f) \), the context-sub-sequence
\[
\gamma''(I) = \begin{cases} 
\gamma(I) & \text{if } 1 \leq I \leq h-1 \\
\mathcal{F}^*(\gamma')(I+h+1) & \text{if } h \leq I \leq h+f-1 \\
\gamma(I+(k-1)) & \text{if } h+f \leq I \leq M-(k-f)
\end{cases}
\]

the predicted sub-sequence

\[
\phi''(I) = \begin{cases} 
\phi(I) & \text{if } 1 \leq I \leq h-1 \\
\mathcal{F}^*(\phi')(I+h+1) & \text{if } h \leq I \leq h+f-1 \\
\phi(I+(k-f)) & \text{if } h+f \leq I \leq M-(k-f)
\end{cases}
\]

and the conditional probability

\[p''=p \cdot p'.\]

Note that if the types of regularities allowed for making the transformation \(\mathcal{F}\) are only of Type I (100% reliable), then \(R''\) will not only be detected—but its unconditional probability \(p''\) will be the same as that of \(R\) (since \(p' = 1.0\)). If the allowed types of regularity used in the recoding are of type II, then \(R''\) will either still be of type II or usually at least of Type III (since it would, in general, take a fairly large number \(k\) of recodings to make

\[(1-\varepsilon)^k < \frac{1}{2}\]

where \(\varepsilon\) was defined earlier to be \(1/c^k\) which is very small. Note that this \(k\) is, in general, only 1, and will be more than 1 only if case (1) applies at some point, or if, in case (2), \(\mathcal{F}(\Delta j)\) is the same length or longer than \(\Delta j\) for some recodings.

If type III regularities are used in the recoding (particularly, if the conditional probability of \(R'\) is \(\sqrt{\frac{1}{2}}\)), then regularities of type III (particularly, if the conditional probability of \(R\) is \(\sqrt{\frac{1}{2}}\)), may be lost. This is of course the reasons why regularities of Type III (particularly, the lower range of Type III are not very suitable for recodings).

It should be noted that this discussion of case (2) assumed one
rule of production (derived from one regularity \( \mathcal{R}' \)) was used to encode the symbols \( \Delta_j \). In general, more than one rule of production could be invoked. However, the argument is not altered—although the probability \( p'' \) would then be the sum of say, two small probabilities. It should be emphasized that because \( M_2 \) is always small, it would be most unlikely that as many as three rules of production would be applied within one such domain.

Case (3) really never happens. By hypothesis, \( \mathcal{R} \) is maximal. Hence there is no regularity of type III or of higher type that encompasses \( a_j \) (or \( b_j \)) and some of the symbols of \( \Delta_j \)—for if there were, there would then have to be a regularity of length \( M+1 \) encompassing the sequence \( a_j \Delta_j \) (or \( \Delta_j b_j \)) of length \( M+1 \)—and this is impossible since the fact that \( \mathcal{R} \) is maximal means that no such regularity exists.

We return now to the example cited earlier—the question of whether the regularity \( \mathcal{R} = <"a\\%\\%\\%\\%\\%\\%h","bcdefg","1.0> \) of length \( M = 8 \) would be missed if the maximum length of regularity searched for in the Search Phase was \( M_2 = 6 \). This regularity states a relationship among 8 symbols—and, in particular, that two widely separated symbols determine the intermediate symbols. We will assume that this regularity \( \mathcal{R} \) is maximal—that is, that the string "abcdefg" appears in a wide variety of different contexts in the sample. With a \( M_2 \) of 6, the Grammar Discovery Algorithm will be recoding the sample using regularities up to length 6. We assume that some recoding "abcdefg" does occur at this level. If some recoding does occur, it is the result of the existence of some regularity \( \mathcal{R}' \) within the 8 symbols—perhaps that "bc" are always followed by "defg". This recoding would be accomplished by the rule of production
\[ bcN \rightarrow bcdefg \]

where \( N \) is a non-terminal. Thus, the 8 symbols at the first level would be encoded as "abcNh" in every case. Now, at the next level, it would be noted that an "a" appearing in position 1 and an "h" appearing in position 5 reliably predicts the occurrence of "bcN" as intermediate symbols. This regularity would be noted because its length is 5, which is less than \( M_2 \), which is 6. Thus, at level 2, this regularity \( \mathcal{R} =\"a\%\%\%h\", _bcN_, 1.0 \) can be used to develop the rule

\[ aPh \rightarrow abcNh \]

where \( P \) is a non-terminal. Note that since "abcdefgh" appears in a variety of different contexts at level one, so will "abcHn" at level two. Thus, no encoding involving the symbols to the left of "a" and no encoding involving symbols to the right of "h" will be developed.
C. EFFECT OF CHOICE OF PARAMETERS IN THE RECODING PROCEDURE

Now let us consider the possible limiting and pre-determining effects of the synchronizing, punctuating, or partitioning of the sample as it is determined by the choice of parameters or the Recoding Procedure. Thus punctuating is the product of choices which either are external choices determined according to heuristic considerations, or which are choices that are an integral part of the Recoding Procedure itself and the fact that the Recoding Procedure is not a combinatorially exhaustive procedure for partitioning the sample.

We claim that a maximal regularity \( R = \langle Y, \phi, P, Y \rangle \) will usually not be "missed" by the Grammar Discovery Algorithm because of the partitioning of \( Y \) by the Recoding Procedure, except when a regularity of equally high type is used instead as the basis for partitioning and recoding.

As before, case (1) (where there is no encoding) presents no problem of losing regularities.

In case (2), we are concerned about the domain (the \( \Delta \) sequence) of a maximal regularity \( R \) being split between two parts of the partition of \( Y \). (Obviously, if the \( \Delta \) of \( R \) lies entirely within one part of the partition of \( Y \), there is no problem).

First of all, the partition is not likely to divide a maximal regularity. The partition is not imposed indiscriminately on \( Y \)--independent of its regularities. Indeed, the partition is the consequence of the existence of regularities of allowably high type and suitable length. More often than not, the hypothesized maximal regularity will itself determine the partition and therefore be located within 1 part of the partition. Recall, in particular, that the
Recoding Procedure considers the regularities in order of decreasing type (that is, Type I first), so that the regularity chosen as the basis for making the partition of $Y$ will, at worst, be of the same type as $R$ itself. Also, within a given type, the regularities are considered in order of decreasing reliability, so that the regularity chosen will have a conditional probability at least as high as that of $R$. Moreover, since the longer regularities are, in practice, considered first (even if the shorter regularities are considered first, it is the longer regularities that are more reliable, so that they tend to appear first in any case) maximal regularity will tend to be used as the basis for making the partition in the first place (in preference to any shorter regularity). Thus, the problem of a maximal regularity being divided is not likely to arise in the first place.

However, suppose this division does occur. Indeed, in a highly structured sample, there may be many reliable regularities that can be used as a basis for partitioning and recoding, and the $\Delta$ of these regularities will tend to overlap. But because of the order of considering regularities for recoding purposes, only regularities at least as reliable as $R$ will be invoked.
D. EFFECT OF PRESENCE OR ABSENCE OF INITIAL PUNCTUATION IN THE SAMPLE

The purpose of this section is to make the assertion (quite contrary to intuition) that the presence or absence of initial punctuation separating sentences in the sample is of little consequence in the grammar discovery process; and that if it is absent, the discovery of sentence boundaries is qualitatively the same punctuation problem as grammar discovery in general, and is moreover not even quantitatively (i.e. combinatorially) much more difficult.

It is important to see that the entire Grammar Discovery Algorithm is a punctuating process—that is, a process of partitioning the sample Y into appropriate parts and then recoding the parts. The problem of inducing appropriate sentence boundaries is no different than the process of inducing appropriate phrase boundaries within sentences.

Naturally, if one wants to induce sentence boundaries, it is necessary that the sample contain repetitions of various sentences (or at least significant parts of them)—just as the induction of phrase boundaries requires repetitions of the phrases. Indeed, it is the repetition of features which establishes them as features. With this precondition in mind, let us use a particular constructed sample Y to argue the main assertion (above).

Suppose that a sample Y consists of 10 copies of 100 different sentences, each sentence being of length 20—the particular numbers and the uniformity of the number of copies and uniformity of sentence length being unimportant to our argument. The sample thus consists of 2,000 sentences. The 2,000 sentences appear in jumbled disorder (if the sentences appeared in a regular order, they would not be individual sentences !). If the Grammar Discovery Algorithm is
operating in the second mode (i.e. with initial punctuation between sentences), the sample appears as a string of 2,000 sentences each separated by an initial punctuation mark (the period). If the Grammar Discovery Algorithm is operating in the first mode (i.e. with no initial punctuation), the sample appears as a string of 20,000 symbols.

In either mode, the Grammar Discovery Algorithm first searches for local regularities and then recodes them. In the second mode, no search for regularities is made across initial punctuation marks. The following happens in the first mode: Since the sample is jumbled, the two sentences on either side of each of the 10 occurrences of a given sentence $\alpha$ will, in general, be different. Indeed, the probability that a particular sentence $\beta$ appears, say, to the left of the given sentence $\alpha$ is only 1/100; and, even with the Birthday Problem phenomena, the odds are against any duplication of any sentence to the left of any of the sentences, and certainly against any reliable occurrence of such a duplication. Thus, it is most unlikely that any of the symbols at the left end of $\alpha$ are related in any way with the symbols occurring to their left. Since the partitioning and recoding of $Y$ is done because of the occurrence of regularities, the symbols within each sentence may be recoded, but groups of symbols spanning the sentence boundary will not. Indeed, each sentence may be what we called a maximal regularity (but it need not be, of course).

Thus, although we cannot predict how the recoding of $Y$ will proceed, we can predict that no groups of symbols spanning a sentence boundary will ever be recoded together. Thus, the sentence boundary will be preserved, as the recoding proceeds from level to level.
Even if each sentence finally becomes encoded as a single non-terminal at a high level, these single non-terminals will display no regularities amongst each other—indeed, the sequence of 2,000 single non-terminals will have maximal entropy or near maximal entropy. This may, of course, be an appropriate moment for inducing disjunctions—but the point is simply that even at this level, no regularities should appear—either for M of 2, 3, 4, etc. because the sentences (being independent units) were jumbled.

Thus, appropriate sentence boundaries will be induced by the same punctuating process as phrase boundaries.

Note also that the search for regularities is not even appreciably shortened by the presence of initial punctuation. If the sentences have average length of 20, most of the searching for local regularities (with small M's of up to 4, 5, or 6) does not occur over the sentence boundaries anyways. Thus, the presence of initial punctuation (across which the search is not made) does not appreciably reduce the combinatorics of the Search.
IV. ADDITIONAL FEATURES OF THE GRAMMAR DISCOVERY ALGORITHM

A. TERNARY MASKS AND DON'T CARE CONDITIONS

In the description of the Grammar Discovery Algorithm, the masks used to search for regularities in the Search Phase were all binary masks—that is, they were sequences of length M over only the symbols "_" ("context") and "#" ("predicted"). The masks were also compact in the sense that the mask expresses a relation within a certain substring of the sample, and all symbols appearing between the left-most symbol of that substring and the right-most symbol of that substring are either in the context part of the regularity or the predicted part.

One may wonder about regularities involving relations between two or more substrings that are widely separated by symbols that are not part of the regularity—as, for example, might occur in mechanically encoded messages. For example, an "a" in a certain position in the sample and a "b" occurring 6 positions later may reliably predict the symbol occurring half-way between them (in position 4).

There are two approaches to finding such regularities.

If binary compact masks are being used, this regularity will be detected as a regularity at a higher level—particularly if the distance between the substrings is moderately large or large. Or, this regularity may be detected in the Generalization Process—that is, a sequence of rules of production are found in which one of more of their context positions are found to vary "freely" over the entire current alphabet $V_c$ and these rules are then replaced by one generalized rule having a universally quantified metavariable in the appropriate positions. This approach is particularly appropriate when the distance between the substring of the relation is rather small.
The second approach is more certain and involves using ternary masks. A ternary mask \( M \) is a sequence of length \( M \) over the symbols "_", "%", and the symbol "#" (which is called don't care). There are \( 3^M \) ternary masks of length \( M \) as opposed to only \( 2^M \) binary masks of length \( M \). To illustrate the use of the "don't care" symbol in ternary masks, consider the following example: The ternary mask is "_#%" (Note that the smallest ternary mask using all 3 symbols is of length 3). This mask refers to the relationship in which a particular symbol predicts a third symbol, regardless of the second symbol. A regularity based on this mask might be <"a#%", "_b", 1.0, Y>—that is, the symbol "a" appearing in any position of the sample \( Y \) reliably predicts the occurrence of symbol "b" two positions later. To express this same regularity using only binary masks would require \( C \) (the number of symbols in the current alphabet) separate regularities—namely, <"aa%", "_b", 1.0, Y>, <"ab%", "_b", 1.0, Y>, <"ac%", "_b", 1.0, Y>, etc. In general, if there are \( h \) don't care positions in a mask, one ternary mask replaces \( C^h \) separate binary masks. Thus, there are \( 2^{MC} \) binary regularities of length \( M \), and there are \( 3^{MC-h} \) ternary regularities.

The smallest non-degenerate situation where both a binary and ternary mask might be used occurs for values of \( M=3 \) (since the mask must contain at least one "_", one "%", and one "#"), \( C=2 \) (a binary alphabet), and \( h=1 \) (at least one don't care position). For this case, the binary mask approach is more efficient than the ternary mask approach. But when

\[
h > \log_C \left( \frac{3}{2} \right)^M
\]

*The variable \( FR \) controls the use of binary or ternary masks in the computer program. Ternary masks are not implemented in the computer program at this time.*
--which encompasses most all other situations (particularly since C will tend to be moderately large, and M is usually small), the ternary masks will involve fewer combinations. Moreover, it should be remembered that the Generalization Process is itself a combinatorial process, so that any apparent efficiency of binary masks in the Search Phase is lost in the Generalization Process. Also, the Generalization Process requires a rather large and complete sample before it can operate (at least without making conjectures), and this is another factor in favor of the efficiency of ternary masks.
B. ALPHABET-ENLARGING VERSUS HUFFMAN-TYPE ENCODINGS

As mentioned earlier, there are two types of transformations: alphabet-enlarging, and non-alphabet-enlarging.

In the alphabet-enlarging transformation (which is the approach used above), there is a non-terminal alphabet. Strings containing these non-terminals from time to time replace strings in the given sample. Thus, at level one the current alphabet \( V_c \) consists only of the terminal alphabet \( V_T \); but at higher levels, new non-terminals are added as needed (as rules of production are developed) and the current alphabet \( V_c \) becomes the union of the original terminal alphabet \( V_T \) and these added new non-terminals. Because these non-terminals are new symbols which are not found in the original sample at level one, a string containing one of these non-terminals can be recognized as being a string from a level higher than level one.*

In the non-alphabet-enlarging transformation, non-terminal symbols are not used. The sample at each level (with the possible exception of the last and highest level) consists only of the symbols occurring at level one. The encoding is done by mapping one string over the given alphabet to another string over the same alphabet. In general, not all strings over the given alphabet are possible in the given sample \( Y \), so that some of the "impossible" strings are available to serve the function which the new non-terminals serve in the alphabet-enlarging approach. Generally, these "impossible" strings will be rather long strings. Note that the two sides of the

*The variable EXR in the computer program specifies whether alphabet-enlarging or Huffman-type encoding is to be done. Although Huffman encoding was attempted in early stages of this work (and the subroutine HUFF remains in the computer program), this feature is no longer operative in the computer program.
rules of production developed from this approach are strings over the same alphabet. Note also that the antecedent (left) side is almost always going to be longer than the consequent (right) side of the rule. This is in contrast to the situation when alphabet-enlarging transformations are used. Thus, when alphabet-enlarging transformation is used, the Grammar Discovery Algorithm is inherently a context-sensitive (or simpler) process; while when the non-alphabet-enlarging approach is used, unrestricted rewrite rules emerge.

Naturally, one must choose the new long strings with care---so that they cannot be confused with natural strings in the sample. These new longer strings must be treated as an encoded unit, and we are not concerned with the statistics or internal regularities that these strings may exhibit. Since the new strings tend to make more symbol strings possible in the sample, the total entropy increases as a result of the addition of them. An optimal coding procedure, such as the Huffman code procedure (Fano), can be used to produce these new strings in a manner that it is possible to reverse the code and recover the original sample. When a Huffman coding procedure is used on a sample of messages which are themselves strings over the encoding alphabet, it is necessary to encode every symbol in the original sample. As this type of encoding is applied from level to level, all possible strings become possible in the sample, and the information rate of the sample increases to a maximum.

The two approaches can be combined into a limited-alphabet transformation. In this approach, the number of non-terminals that can be added is limited to a fixed number $h$. If more non-terminals are needed, combinations (in the Huffman sense) of the allowed non-terminals must be used. Unrestricted rewrite rules would then result.
C. RECODING WITH NOISE IN THE SAMPLE

When a recoding is based only on Type I regularities,* no information is lost in the recoding. Whenever a recoding is based on a Type II or lower type regularity \( R = \langle \gamma, \phi, P, Y \rangle \), then \((1-p)\) 100% of the time the transformation is not faithful to the sample. There can be two motivations for using regularities of Type II or lower type:

(1) The sample \( Y \) is assumed to contain a small amount of noise and is therefore assumed not to be 100% accurate itself. In this situation, if a regularity is found which has a conditional probability of nearly 100% accurate itself. In this situation, if a regularity is found which has a conditional probability of nearly 100%, it is assumed that the parts of the sample that do not conform to the regularity are in error. The use of a regularity with conditional probability near 100% therefore has the effect of correcting the sample and removing the alleged errors in it.

(2) The sample \( Y \) is accepted as being 100% accurate, but one desires to develop a very simple grammar for the sample. Parts of the sample which do not conform to regularities whose conditional probabilities are near 100% are assumed to represent "exceptions", and one is not willing to sacrifice the parsimony necessary to account for all the exceptions. Thus, the simplified grammar will be one that represents most of the sample most of the time. This simplified grammar will have fewer rules of production (more parsimony), but will have higher entropy (more information from the sample is lost).

*The variable ALLOW in the computer programs regulates the type of regularity allowed in the recoding.
D. THE CONTEXT-FREE CASE

In the Recoding Phase, certain strings of symbols are recoded. Up to now, this recoding (assuming an alphabet-enlarging encoding) always has involved substituting a new non-terminal symbol for a string of contiguous symbols that are well predicted by a certain context. Thus, the antecedent (left) side of any rule of production that is ultimately developed has less than or the same number of symbols as the consequent (right) side of the rule. Thus, each rule developed will be context-sensitive, and the grammar ultimately induced will also be context-sensitive. Thus, a grammar discovery algorithm based on contexts and regularities is inherently a context-sensitive process.

It might appear that because the whole Recoding Phase is based on context and conditional probabilities that only strictly-context-sensitive rules can be developed. Indeed, regularities that have no context part (as, for example, regularities developed from the mask $\%^{M}$) merely record frequencies of occurrences of substrings—rather than any relationship among symbols in the sample. Therefore, it would appear that every rule must have a non-empty context and therefore be strictly-context-sensitive. In fact, however, context-free rules (and regular rules—which are context-free rules with an $M$ of 2) can be developed in one of several ways by the Grammar Discovery Algorithm.

First of all, a context-free rule can be induced using the Generalization Process. Whenever a combinatorially complete set of strictly-context-sensitive rules exists (that is, given $h$ context positions in a rule, all $c^h$ possible strictly-context-sensitive rules appear in the grammar), the Generalization Process can then induce a
context-free rule to replace the entire set of strictly-context-sensitive rules. Certainly, if a given substitution is made in all possible contexts, then the substitution can be made without regard to that context—that is, the substitution is context-free. However, because the Generalization Process is a combinatorial process, this approach is generally not available—whether for reasons of the large combinatorics involved or because not all \( C^h \) strictly-context-sensitive rules appear. The latter condition is indeed most restrictive. One can loosen the Generalization Process by allowing the generalization if no exceptions are found—that is, if among all the contexts that do appear, the substitution is invariably made, then a context-free substitution can be generalized. However, even this approach involves an exhaustive examination of a fair number of cases.

A second approach is based on the idea of maximal regularity, which was defined earlier. Whenever a rule is developed from a maximal regularity \( R = \langle Y, \phi, P, Y \rangle \), a context-free rule can be induced, namely

\[
N \rightarrow \Delta,
\]

where \( N \) is a new non-terminal, and where \( \Delta \) is the \( \Delta \)-sequence for \( R \). The justification for the writing of this context-free rule in this situation is that the regularity \( R \) cannot be extended. This is the case because \( \Delta \) appears in the sample \( Y \) in a wide variety of contexts (the distribution of symbols surrounding \( \Delta \) have a non-zero and presumably high entropy), and therefore \( \Delta \) is "free" of its contexts. However, determining that a regularity is maximal (although involving only the checking of \( 2 \cdot C \) p-extensions and \( 2 \cdot C \) c-extensions) is itself as small-scale combinatorial process and somewhat unnatural. However, this second approach suggests a third approach.

The third approach is based on the fact that regularities are
considered in the Recoding Phase in decreasing order of their length. Thus, if any regularities discovered in the Search Phase are maximal regularities, they will be considered first in the Recoding Phase. Thus, we can simply write

$$N \rightarrow \Delta$$

whenever we have a regularity $\mathcal{R}$. Moreover, note that if $\mathcal{R}$ is not maximal, nothing is lost by this approach because at a higher level, the maximal regularity will still appear as a regularity encompassing $N$ and various other symbols. Suppose the $\Delta$-sequence of a maximal regularity $\mathcal{R}'$ of length $M'$ is $\Delta'$, and that a regularity $\mathcal{R}$ of length $M$ $M < M'$ is used to write the rule

$$N \rightarrow \Delta$$

where $\Delta$ is the $\Delta$-sequence of $\mathcal{R}$. Then, with $\Delta$ a proper substring of $\Delta'$, the maximal regularity $\mathcal{R}'$ manifests itself at a higher level over the string

$$\Delta(1)' \cdots \Delta'(h) \quad N \quad \Delta'(k) \cdots \Delta'(M')$$

of length $M' - M$. Using the maximal regularity to write a rule now (assuming that $M' - M$ is now $M_2$), we might get $A \rightarrow \Delta'(1) \cdots \Delta'(h)$ $N \quad \Delta'(k) \cdots \Delta'(M')$, where $A$ is a non-terminal. Thus, we would have two rules, both context-free, and this third approach works.*

*The variable RIT controls whether context-free rules are written in this way in the computer program.
V. EXAMPLES

In this section, we consider a number of examples that illustrate the Grammar Discovery Algorithm.

EXAMPLE A: Consider the following sample of sentences:

This sample presumably is from the language whose sentences are all of the form \( A^{2n} \), where \( n \geq 1 \). The terminal alphabet consists only of the single symbol "A".

The input to the Grammar Discovery Algorithm might be as follows:

| MAXIMUM NUMBER OF LEVELS TO BE TRIED | 3 |
| SMALLEST M TO BE TRIED | 2 |
| LARGEST M TO BE TRIED | 4 |
| DIRECTION OF CONSIDERING M IN RECODE | DESCEND |
| \( \Lambda \alpha \beta \gamma \) -- COEFFICIENT OF PARSIVIANY | 1.00000 |
| \( \Lambda \alpha \beta \gamma \) -- COEFFICIENT OF RECURSIVE PARSIVIANY | 0.10000 |
| SOURCE OF INITIAL SYMBOI STRING | READ |
| CODE - INITIAL PUNCTUATION MARK | 2ND |
| TYPE OF GRAMMAR DESIRED (Masks TO BE TRIED) | LS-RS |
| P1 - CONTROLS ANCESTOR SIDE OF RULES | CF |
| WORST TYPE OF PII ALLOWED TO ENTER GCEE | YES |
| USE OF BEST Y STRICTLY | YES |
| ER -- RADIX CF MASK | 2 |
| PRINT CONTROL -- PRINT UNSORTED P(I1) | NO |
| PRINT CONTROL -- PRINT SORTED P(I1) | NO |
| PRINT CONTROL -- PRINT CPHW/C P(I1)=S | ALL |
| PRINT CONTROL -- PRINT SEQUENCE NUMBERS | YES |
| SLIGHTLINE USED FOR RECODING | RECODE |
| CVERFICE VALUE FOR MPEST (MCH) | 0 |
| EARLY ELIMINATION CF UNALLOWABLE P(I1) | NO |
| INCLUDE ALL-PREDICTED MASK | NO |
| START RECODE AT TIME CF MPEST | YES |
| NSFL -- METHOD FOR COMPUTING \( P \) | 4 |
| CLIF-CUT ON FINDING GCEE M | NO |
| METHOD OF FINDING RECURSIONS | RULE |
| INCLUDE IDENTIFY RULES | NO |
| TEST FOR RECURSION | YES |

| SIZE OF TERMINAL ALPHABET (INITIAL STRING) | 1 |
| TERMINAL ALPHABET: | A |
| INITIAL PUNCTUATION MARK (MODE=2) | |
| SIZE OF BASIC NCA-TERMINAL ALPHABET | 10 |
| SYMBOLS OF THE BASIC NCA-TERMINAL ALPHABET: 0123456789 | |
At level 1, for an M of 2, the Grammar Discovery Algorithm will develop exactly one possible rule of production, as follows:

```
LEVEL 1

SYMBOL STRING OF LENGTH 35 AND USING ALPHABET OF SIZE 1

AA.AAAA.AAAAA.AAAAAAA.AAAAAAAA.
```

```
LEVEL 1  M = 2

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) 2
NUMBER OF TYPE 2 SEQUENCES (MESSAGE) 0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE) 0
NUMBER OF TYPE 4 SEQUENCES (MESSAGE) 0
NUMBER OF TYPE 5 SEQUENCES (MESSAGE) 0
NUMBER OF DIFFERENT MASKS USED 2
NUMBER OF POSSIBLE K- SEQUENCES 1
NUMBER OF POSSIBLE P(I) 2
NUMBER OF DIFFERENT CONTEXTS 2
SEPARATION VALUE BETWEEN TYPES 3 AND 4 0.50000
EPSILON FOR DEFINING TYPES 2 AND 3 0.34000

TENTATIVE RULES OF PRODUCTION FOR Y OF 2

\[ 0 \rightarrow AA \]
```

```
ENTROPY TERM 0.0
PARSIMONY TERM 1.00000
RECURSIVE PARSIMONY TERM 0.0
VALUE OF H FOR THIS RECODING 1.00000
```

```
NUMBER OF RULES OF PRODUCTION 1
NUMBER OF RECURSIVE RULES 0
NUMBER OF IDENTIFY RULES C
NUMBER OF TIMES RULES ARE APPLIED 1
```

CURRENT STRING Y

AA.AAAA.AAAAA.AAAAAAA.AAAAAAAA.

NEW STRING

GG.COG.COG.COG.COG.
The $p_{ij}$-graph for the one rule actually used in the recoding is rather sparse:

<table>
<thead>
<tr>
<th>Graph of $P(i)$ used in recoding for level 1 and $M$ of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>0.98</td>
</tr>
<tr>
<td>0.90</td>
</tr>
<tr>
<td>0.92</td>
</tr>
<tr>
<td>0.88</td>
</tr>
<tr>
<td>0.86</td>
</tr>
<tr>
<td>0.84</td>
</tr>
<tr>
<td>0.78</td>
</tr>
<tr>
<td>0.76</td>
</tr>
<tr>
<td>0.74</td>
</tr>
<tr>
<td>0.72</td>
</tr>
<tr>
<td>0.68</td>
</tr>
<tr>
<td>0.64</td>
</tr>
<tr>
<td>0.60</td>
</tr>
<tr>
<td>0.58</td>
</tr>
<tr>
<td>0.56</td>
</tr>
<tr>
<td>0.54</td>
</tr>
<tr>
<td>0.52</td>
</tr>
<tr>
<td>0.50</td>
</tr>
<tr>
<td>0.48</td>
</tr>
<tr>
<td>0.46</td>
</tr>
<tr>
<td>0.44</td>
</tr>
<tr>
<td>0.42</td>
</tr>
<tr>
<td>0.40</td>
</tr>
<tr>
<td>0.38</td>
</tr>
<tr>
<td>0.36</td>
</tr>
<tr>
<td>0.34</td>
</tr>
<tr>
<td>0.32</td>
</tr>
<tr>
<td>0.30</td>
</tr>
<tr>
<td>0.28</td>
</tr>
<tr>
<td>0.26</td>
</tr>
<tr>
<td>0.24</td>
</tr>
<tr>
<td>0.22</td>
</tr>
<tr>
<td>0.20</td>
</tr>
<tr>
<td>0.18</td>
</tr>
<tr>
<td>0.16</td>
</tr>
<tr>
<td>0.14</td>
</tr>
<tr>
<td>0.12</td>
</tr>
<tr>
<td>0.10</td>
</tr>
<tr>
<td>0.08</td>
</tr>
<tr>
<td>0.06</td>
</tr>
<tr>
<td>0.04</td>
</tr>
<tr>
<td>0.02</td>
</tr>
</tbody>
</table>

| Number of Type 1 Sequences (Structural) | 1 |
| Number of Type 2 Sequences (Message) | 0 |
| Number of Type 3 Sequences (Message) | 0 |
| Number of Type 4 Sequences (Message) | 0 |
| Number of Type 5 Sequences (Noise) | 0 |
| Number of $P(i)$ | 1 |
Similarly, for an M of 3 of level 1, the sample admits
of the development of only one rule of production:

\[ \text{LEVEL} = 1 \quad M = 3 \]

\begin{tabular}{|l|l|}
  \hline
  \text{NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL)} & 4 \\
  \text{NUMBER OF TYPE 2 SEQUENCES (MESSAGE)} & 0 \\
  \text{NUMBER OF TYPE 3 SEQUENCES (MESSAGE)} & 0 \\
  \text{NUMBER OF TYPE 4 SEQUENCES (MESSAGE)} & 0 \\
  \text{NUMBER OF TYPE 5 SEQUENCES (NOISE)} & 0 \\
  \text{NUMBER OF DIFFERENT MASKS USED} & 4 \\
  \text{NUMBER OF POSSIBLE \( \pi \)-SEQUENCES} & 1 \\
  \text{NUMBER OF POSSIBLE \( \pi(i) \)} & 4 \\
  \text{NUMBER OF DIFFERENT CONTEXTS} & 6 \\
  \text{Epsilon value between types 2 and 3} & 0.33333 \\
  \text{Epsilon for defining types 2 and 3} & 0.50000 \\
  \hline
\end{tabular}

\text{TENTATIVE RULES OF PRODUCTION FOR M OF 3}

\[ 0 \rightarrow \rightarrow \rightarrow AAA \]

\begin{tabular}{|l|l|}
  \hline
  \text{FIRST TERM} & 0.0 \\
  \text{PARSIMONY TERM} & 1.00000 \\
  \text{RECURSIVE PARSIMONY TERM} & 0.0 \\
  \text{VALUE OF H FOR THIS RECODING} & 1.00000 \\
  \hline
\end{tabular}

\text{NUMBER OF RULES OF PRODUCTION} = 1
\text{NUMBER OF RECURSIVE RULES} = 0
\text{NUMBER OF IDENTITY RULES} = 0
\text{NUMBER OF TIMES RULES ARE APPLIED} = 1

\text{CURRENT STRING Y}

\[ AAAAAAAAAAAAAAAAAAAAAA. \]

\text{NEW STRING}

\[ AA.CA.CG.CCAA.CCGA. \]

Note that the new string contains both the original terminal symbol "A" and the induced non-terminal "0".
The $p_{ij}$-graph for $M$ of 3 is as sparse as for $M$ of 2, and will not be shown here.

For an $M$ of 4, one rule is again induced at level 1, as follows:

\[
\text{TENTATIVE RULES CF PRODUCTION FOR M CF 4}
\]

\[
0 \quad \text{---->} \quad \text{AAAA}
\]

Application of this rule leads to the following new string:

\[
\text{AAAAедакto�}
\]

We omit the $p_{ij}$-graph and other output for $M$ of 4.

Note that we have temporarily suppressed consideration of the effect of identity rules in this example. In each case the total value of $H$ consisted only of the value of parsimony. For $M$ of 2, 3, and 4, the value of parsimony was the same. The best $M$, therefore, would be the shortest $M$ and an $M$ of 2 would be chosen, as noted by the following:

\[
\text{REST M IS 2 WITH M CF 1,CCDOD}
\]

More significantly, however, is that 2 is in fact the best $M$ when the effect of the identity rules are considered. For an $M$ of 3 or 4 (but not an $M$ of 2), a recoding of the entire sample requires application of an identity rule which transforms the terminal symbol "A" into itself. This rules, when given any non-zero weight at all, is sufficient to dictate the choice of $M$ of 2 as the best $M$.

In Example 2, the role of these identity rules will again be crucial.
To review level 1, the following is a table of the
regularities observed in the sample

<table>
<thead>
<tr>
<th>#</th>
<th>LEVEL</th>
<th>LENGTH</th>
<th>FRCB</th>
<th>TYPE</th>
<th>MASK</th>
<th>CONTEXT</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1.00</td>
<td>1</td>
<td>%</td>
<td>AA</td>
<td>A_</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AA</td>
<td>A_</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>4</td>
<td>1.0C</td>
<td>1</td>
<td>%</td>
<td>AAA</td>
<td>AA_</td>
</tr>
</tbody>
</table>

and the following is the one non-identity rule of
production induced at level 1:

0 ----> AA

The new string resulting from application of this one
rule of production is as follows:

NEW STRING

0.CC.CCO.CCO0.CCO0C.

The Algorithm now accepts this new string as its input for
level 2:

--------------------------- LEVEL 2 ---------------------------
SYMBOL STRING OF LENGTH 20 AND USING ALPHABET OF SIZE 2

0.CC.CCO.CCO0.CCO0C.
For M of 2 at level 2, the Grammar Discovery Algorithm proceeds just as it did for level 1. The same regularities that were observed at level 1 are observed in this particular sample, except that they occur in terms of the induced non-terminal "0" instead of in terms of the terminal symbol "A".

The rule of production 1 →→ 00 is the rule of production which the Algorithm begins to induce. However, before inducing a new rule of production at a level above level 1, the possibility of instead inducing a recursive rule of production must be considered. Note that the rule 1 →→ 00 is isomorphic to the rule 0 →→ AA, and that the symbol "0" is common to both the antecedent side of the rule at level 2, and the consequent side of the rule at level 1. The same symbol positions of both rules are predicted and context symbols.

Therefore, instead of inducing the rule 1 →→ 00 at level 2 and adding this rule to the induced grammar, we instead induce the recursive rule A →→ AA. We delete 0 →→ AA as a rule, and we suppress the rule 1 →→ 00. We obtain

\[\text{TENTATIVE RULES OF PRODUCTICA FOR M OF 2}\
\text{RECUPSION NO. 1}\
A \quad \rightarrow \quad AA\]

The new rule (which is a recursive rule) is applied in all possible ways, yielding:

A.A.A.A.A.
The termination conditions are now satisfied, since the sample is now entirely reduced to sentences of length 1. The induced grammar is thus $A \rightarrow AA$. No further $M$'s need be considered at level 2; however, if we did continue we would induce $A \rightarrow AAA$ for $M$ of 3, and $A \rightarrowAAAA$ for an $M$ of 4. The parsimony again would be identical for $M$ of 2, 3, and 4, except for the fact that identity recodings would be necessary when $A \rightarrow AAA$ and $A \rightarrowAAAA$ are used. Thus, again, an $M$ of 2 would be the best $M$. 
EXAMPLE B: Suppose the sample consists of a lightly different set of sentences---namely, sentences of the form $A^n$, $n \geq 2$, as follows:

```
LEVEL 1
SYMBOL STRING OF LENGTH 63 AND USING ALPHABET OF SIZE 1

AA.AAA.AAAA.AAAAA.AAAAA.AAAAA.AAAAA.AAAAA.AAAAAA.AAAAAA.AAAAAAA
AA.
```

The analysis will proceed in a manner similar to that of Example A, except that now two rules of production will always result. The two rules will be either $A \rightarrow AA$ and $A \rightarrow AAA$, or $A \rightarrow AA$ and an identity rule mapping $A$ into itself. The additional rule, in both cases, accommodates the sentences of odd length.

Note that, as in Example A, other rules (such as perhaps $A \rightarrow AAA$ or $A \rightarrow AAAAA$) will not be developed because $A \rightarrow AA$ subsumes both and is more parsimonious.
**EXAMPLE C:** In this example, we consider a sample of well-formed sentences of the propositional calculus in Polish notation, with one term and one operation.

The sample is

\[ P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10} \]

The input to the Grammar Discovery Algorithm is as follows:

- **MAXIMUM NUMBER OF LEVELS TO BE TRIED:** 10
- **SMALLEST N TO BE TRIED:** 2
- **LARGEST N TO BE TRIED:** 3
- **_DIRECTION_ OF _CONSIDERING_ N IN _REGUDE:_** DESCEND
- **LAMBDA -- COEFFICIENT OF _PARSIMONY:_** 1.0
- **LAMK -- COEFFICIENT OF _RECURSIVE_ _PARSIMONY:_** 0.1
- **SOURCE OF _INITIAL_ _SYMBOL_ _STRING:_** READ
- **CODE -- _INITIAL_ _PUNCTUATION_ _MARK:_** 
- **TYPE OF _GRAMMAR_ _RECOGNIZED_ (_Masks_ TO BE _TRIED:_** LS-RS
- **RIT -- _CONTROLS_ _ANTENODER_ _SIDE_ _OF_ _RULES:_** 4
- **ERROR TYPE OF _RULE_ ALLOWS TO ENTER CODE:** YES
- **USE OF _FIRST_ _STRATEGY:_** 2
- **MR --- _PRINT_ _OUT_ _MARK:_** 2
- **PRINT CONTROL -- _PRINT_ _UNSORTED_ _P(I):** YES
- **PRINT CONTROL -- _PRINT_ _SEATED_ _P(I):** NO
- **PRINT CONTROL -- _PRINT_ _GRAPH_ _OF_ _P(I)-S:_** NO
- **PRINT CONTROL -- _PRINT_ _SEQUENTIAL_ _NUMBERS:_** YES
- **SUBROUTINE USED FOR _RECODING:_** RECODE
- **OVERFLOW VALUE FOR _TEXT_ (_ACH:_** 0
- **EARLY ELIMINATION OR _UNRECOGNIZABLE_ _P(I):** NO
- **INCLUDE _ALL-RECOGNITED_ _MARK:_** NO
- **START _RECODE_ AT TIME _OF_ _FIRST_ _ERROR:_** YES
- **ASFL -- _METHOD_ FOR _COMPLETION:_** 4
- **DELETION _UNRECOGNITED_ _MARK:_** NO
- **METHOD FOR _BUILDING_ _RULES:_** RULE
- **INCLUDE _IDENTITY_ _RULES:_** NO
- **TEST FOR _RECODING:_** YES

**SIZE OF TERMINAL ALPHABET (_INITIAL_ _STRING:_** 2
**TERMINAL ALPHABET:** +P
**INITIAL _PUNCTUATION_ _MARK_ (_MYP=?):** .

**SIZE OF BASIC NON-TERMINAL ALPHABET:** 10
**SYMBOLS OF THE BASIC NON-TERMINAL ALPHABET:** 1234567890
For an M of 2, three rules of production are developed, as follows:

LEVEL = 1  M = 2

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) = 0
NUMBER OF TYPE 2 SEQUENCES (MESSAGE) = 2
NUMBER OF TYPE 3 SEQUENCES (MESSAGE) = 4
NUMBER OF TYPE 4 SEQUENCES (MESSAGE) = 0
NUMBER OF TYPE 5 SEQUENCES (NOISE) = 2
NUMBER OF DIFFERENT MASKS USED = 2
NUMBER OF POSSIBLE M-SEQUENCES = 4
NUMBER OF POSSIBLE P(I) = 8
NUMBER OF P(I) = 8
NUMBER OF DIFFERENT CONTEXTS = 4
SEPARATION VALUE BETWEEN TYPES 3 AND 4 = 0.50000
EPSILON FOR DEFINING TYPES 2 AND 3 = 0.34000

TENTATIVE RULES OF PRODUCTION FOR M OF 2
1 ----> PP
2 ----> ++
3 ----> P+

The value of H obtained upon applying these rules is as follows:

ENTROPY TERM = 1.20941
PARSIMONY TERM = 3.00000
RECURSIVE PARSIMONY TERM = 0.0

VALUE OF H FOR THIS RECODING = 4.20941

NUMBER OF RULES OF PRODUCTION = 3
NUMBER OF RECURSIVE RULES = 0
NUMBER OF IDENTITY RULES = 0
NUMBER OF TIMES RULES ARE APPLIED = 3

The new string obtained is

NEW STRING

The $p_{ij}$-graph for this recoding is as follows:

| $p_{ij}$ | 1.00+++ | 0.94 | 0.90 | 0.90 | 0.88 | 0.86 | 0.84 | 0.82 | 0.80 | 0.78 | 0.76 | 0.74 | 0.72 | 0.70++ | 0.68 | 0.66 | 0.64 | 0.62 | 0.60 | 0.58 | 0.56 | 0.54 | 0.52 | 0.50+++ | 0.48 | 0.46 | 0.44 | 0.42 | 0.40 | 0.38 | 0.36 | 0.34 | 0.32 | 0.30 | 0.28 | 0.26 | 0.24 | 0.22 | 0.20 | 0.18 | 0.16 | 0.14 | 0.12 | 0.10 | 0.08 | 0.06 | 0.04 | 0.02 | 0.00+++ |
|----------|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

<table>
<thead>
<tr>
<th>Type</th>
<th>1 Sequences (Structural)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>2 Sequences (Message)</td>
<td>2</td>
</tr>
<tr>
<td>Type</td>
<td>3 Sequences (Message)</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>4 Sequences (Message)</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>5 Sequences (Noise)</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>$p(i)$</td>
<td>3</td>
</tr>
</tbody>
</table>
For an M of 3, we obtain the following:

\[
\begin{array}{l}
\text{LEVEL} = 1 \quad \text{M} = 3 \\
\text{NUM}_{1} = 0 \\
\text{NUM}_{2} = 4 \\
\text{NUM}_{3} = 7 \\
\text{NUM}_{4} = 6 \\
\text{NUM}_{5} = 15 \\
\text{NUM}_{\text{DIFF}} = 4 \\
\text{NUM}_{\text{POSE}} = 8 \\
\text{NUM}_{\text{PI}} = 32 \\
\text{NUM}_{\text{DIFF}} = 32 \\
\text{SEP}_{34} = 0.50000 \\
\text{EPSILON} = 0.33333 \\
\end{array}
\]

TENTATIVE RULES OF PRODUCTION FOR M OF 3

1 \rightarrow PP+

The value of H is as follows:

\[
\begin{array}{ll}
\text{ENTROPY TERM} & 0.64752 \\
\text{PARSIMONY TERM} & 2.00000 \\
\text{RECURSIVE PARSIMONY TERM} & 0.0 \\
\text{VALUE OF H FOR THIS RECODING} & 2.64752
\end{array}
\]

The string is then recoded as follows:

NEW STRING

\[p_{1}l_{1}++pp_{1}++p_{1}+p_{1}+p_{1}+p_{1}+p_{1}+p_{1}+p_{1}+\]

Note that the rule 1 \rightarrow PP+ is reapplied several times in the recoding.
After considering tentative recodings based on an
M of 2, 3, and 4, the Algorithm concludes that

**BEST **4 IS **3 WITH H OF **2.64752

and then uses the rule

to recode the sample. The resulting string is then the
input to the Algorithm at level 2.

--- RULES OF PRODUCTION FOR LEVEL 1 ---

1 -----> pp+

At level 2, the Search Phase again searches for
regularities in the new sample, develops rules of
production, and computes the value of H for the
resulting transformations. For an M of 2 at level 2,
we obtain the following:

**LEVEL= 2  M= 2**

| NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) | 1 |
| NUMBER OF TYPE 2 SEQUENCES (MESSAGE)  | 2 |
| NUMBER OF TYPE 3 SEQUENCES (MESSAGE)  | 4 |
| NUMBER OF TYPE 4 SEQUENCES (MESSAGE)  | 1 |
| NUMBER OF TYPE 5 SEQUENCES (NOISE)    | 10 |
| NUMBER OF DIFFERENT MASKS USED        | 2 |
| NUMBER OF POSSIBLE M-SEQUENCES         | 16 |
| NUMBER OF POSSIBLE P(i)               | 32 |
| NUMBER OF P(i)                        | 13 |
| NUMBER OF DIFFERENT CONTEXTS          | 7 |
| SEPARATION VALUE BETWEEN TYPES 3 AND 4 | 0.50000 |
| EPSILON FOR DEFINING TYPES 2 AND 3    | 0.34000 |
The value of $H$ is as follows:

<table>
<thead>
<tr>
<th>Term</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entropy Term</td>
<td>2.15596</td>
</tr>
<tr>
<td>Parsimony Term</td>
<td>4.00000</td>
</tr>
<tr>
<td>Recursive Parsimony Term</td>
<td>0.0</td>
</tr>
<tr>
<td>Value of $H$ for this recoding</td>
<td>6.15596</td>
</tr>
</tbody>
</table>

- Number of Rules of Production: 4
- Number of Recursive Rules: 0
- Number of Identity Rules: 0
- Number of Times Rules are Applied: 7
The $p_{ij}$-graph for an $M$ of 2 at level 2 is as follows:

```
  | 1.00  | 0.88  | 0.62  | 0.39  | 0.20  | 0.08  | 0.00  |
  | 0.98  | 0.65  | 0.56  | 0.39  | 0.20  | 0.08  | 0.00  |
  | 0.94  | 0.64  | 0.52  | 0.34  | 0.16  | 0.08  | 0.00  |
  | 0.92  | 0.63  | 0.52  | 0.34  | 0.16  | 0.08  | 0.00  |
  | 0.90  | 0.63  | 0.52  | 0.34  | 0.16  | 0.08  | 0.00  |
  | 0.89  | 0.64  | 0.54  | 0.36  | 0.18  | 0.09  | 0.01  |
  | 0.86  | 0.64  | 0.54  | 0.36  | 0.18  | 0.09  | 0.01  |
  | 0.84  | 0.66  | 0.56  | 0.38  | 0.20  | 0.10  | 0.02  |
  | 0.82  | 0.66  | 0.56  | 0.38  | 0.20  | 0.10  | 0.02  |
  | 0.80  | 0.70  | 0.58  | 0.40  | 0.22  | 0.12  | 0.03  |
  | 0.76  | 0.67  | 0.58  | 0.40  | 0.22  | 0.12  | 0.03  |
  | 0.74  | 0.68  | 0.58  | 0.40  | 0.22  | 0.12  | 0.03  |
  | 0.72  | 0.68  | 0.58  | 0.40  | 0.22  | 0.12  | 0.03  |
  | 0.70  | 0.69  | 0.58  | 0.40  | 0.22  | 0.12  | 0.03  |
```

**Number of Type 1 Sequences (Structural):** 1

**Number of Type 2 Sequences (Message):** 2

**Number of Type 3 Sequences (Message):** 1

**Number of Type 4 Sequences (Message):** 0

**Number of Type 5 Sequences (Noise):** 0

**Number of $p_{ij}$:** 4
For an $M$ of 3 at level 2, the Algorithm considers developing the rule $2 \rightarrow 11+$, but first checks to see whether a recursion is a possibility. In fact, one recursive rule can replace both $2 \rightarrow 11+$ and the rules $1 \rightarrow PP+$ developed at level 1. We thus obtain the following recursion instead:

\begin{align*}
\text{TENTATIVE RULES OF PRODUCTION FOR } M &= 3 \\
\text{RECURSION NO.} &= 1 \\
\text{P} & \rightarrow \rightarrow \rightarrow \text{ PP+}
\end{align*}

Upon applying the recursion once, we obtain the following new string:

\[
P \cdot P \cdot 11+.PP1++.P1+P+.1P+.P+.P1+.1P+
\]

Then upon applying the recursion in every possible way to the new string (and taking into account the fact that the non-terminal "1" is now superseded by the recursive symbol "P"), we obtain the following as the recoding:

\[
P \cdot P \cdot P \cdot P \cdot P \cdot P \cdot P \cdot P
\]

The conditions for terminating the Algorithm now obtain, and not further levels are considered.
EXAMPLE D: In this section, we analyze the behavior of the Grammar Discovery Algorithm for a language---rather than merely a particular finite sample of sentences from the language. We do this to demonstrate that the kinds of results one obtains from the Algorithm for small samples of sentences, from languages with relatively simple grammars, over relatively small alphabets can be expected to obtain also for larger samples from languages with more complex grammars over alphabets with large numbers of symbols. This particular example also serves to illustrate the full range of grammar induction devices discussed, including those which are not implemented in the computer program for the Algorithm, or which require large combinatorial searches---namely, context sensitive regularities, maximal regularities, recursions, disjunctions, as well as context-sensitive and context-free rules of production.

With this in mind, consider the language generated by the following rules:

\[
\begin{align*}
V & \rightarrow abV \mid \emptyset \\
W & \rightarrow cdeW \mid \emptyset \\
X & \rightarrow fghix \mid \emptyset \\
P & \rightarrow VWX \\
Y & \rightarrow 1mY \mid \emptyset \\
Z & \rightarrow nopZ \mid \emptyset \\
Q & \rightarrow Y \mid Z \\
S & \rightarrow P \mid Q
\end{align*}
\]
Similarly, no regularities involving "cde" will appear of length greater than 3 which will be as reliable as those for length 3, and no regularity involving "fghi" of length greater than 4 will be superior to those of length 4.

Finally, similar observations will apply to the two phrases "lm" and "nop".

The second kind of regularity will involve strings longer than the lengths of the basic 5 phrases. These regularities will catalog the propensity of "ab" to follow an "ab". As mentioned above, these regularities will be less reliable than the corresponding shorter regularities.

The third kind of regularity for this language will involve the interface between occurrences of the 5 basic phrases and the symbol "j". These will be of type II or worse.

In order to recode the sample at level 1, each of the 5 basic phrases will have to be recoded in some way internal to themselves. The reason why there will be no inter-phrase recodings is that a type I regularity internal to the phrase exists in each case, while any inter-phrase regularity will be of type II or worse. Exactly which recoding will be used will depend on several externally-specified parameters. For example, if context-sensitive rules are searched for, the Algorithm will find and use a regularity such as

\[
\langle "c\%e", \_d\_, 1.0 \rangle
\]

whereas if only right-sensitive regularities are searched for, something such as

\[
\langle "\%\%e", "cd-", 1.0 \rangle
\]
might be used. The Recoding Procedure, and in particular the point in the sample where recoding begins, all affect the exact recoding chosen. However, even within the variability permitted by these external choices, it can be predicted that the phrase "cde" will be recoded as a unit, and that, for example, no phrase involving "cdecde" will be used (of length 6). Thus, the initial sample of sentences will be punctuated into sub-phrases of length 2, 3, and 4 corresponding to the 5 basic phrases.

However, these phrases are recoded, the opportunity to recode them at level 2 will occur, and it is at level 2 that the recursive structure of the sample will be discovered. Again, the exact way in which the recursions will be discovered will depend on some externally-specified choices. If, for example, the "uvwxy" method of inducing recursions is being used, the Algorithm will note that there are recurrences of the 5 basic phrases "ab", "cde", etc. or recurrences of whatever symbols now represent these phrases. If the rule-oriented method of inducing recursions is being used, then at level 2 there may be recurrences of non-terminals such as V, W, X, Y, and Z. These non-terminals may be recoded, in a manner similar to that of Examples A and B into other non-terminals, and the recursion discovered at level 3.

In any event, the recursive structure of the sample will be discovered somewhere between levels 1 and 3 of the recoding regardless of how the recoding actually proceeds.

Also, the symbol "j" will not be recoded at all because it is the exceptional suffix to "fghi" and the exceptional
Note that the sentences of this language have the general form

\[(ab)^r (cde)^s (fghi)^t \ j \ (lm)^u \ (nop)^v\]

where \(r, s, t \geq 0\) and either \(u\) or \(v\) is 0, while the other is \(\geq 0\).

In the Search Phase, one finds several different kinds of regularities. First, there are regularities within the 5 phrases "ab", "cde", "fghi", "lm" and "nop". Second, there are regularities occurring between each of these 5 phrases and themselves. Third, there are regularities occurring between 2 different ones of these 5 phrases as well as with the symbol "\(j\)."

Consider first the first kind of regularity. Among these will be the regularity which catalogs the fact that an "a" always predicts a "b" to the right. This regularity is a type I left-sensitive regularity. There is also the fact that a "b" is almost always--but not always--preceded by an "a". This is a type II right-sensitive regularity. Similarly, a "b" is usually followed by an "a", but occasionally is followed by a "c" or even an "f". No regularity involving "a" and "b" of length greater than 2 will be as reliable as those of length 2 because, as will turn out, the string "ab" is a maximal regularity.

Similarly, within the phrases "cde" and "fghi", there will be regularities. Some of these will be context-sensitive regularities (something that was impossible in a phrase of length 2), such as the type I regularity

\[\langle "c\%e", \ "_d_", \ 1.0 \rangle\]

Again, all regularities will be within the phrases.
prefix to whatever follows.

The "j" will be followed approximately equally often by either "lm" or "nop." After these 2 phrases are recoded internally, the non-terminals into which they have been recoded will appear as the two possible substitution possibilities following the "j". Thus, a situation where the suffix to the "j" presents maximal entropy will exist. If the phrase "lm" is recoded as Y, and the phrase "nop" is recoded as "Z", then the disjunction \( Q \rightarrow Y \lor Z \) can be induced.

With the 3 recursions discovered at the beginning of each sentence, and the 2 recursions and 1 disjunction at the end of each sentence discovered, each sentence will have the form VWX j Q, or perhaps P j Q, depending on the intermediate recoding that occurred. When each sentence of the sample is reduced to a common form, the conditions for terminating the Algorithm have been satisfied. Thus, in this case, we have induced the original grammar, or one virtually the same as the original grammar.

Note that the different recursions do not get "tangled" because the phrases in which they are embedded are all maximal regularities.
EXAMPLE E: Here again we shall consider a language rather than a finite sample of sentences in order to illustrate a situation which cannot be represented by any finite sample.

Consider an ergodic source generated strings over a terminal alphabet A of size $C_1$. Perhaps the grammar which describes the language is

$$S \rightarrow (a_1 a_2 a_3 \ldots a_{C_1}) S.$$

Any finite subset of the sentences generated by such a grammar (even a subset that is complete in the sense that is includes all sentences of the language up to a certain specified length) has certain regularities which are artifacts of the finiteness of the sample. The nature of ergodicity is precisely the opposite---namely, that there are essentially no regularities in the sample. Therefore, to discuss the ergodic case, one cannot consider any specific finite sample.

Recall that the Algorithm is given an externally specified range of M's. This range is typically from a lower value of 2 up to a small whole number such as 5. The Algorithm is also given an externally specified direction for considering the M's for recoding purposes (almost always descending---and always descending when context-free rules are being generated). Regardless of these two choices, the entropy associated with any ergodic sample is always the maximal value. Thus, a disjunction expressed in terms of phrases of length M (the first M) is induced. If an M of 1 is consider, the disjunction is in terms of single singles from the alphabet. (Note that the only useable mask for an M of 1 is the "predicted" mask which catalogs probability of occurrence).
Thus, the Grammar Discovery Algorithm produces a reasonable result for an ergodic source generating the sentences of the given sample.

The above discussion suggests, in effect, an alternative definition of ergodicity. In this paper, the notion of the universe of masks is well defined, as is the concept of the universe of possible regularities for a given value of $M$. A source is ergodic, therefore, if there are no regularities in the sentences produced by the source except for type VI regularities having a conditional probability $p_{ij}$ equal to $\theta = 1/c^g$, where $C$ is the size of the alphabet, and $g$ is the number of predicted positions in the regularity. In a more practical vein, if the limiting value of the $p_{ij}$'s are their respective $1/C^g$, then we can call the source ergodic.
EXAMPLE E: In this example, we will consider a very simple example of sentences from English. The sample is as follows:

```
THE-BAD-DOG-SAT--
```

The input to the Grammar Discovery Algorithm is as follows:

```
MAXIMUM NUMBER OF LEVELS TO BE TRIED 3
SMALLEST M TO BE TRIED 2
LARGEST M TO BE TRIED 4
DIRECTION OF CONSIDERING M IN RECODE DESCEND
LAMBDA -- COEFFICIENT OF PAR SIMONY 1.00000
LAMK -- COEFFICIENT OF RECURSIVE PAR SIMCNY 0.10000
SOURCE OF INITIAL SYMBOL STRING READ
MODE -- INITIAL PUNCTUATION MARK 2ND
TYPE OF GRAMMAR DESIRED (MASKS TO BE TRIED) LS-RS
FIT - CONTROLS ANTECEDENT SIDE OF RULES CF
WORST TYPE OF P(I) ALLOWED TO ENTER CODE 4
USE OF BEST M STRICTLY YES
FR -- RADIX OF MASK 2
PRINT CONTROL -- PRINT UNSORTED P(I) NO
PRINT CONTROL -- PRINT SORTED P(I) NO
PRINT CONTROL -- PRINT GRAPH OF P(I)-S ALL
PRINT CONTROL -- PRINT SEQUENCE NUMBERS YES
SUBROUTINE USED FOR RECODING RECODE
OVERRIDE -VALUE FOR MBEST (MCH) 0
EARLY ELIMINATION OF UNALLOWABLE P(I) NO
INCLUDE ALL-PREDICTED MASK NU
START RECODE AT TIME OF MBEST YES
HSEL -- METHOD FOR COMPUTING M 4
CUT-OUT ON FINDING GOOD M NO
METHOD OF FINDING RECURSIONS RULE
INCLUDE IDENTITY RULES NO
TEST FOR RECURSION YES

SIZE OF TERMINAL ALPHABET (INITIAL STRING) 14
TERMINAL ALPHABET: ABCDEGHINORST-
INITIAL PUNCTUATION MARK (MODE=2) .

SIZE OF BASIC NON-TERM ALPHABET 10
SYMBOLS OF THE BASIC NON-TERM ALPHABET: 0123456789
```
The search phase produces the following for an M of 2:

\[
\text{LEVEL}=1 \quad \text{M}=2
\]

| NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) | 15 |
| NUMBER OF TYPE 2 SEQUENCES (MESSAGE) | 0  |
| NUMBER OF TYPE 3 SEQUENCES (MESSAGE) | 11 |
| NUMBER OF TYPE 4 SEQUENCES (MESSAGE) | 0  |
| NUMBER OF TYPE 5 SEQUENCES (NOISE)   | 16 |
| NUMBER OF DIFFERENT MASKS USED       | 2  |
| NUMBER OF POSSIBLE M-SEQUENCES       | 156|
| NUMBER OF POSSIBLE P(I)              | 392|
| NUMBER OF P(I)                       | 46 |
| NUMBER OF DIFFERENT CONTEXTS        | 28 |
| SEPARATION VALUE BETWEEN TYPES 3 AND 4 | 0.50000 |
| EPSILON FOR DEFINING TYPES 2 AND 3  | 0.34000 |

TENTATIVE RULES OF PRODUCTION FOR M OF 2

0  ----->  TH
1  ----->  E
2  ----->  B
3  ----->  G
4  ----->  C
5  ----->  T
6  ----->  R
7  ----->  N
8  ----->  B
9  ----->  D

<00>  ----->  DO

<01>  ----->  SA

The value of H for M of 2 is as follows:

| ENTROPY TERM | 1.50000 |
| PARSIMONY TERM | 12.00000 |
| RECURSIVE PARSIMONY TFRM | 0.0 |

VALUE OF H FOR THIS RECORDERING .......... 13.50000

NUMBER OF RULES OF PRODUCTION | 12 |
NUMBER OF RECURSIVE RULES     | 0  |
NUMBER OF IDENTITY RULES     | 0  |
NUMBER OF TIMES RULES ARE APPLIED | 12 |

The sample is then recoded as follows:

.01234567.01894567.0123<00>367.0189<00>367.012345<01>5.01894
5<01>5.0123<00>3<01>5.0189<00>3<01>5.
The \( p_{ij} \)-graph for this recoding based on an \( M \) of 2 is as follows:

*GRAPH OF \( P(1) \) USED IN RECODING FOR LEVEL 1 AND \( M \) OF 2*

```
| 1.60 | ** | ** | ** | ** |
| 0.98 | | | | |
| 0.96 | | | | |
| 0.94 | | | | |
| 0.92 | | | | |
| 0.90 | | | | |
| 0.88 | | | | |
| 0.86 | | | | |
| 0.84 | | | | |
| 0.82 | | | | |
| 0.80 | | | | |
| 0.78 | | | | |
| 0.76 | | | | |
| 0.74 | | | | |
| 0.72 | | | | |
| 0.70 | | | | |
| 0.68 | | | | |
| 0.66 | | | | |
| 0.64 | | | | |
| 0.62 | | | | |
| 0.60 | | | | |
| 0.58 | | | | |
| 0.56 | | | | |
| 0.54 | | | | |
| 0.52 | | | | |
| 0.50 | | | | |
| 0.48 | | | | |
| 0.46 | | | | |
| 0.44 | | | | |
| 0.42 | | | | |
| 0.40 | | | | |
| 0.38 | | | | |
| 0.36 | | | | |
| 0.34 | | | | |
| 0.32 | | | | |
| 0.30 | | | | |
| 0.28 | | | | |
| 0.26 | | | | |
| 0.24 | | | | |
| 0.22 | | | | |
| 0.20 | | | | |
| 0.18 | | | | |
| 0.16 | | | | |
| 0.14 | | | | |
| 0.12 | | | | |
| 0.10 | | | | |
| 0.08 | | | | |
| 0.06 | | | | |
| 0.04 | | | | |
| 0.02 | | | | |
| 0.0 | | | | |
```

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) = 9
NUMBER OF TYPE 2 SEQUENCES (MESSAGE) = 0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE) = 3
NUMBER OF TYPE 4 SEQUENCES (MESSAGE) = 0
NUMBER OF TYPE 5 SEQUENCES (NOISE) = 12
For an $M$ of 3, the following tentative rules of production are developed:

$$LEVEL = 1 \quad M = 3$$

- **NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL)**: 46
- **NUMBER OF TYPE 2 SEQUENCES (MESSAGE)**: 0
- **NUMBER OF TYPE 3 SEQUENCES (MESSAGE)**: 34
- **NUMBER OF TYPE 4 SEQUENCES (MESSAGE)**: 0
- **NUMBER OF TYPE 5 SEQUENCES (NOISE)**: 32
- **NUMBER OF DIFFERENT MASKS USED**: 4
- **NUMBER OF POSSIBLE $M$-SEQUENCES**: 2744
- **NUMBER OF POSSIBLE $P_{II}$**: 10576
- **NUMBER OF $P_{II}$**: 112
- **NUMBER OF DIFFERENT CONTEXTS**: 98
- **SEPARATION VALUE BETWEEN TYPES 3 AND 4**: 0.50000
- **EPSILON FOR DEFINING TYPES 2 AND 3**: 0.33333

**TENTATIVE RULES OF PRODUCTION FOR $M$ OF 3**

- 0  ----->  THE
- 1  ----->  ~6I
- 2  ----->  G~C
- 3  ----->  AT~
- 4  ----->  KAN
- 5  ----->  ~BA
- 6  ----->  D~C
- 7  ----->  G~D
- 8  ----->  OG~
- 9  ----->  D~D

The value of $H$ for the $M$ of 3 is:

- **ENTROPY TERM**: 2.60000
- **PAPSIMONY TERM**: 10.00000
- **RECURSIVE PAPSIMONY TERM**: 0.0

**VALUE OF $H$ FOR THIS RECODING**: 12.00000

The sample is then recoded as follows:

```
~01234~.05634~.01784~.05984~.012353.056353.017853.059853.
```
For an $M$ of 3, the $p_{ij}$-graph is as follows:

```
GRAPH OF P(I) USED IN RECORDING FOR LEVEL 1 AND M OF 3

1.00  ****-----
0.98   |
0.96   |
0.94   |
0.92   |
0.90   |
0.88   |
0.86   |
0.84   |
0.82   |
0.80   |
0.78   |
0.76   |
0.74   |
0.72   |
0.70   |
0.68   |
0.66   |
0.64   |
0.62   |
0.60   |
0.58   |
0.56   |
0.54   |
0.52   |
0.50  ----****
0.48   |
0.46   |
0.44   |
0.42   |
0.40   |
0.38   |
0.36   |
0.34   |
0.32   |
0.30   |
0.28   |
0.26   |
0.24   |
0.22   |
0.20   |
0.18   |
0.16   |
0.14   |
0.12   |
0.10   |
0.08   |
0.06   |
0.04   |
0.02   |
0.0   ----****
```

```
NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL)  6
NUMBER OF TYPE 2 SEQUENCES (MESSAGE)     0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE)     4
NUMBER OF TYPE 4 SEQUENCES (MESSAGE)     0
NUMBER OF TYPE 5 SEQUENCES (NOISE)       0
NUMBER OF P(I)                            10
```

Note that there are 6 type I transformations, and 4 type II transformations.
For an M of 4, the search phase produces the following:

LEVEL = 1  M = 4

NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL)  73
NUMBER OF TYPE 2 SEQUENCES (MESSAGE)  0
NUMBER OF TYPE 3 SEQUENCES (MESSAGE)  81
NUMBER OF TYPE 4 SEQUENCES (MESSAGE)  18
NUMBER OF TYPE 5 SEQUENCES (NOISE)  20
NUMBER OF DIFFERENT MASKS USED  6
NUMBER OF POSSIBLE M-SEQUENCES  38416
NUMBER OF POSSIBLE P(I)  230496
NUMBER OF P(I)  192
NUMBER OF DIFFERENT CONTEXTS  219
SEPARATION VALUE BETWEEN TYPES 3 AND 4  0.500000
EPSILON FOR DEFINING TYPES 2 AND 3  0.250000

TENTATIVE RULES OF PRODUCTION FOR M OF 4

0 -------> THE~
1 -------> BIG~
2 -------> CAT~
3 -------> RAN~
4 -------> BAD~
5 -------> DOG~
6 -------> SAT~

The value of H is as follows:

ENTROPY TERM  0.0
PARSIMONY TERM  7.00000:
RECURSIVE PARSIMONY TERM  0.0

VALUE OF H FOR THIS RECODING............  7.00000

NUMBER OF RULES OF PRODUCTION  7
NUMBER OF RECURSIVE RULES  0
NUMBER OF IDENTITY RULES  0
NUMBER OF TIMES RULES ARE APPLIED  7

The sample is then recoded as follows:

NEW STRING

.0123.0423.0153.0453.0126.0426.0156.0456.
For an $M$ of 4, the $p_{ij}$-graph shows that there are 7 type I regularities in the recoding:

**GRAPH OF $p_{ij}$ USED IN RECODING FOR LEVEL 1 AND $M$ OF 4**

```
1.00 ******
0.98   |
0.96   |
0.94   |
0.92   |
0.90   |
0.88   |
0.86   |
0.84   |
0.82   |
0.80   |
0.78   |
0.76   |
0.74   |
0.72   |
0.70   |
0.68   |
0.66   |
0.64   |
0.62   |
0.60   |
0.58   |
0.56   |
0.54   |
0.52   |
0.50   +-----+
0.48   |
0.46   |
0.44   |
0.42   |
0.40   |
0.38   |
0.36   |
0.34   |
0.32   |
0.30   |
0.28   |
0.26   |
0.24   |
0.22   |
0.20   |
0.18   |
0.16   |
0.14   |
0.12   |
0.10   |
0.08   |
0.06   |
0.04   |
0.02   +-----+
0.0   ******
```

**NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL)**: 7
**NUMBER OF TYPE 2 SEQUENCES (MESSAGE)**: 0
**NUMBER OF TYPE 3 SEQUENCES (MESSAGE)**: 0
**NUMBER OF TYPE 4 SEQUENCES (MESSAGE)**: 0
**NUMBER OF TYPE 5 SEQUENCES (NOISE)**: 0
**NUMBER OF $p_{ij}$**: 7
The Algorithm then selects the best M, as follows:

\[ \text{BEST M IS 4 WITH } \Delta \text{ OF 7.00000} \]

Using the best M (of 4) as the basis for the recoding, the original sample is then recoded, and this recoded version is then used as input to level 2, as follows:

\[ \text{LEVEL 2} \]
\[ \text{SYMBOLO STRING OF LENGTH 41 AND USING ALPHABET OF SIZE 21} \]
\[ .0123.0423.0153.0453.0126.0426.0156.0456. \]

The resolution of this sample at level 2 requires the induction of a disjunction at level 2 --- a feature not now implemented in the computer program. However, if an M of 1 is used at level 2, this sample will be recognized as being derived from a grammar which places a "0" as the first symbol of every string, and then disjunctively places a "1" or a "4" at position 2; a "2" or a "5" at position 3; and a "3" or a "6" at position 4.
VI. FUTURE DIRECTIONS

This paper raises numerous additional questions and suggests further research about grammar induction and grammar discovery. Among these questions are the following:

First, it would be interesting to explore the two-dimensional pattern recognition problem using the methods of the Grammar Discovery Algorithm. The concepts of context-sub-sequences, predicted-sub-sequences, regularities, and hierarchical rules of production would seem to have application to pattern recognition problems as well as grammar induction problems. The kinds of descriptions that the Grammar Discovery Algorithm builds up in analyzing samples would seem to parallel the kind of descriptions one would want a good pattern recognition algorithm to develop about its samples.

Second, a precondition to seriously analyzing the two-dimensional pattern recognition problem would be to rewrite the computer program for implementing the Grammar Discovery Algorithm. The computer program here was an integral part of the research for this paper. As such, the program was constantly changed as the Algorithm was developed. Thus, the computer program is structured in ways peculiar to its development (e.g. the time-consuming hash coding of sequences) but which make it quite inefficient for analyzing large samples.

Third, there are several aspects of the induction problem that are amenable to more formal analysis than has been presented here. For example, the combined measure of entropy and parsimony and recursive parsimony used in the computer program is essentially an *ad hoc* measure based on intuitive considerations. An axiomatic development of a combined measure would be interesting. Also, some of the questions
of convergence and rates of convergence of the Grammar Discovery Algorithm should be possible.

Fourth, it would be interesting to consider extensions of the grammar discovery process that would lend themselves to samples consisting of symbols that are ordered or partially ordered. This would include samples of integers, or of playing cards, or of musical notes which present interesting questions of induction or of identifying underlying structure.
APPENDIX A. DESCRIPTION OF INPUT TO THE COMPUTER PROGRAM IMPLEMENTING THE GRAMMAR DISCOVERY ALGORITHM

The input to the computer program implementing the Grammar Discovery Algorithm consists of three control cards, followed by additional cards containing the sample of sentences.

The first control card contains a variety of parameters which control the operation of the Grammar Discovery Algorithm. All are integer numbers, unless otherwise specified.

<table>
<thead>
<tr>
<th>Card Columns</th>
<th>Variable Name</th>
<th>Description</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>L3</td>
<td>Upper Limit on the number of levels to be tried by the algorithm. This parameter has no theoretical significance, and is used only to conserve computer time in testing. By choosing a large value for L3, its function is effectively negated. (See termination—II.G.)</td>
<td>99</td>
</tr>
<tr>
<td>6-10</td>
<td>M1</td>
<td>Lower Limit on length of regularities searched for in the Search Phase. This parameter is almost always set to be 2. (See section II.H. and III.B.)</td>
<td>2</td>
</tr>
<tr>
<td>11-15</td>
<td>M2</td>
<td>Upper Limit on length of regularities searched for in the Search Phase. This parameter</td>
<td>8</td>
</tr>
</tbody>
</table>
is usually a small whole number. (See section II.H. and III.B.)

Direction of Considering M's in Recoding Procedure. (See section II.C.4.) A value of +1 specifies ascending values, and a value of -1 specifies descending values. The choice of -1 is usual, especially when context-free rules are being generated (IV.D) (See RIT below).

Real Number — Coefficient of Parsimony. (See section II.D.2).

Real Number — Coefficient of Recursive Parsimony. Typically LAMBDA << LAMR. (See section II.D.2).

Specifies source of the sample. A value of 1 specifies that the sample is to be read in on cards, after the 3 control cards. A value of 2 or 3 specifies that sample is to be generated by experimental subroutines GEN2 or GEN3, respectively. CONT is
usually 1. The subroutine GENI reads the cards, if CONT = 1.

33-34 MODE Specifies the mode of the initial sample. In the first mode (MODE=1), there is no initial punctuation in the sample, while in the second mode (MODE=2), there is. (See section II.A.)

35-36 GRAM Specifies grammatical type of masks that are used in the Search Phase to search for regularities. A value of 0 specifies Unrestricted Rewrite type; a value of 1, context-sensitive. If context-free rules are desired, GRAM may be 0 or 1, provided RIT is properly set. If regular rules are desired, GRAM may be 0 or 1, provided RIT is properly set, and provided M2 and M1 are both 2. A value of 0 is most general. (See section II.B.9.)

37-38 ALLOW The worst type of regularity allowed to be used in Recoding. ALLOW may be 1 (only 100% reliable regularities are to be used); or 2 (either Type I or
Type II may be used; or 3
(Types I, II, or III may be used). It would be unusual
for ALLOW to be 4 or 5. (See section II.C.4.)

39-40  STRICT  A value of 1 specifies the use of only one value of M (namely, MBEST) in Recoding, and is suitable only when the sample is from a uniform code source. A value of 0 allows any value of M from M1 to MBEST (which is limited above by M2) in Recoding. STRICT is almost always 0. (See section II.C.4.)

41-42  FR  Radix of Masks. A value of 2 specifies binary masks, and a value of 3 specifies ternary masks (i.e. masks with "don't care" positions). (See section IV.A.)

43-44  PRI  Controls printing of unsorted table of regularities. A value of 1 specifies printing, and a value of 0 specifies no printing.

45-46  PR2  Controls printing of sorted
table of regularities. A value of 1 specifies printing, and a value of 0 specifies no printing.

47-48 PRG Controls Printing of Graph 1 or 2 of regularities used in Recordings. A value of 0 specifies the printing of no graphs; a value of 1 specifies printing of some graphs (namely, when a recoding is actually used); a value of 1 specifies printing of all graphs (i.e. for all trial recodings and the one actual recoding).

49-50 PRS Controls printing of SEQNO 0 table (table of indices of regularities). A value of 0 specifies no printing, and a value of 1 specifies printing.

51-52 EXR A value of 1 specifies that 1 alphabet-enlarging Recoding as contained in subroutine RECODE is to be used. 2 specifies Huffman type of Recoding is to be used. (See section IV.B.)
53-54  MCH  If MCH is not-zero, this value is taken as an override value of MBEST for purposes of Recoding. This parameter is used only to force a value for the "best M" for Recoding, for testing purposes.

55-56  EARLY  A value of 1 specifies that early culling of the table of regularities is to be done (to conserve memory space in the computer), and a value of 0 specifies that it is not. This parameter has no theoretical significance. (See section II.B.9.)

57-58  LAWL  This variable is no longer used by the program.

59-60  VSTART  A value of 1 specifies that the scan in the Recoding Phase is to start at position MBEST (rather than M1) of the sample (or if MODE=2, of each individual sentence). (See section II.I. and
III.C. for discussion of effect of VSTART.)

Selects method of computing $H$ (the combined entropy-parsimony-recursive parsimony measure):

<table>
<thead>
<tr>
<th>HSEL</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-\sum_{i,j} \log_2 P_{ij} + N_q \lambda + N_r \lambda$</td>
</tr>
<tr>
<td>2</td>
<td>$-\sum_{i,j} \log_2 P_{ij} + N_q \lambda + N_r \lambda$</td>
</tr>
<tr>
<td>3</td>
<td>$-\sum_{i,j} \log_2 P_{ij} + N_q \lambda + N_r \lambda$</td>
</tr>
<tr>
<td>4</td>
<td>$-\sum_{i,j} \log_2 P_{ij} + N_q \lambda + N_r \lambda$</td>
</tr>
</tbody>
</table>

where $n_q$ is the number of rules of production, where $n_r$ is the number of recursive rules, where NTM is the number of times the rules are actually applied in the transformation, where MBE is the current $M$ being used (as a maximum $M$ for any rules in this transformation). The usual value of HSEL is 1. The other expres-
63-64 WDF Controls effect of discovering a Resolving Transformation. A value of 1 specifies that if a value of M results in a resolving transformation, then this M is immediately used for the actual recoding at this level. A value of 0 allows the continued examination of M (and thereby allows another equally good or possibly even better M to be used. Note that variable WDIR controls the order of considering the M's (and that this is usually descending order). WDF is usually 1. (See section II.C.4).

65-66 WCV Specifies Approach for inducing recursions: A value of 1 specifies the Rule-Oriented Method (See section II.E.4), and a value of 2 specifies the Sentence-Oriented ("uvwxy") Approach (See Section II.E.5). WCV is usually 1.

67-68 WXL This variable is no longer used by the program.
Specifies whether Recursions should be induced. A value of 1 specifies that recursions should be induced (if possible), while a value of 0 specifies that no attempt to induce recursions should be made. (See section II.E.1).
The second control card contains the terminal alphabet and the initial punctuation mark (if any). The variable CI specifying the size of the terminal alphabet appears in card columns 1-5. The CI symbols of the terminal alphabet appear starting in column 6. If MODE = 2, the initial punctuation mark (customarily, the period) appears after the CI symbols. If the sample contains blanks, it is customary to use the symbol "\" in place of blanks to improve readability of the analytical tables. This convention in no way identifies the blank as a distinguished symbol to the Algorithm, however.

The third control card contains the non-terminal alphabet. The variable N26 specifies the size of the non-terminal alphabet and appears in card columns 1-5. The N26 symbols starting in column 6 are the non-terminal alphabet symbols. If more than N25 non-terminal symbols are required by the Algorithm, the Algorithm uses bracketed pairs of these N26 symbols in the fashion of backus normal form. It is therefore advisable not to use brackets in either the terminal or non-terminal alphabets. It is customary to use numbers for non-terminals, if letters have been used in the non-terminal alphabet, and vice versa; or, alternately, to use upper case letters for non-terminals and lower-case letters for terminals (if a suitable output printer is available).

If the sample is to be read in from cards (and this is under the control of the variable CONT), then the fourth card is a card containing the value of N, the length of the sample, in card columns 1-5. The fifth and succeeding cards now contain the sample, 80 symbols to a card. The size of the sample includes the initial punctuation in the sample, if any.
The following is an example of input to the computer program implementing the Grammar Discovery Algorithm:

```
3 2 5 -1 0.5 0.1 1 1 2 1 4 1 2 0 1 2 1 1 0 0 1 1 4 0 1 1
26ABCDEFHIJKLMNOPQRSTUVWXYZ
272
11101101100100000111011110111101011110010110000011100110111011000111111111110100
111001101000000111110011011110111010110110110010011101101000100011101101100110
010000001110111111001101011101110111011101110111101110111011101110111011111111100111111111110110
11101101111111101001000000
```

This input would be interpreted as follows:

```
MAXIMUM NUMBER OF LEVELS TO BE TRIED            3
SMALLEST M TO BE TRIED                           2
LARGEST M TO BE TRIED                            5
DIRECTION OF CONSIDERING M IN RECODE             DESCEND
LAMBDA -- COEFFICIENT OF PARSIMONY                0.50000
LAMR -- COEFFICIENT OF RECURSIVE PARSIMONY       0.10000
SOURCE OF INITIAL SYMBOL STRING                  READ
MODE -- INITIAL FLUCTUATION MARK                 1ST
TYPE OF GRAMMAR DESIRED (MASKS TO BE TRIED)       LS-RS
RII -- CONTROLS ANTECEDENT SIDE OF RULES          CF
WRIIT  TYPE OF P(I) ALLOWED TO ENTER CODE         4
USE OF BEST M STRICTLY                           YES
FR -- RADIX OF MASK                              2
PRINT CONTROL -- PRINT UNSIGNED P(I)              NO
PRINT CONTROL -- PRINT SORTED P(I)               YES
PRINT CONTROL -- PRINT GRAPH OF P(I)-S           ALL
PRINT CONTROL -- PRINT SEQUENCE NUMBERS          YES
SUBROUTINE USED FOR RECODING                      RECODE
OVERRIDE VALUE FOR MBEST (MCH)                    0
EARLY ELIMINATION OF UNALLOWABLE P(I)             NO
LAWL                                             YES
START RECODE AT TIME OF MBEST                    YES
HSFL -- METHOD FOR CREATING H                    4
CUT-OUT ON FINDING GOOD M                        NO
METHOD OF FINDING RECUSIONS                      RULE
WXIX                                             NO
TEST FOR RECURSION                               YES
SIZE OF TERMINAL ALPHABET (INITIAL STRING)       2
TERMINAL ALPHABET:                               01
SIZE OF BASIC NON-TERMINAL ALPHABET:             26
SYMBOLS OF THE BASIC NON-TERMINAL ALPHABET:      ABCDEFGHIJKLMNOPQRSTUVWXYZ
```
APPENDIX B: LISTING OF THE COMPUTER PROGRAM IMPLEMENTING THE GRAMMAR DISCOVERY ALGORITHM

C
C GRAMMAR DISCOVERY PROGRAM
C IMPLICIT INTEGER ( A-Z)
C REAL SCALARs
REAL CRIT
REAL CX
REAL FPS
REAL FC
REAL HBEST
REAL HP
REAL LAMBDA
REAL LAMR
REAL SEPAR
REAL TT
C REAL FUNCTIONS
REAL ALUG
C VECTORS OF LENGTH NS
DIMENSION Y(500)
DIMENSION ALPHA(500)
DIMENSION YNEW(500)
DIMENSION SECU(500)
INTEGER DUNE(500)
C VECTORS OF LENGTH MSIZE
REAL P(500)
REAL PO(500)
DIMENSION SQTXT(500)
DIMENSION SCLEN(500)
DIMENSION SCL 교(500)
DIMENSION TYPE(500)
DIMENSION INDEX(500)
DIMENSION INDP(500)
DIMENSION TEPHI(500)
DIMENSION TNEXT(500)
DIMENSION TBNT(500)
C VECTORS OF LENGTH M2MAX
REAL H(20)
DIMENSION MASK(20)
DIMENSION PRE(20)
DIMENSION PST(20)
REAL FIRST(20)
REAL SECOND(20)
REAL THIRD(20)
INTEGER BI(20)
INTEGER BII(20)
C VECTORS OF LENGTH N40
DIMENSION SCKJ(40)
DIMENSION SCKJAL(40)
DIMENSION SCR(40)
DIMENSION SCRCEAL(40)
DIMENSION SCRMAIL(40)

INTEGER LLIST(50)
INTEGER RPALLEN(50)
INTEGER RCLEN(50)
INTEGER RLLEV(50)
INTEGER RPRIX(50)
INTEGER RTIX(50)
REAL POX(50)
INTEGER DESC(50)

ARRAYS OF SIZE RPMAK * M2MAK
INTEGER RPA(50,20)
INTEGER RPC(50,20)
INTEGER RPAM(50,20)

VECTORS OF LENGTH YRPMAK
INTEGER YRP(300)

VECTORS OF LENGTH MNDC
REAL HOJ(500)
INTEGER ILLLEN(500)

OTHER VECTORS
DIMENSION NUMT(5)
DIMENSION LETTER(75)
DIMENSION CHAR(78)
DIMENSION MARKER(3)
DATA MARKER / 'A', ' ', ' ', ' ' /
DIMENSION BLANKS(5)
DATA BLANKS / ' ', ' ', ' ', ' ', ' ' /
DIMENSION ARCHN(12)
DATA ARCHN / ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' /
DATA SSS / 'S' /
REAL NAM(5)
DATA NAME/'URS', 'CSG', 'LS-RS', 'CFG', 'RG' /
INTEGER SOURCE(3)
DATA SOURC1/'READ', 'GEN2', 'GEN3' /
REAL WAYS(2)
DATA WAYS/'RECODE', 'HUFFMAN' /
REAL TTQ(2)
DATA TTQ / 'RULE', 'UVWXY' /
INTEGER TRI(3)
DATA TR1/'NCNE', 'SOME', 'ALL' /
INTEGER ANSW(2)
DATA ANSW/'NO', 'YES' /
INTEGER UNO(2)
DATA UNO/'1ST', '2ND' /
REAL DIRECT(3)
DATA DIRECT /'DESCEND', ' ', 'ASCEND' /
REAL ANTE(2)
DATA ANTE/'CONTEXT', 'CF' /

COMMON STORAGE
COMMON COMMCN, C1, CONT, L3, M1, M2, PR1, PR2, FR, LAMBDAM, RIT,
VECTORS AND SCALARS...

ALLOW=THE WORST TYPE OF P(I) ALLOWED IN THE RECODING
ALPHA(T)=ALPHABETIC VERSION OF Y
ANSW=ALPHABETIC VECTOR FOR YES-NO PRINTING
ANTE=ALPHABETIC VECTOR FOR PRINTING ANTECEDENT TYPES
ARROW=ALPHABETIC ARROW
BI1 BI2=VECTORS FOR TESTING ISOMORPHISM OF RULES
BLANKS=ALPHABETIC BLANKS
C = # OF DIFFERENT SYMBOLS IN SYMBOL STRING Y AT THIS LEVEL
     INCLUDES THE SYMBOLS OF MARKER
     EXCLUDES INITIAL PUNCTUATION MARK (.), IF ANY
     THIS INCLUDES THE TERMINALS AND THE NON-TERMINALS
C1= # OF DIFFERENT TERMINAL SYMBOLS IN INITIAL STRING Y
     EXCLUDES THE SYMBOLS OF MARKER
     EXCLUDES INITIAL PUNCTUATION MARK (.), IF ANY
C2= # OF TERMINAL AND NON-TERMINAL SYMBOLS IN CURRENT STRING
     EXCLUDES THE SYMBOLS OF MARKER
     EXCLUDES INITIAL PUNCTUATION MARK (.), IF ANY
CX= REAL Cx
CHAR=VECTOR OF NCHAK ALPHABETIC CHARACTERS
CCNT= CONTROLS METHOD OF PRODUCING INITIAL STRING
     1=USE GEN1 2=USE GEN2 3=USE GEN3
CRIT=THRESHOLD CRITERION FOR H
DESC=TYPES OF P(I)'S BEING GRAPIED BY 'GRAPH'
DIRECT= ALPHABETIC VECTOR FOR PRINTING DIRECTIONS
DONE(I): 0=POSITION I OF Y IS NOT YET RECODED 1=IS DONE
EARLY= COUNTS TIME OF CULLING OUT SMALL P(I)'S
     1=EARLY CULLING 0=NO EARLY CULLING
EPS= EPSILON USED TO SEPARATE TYPE 2 AND TYPE 3 P(I)'S
FG=REAL CONSTANT TO CONVERT TO BASE 2 LOGARITHMS
FIRST(I)=FIRST TERM OF COMBINED H MEASURE
FR=KADIOX OF MASKS 2=BINARY 3=TERNARY
GRAM=TYPE OF GRAMMAR DESIRED
     0=UNRESTRICTED RERITE
     1=CONTEXT SENSITIVE
     2=LEFT-SENSITIVE AND RIGHT-SENSITIVE
     3=IF CONTEXT-FREE RULES ARE DESIRED, SET RIT=1
     IN THIS EVENT, GRAM CONTROLS TYPE OF MASKS TO BE TRIED
H(M)=COMBINED ENTROPY AND PARSIMONY MEASURE
HBEST= BEST VALUE OF H FOR THIS LEVEL
HP=REAL CONSTANT FOR CUMULATING H MEASURE
HDJ(I)=THE ENTROPY OF THE I-TH DIFFERENT CONTEXT
     (SUMMED OVER THE DIFFERENT PREDICTED PARTS)
ILCLn(I)=NUMBER OF DIFFERENT PREDICTED PARTS ASSOCIATED
     WITH CONTEXT I IN ILC ARRAY
INDEX=INDEX VECTOR FOR SORTING PD INTO DESCENDING ORDER
INDEX=INDEX VECTOR FOR SORTING P INTO DESCENDING ORDER
KAPPA= 1=CHECK FOR RECURSION 0=DON'T CHECK
L3=NUMBER OF LEVELS TO BE TRIED
LAMDA=Coefficient of PARSIMONY
PO(I)=COPY OF P WHICH IS SORTED INTO DESCENDING ORDER
ALSO USED TO SORT P(I)'S THAT ARE USED IN RECODING
PCX(I): CONDITIONAL PROBABILITY OF SEQUENCE PAIR THAT
PRODUCED THIS RULE OF PRODUCTION
PR1=PRINT CONTROL FOR UNSORTED P(I)'S 0=DON'T PRINT 1=PRINT
PR2=PRINT CONTROL FOR SORTED P(I)'S 0=DON'T PRINT 1=PRINT
PRG=PRINT CONTROL FOR GRAPH OF P(I)'S USED IN RECODING
0=DON'T PRINT GRAPH
1=PRINT SOME GRAPHS ONLY WHEN TRIAL=2
2=PRINT ALL GRAPHS
PRS=CONTROLS PRINTING OF SEQUO 0=DON'T PRINT 1=PRINT
PRE(I)=M-VECTOR OF CONTEXT PART OF SEQUENCE
POST(I)=M-VECTOR OF PREDICTED PART OF SEQUENCE
PDX(I)=PROBABILITY ASSOCIATED WITH RULE OF PRODUCTION 1
RIT=CONTROLS FORM OF ANTECEDENT (LEFT) SIDE OF
RULE OF PRODUCTION
1=(CONTEXT-FREE)= REPLACE ENTIRE PREDICTOR AND PREDICTED
SEQUENCES WITH ONE NEW NCN-TERMINAL SYMBOL
0=REPLACE ALL CONTIGUOUS PREDICTOR SYMBOLS WITH
NEW NCN-TERMINAL SYMBOL
RPAL(I,J)=J-TH SYMBOL OF ANTECEDENT SIDE OF I-TH RULE OF PROD
RPC(I,J)=J-TH SYMBOL OF CONSEQUENT SIDE OF I-TH RULE OF PROD
RPALEN(I)=LENGTH OF ANTECEDENT SIDE OF I-TH RULE OF PRODUCTION
RPCLEN(I)=LENGTH OF CONSEQUENT SIDE OF I-TH RULE OF PROD
RPLEVEL(I)=LEVEL OF RULE I
RPAM(I,J)=RECORD OF PREDICTED PARTS FOR RULE I
RPMA=MAXIMUM MAX # OF RULES OF PRODUCTION IN RPA AND RPC
RPXIX(I): STATUS OF RULE I IN RPA AND RPC TABLES
0=RULE I HAS BEEN MADE INTO A RECURSIVE RULE AND DEACTIVATED
-2=NON-RECURSIVE RULE
POSITIVE # =RECURSIVE RULE. THE NUMBER POINTS TO THE
ORIGINAL NON-RECURSIVE RULE FROM WHICH IT WAS
DERIVED
RTIX(I)=STORAGE FOR RPIX OUTSIDE OF RECODE SUBROUTINE
STRUCTURE: RPA(I,J), RPC(I,J), RPAL(I), RPCLEN(I),
RPLEVEL(I), RTIX(I), RPIX(I), POX(I)
SRCD, SCRJ=SCRATCH VECTORS
SCRDAL, SCRJAL, SCRPMAL=ALPHABETIC SCRATCH VECTORS
SECOND(I)=SECOND TERM OF COMBINED H MEASURE
SEP=VALUE SEPARATING TYPE 3 AND TYPE 4 P(I)'S
SEND(I)=INDEX OF ENTRY IN SGCTX-TABLE CONTAINING PHI OF
LONGEST STRUCTURAL SEQUENCE BEGINNING AT TIME I
SOURCE=ALPHABETIC VECTOR OF SOURCES OF INPUT SAMPLE
SGCTX(I)=PHI OF THE CONTEXT PART OF THE SEQUENCE
SUPRED(I)=PHI OF THE PREDICTED SEQUENCE
SULEN(I)=LENGTH OF STRUCTURAL SEQUENCE I
SSSL=ALPHABETIC I'S
STRICT=CONTROLS VARIETY OF M'S ALLOWED IN RECODING
O=USE M...MBEST 1=USE MBEST ONLY
T IS TIME
TBCNT(I)=TEMPORARY COUNT OF OCCURRENCES OF SUB-SEQUENCES
TBPH(I)=TEMPORARY CONTEXT FOR SUB-SEQUENCES
TBTEXT(I)=TEMPORARY PREDICTED PART FOR SEQUENCES
THIRD(I)=THIRD TERM OF COMBINED H MEASURE
TT=# OF CCCURRENCES OF SEQUENCES PHI (REAL)
REPRESENTING PARTICULAR CONTEXT
TR=ALPHABETIC VECTORS FOR PRINTING
TIG=ALPHABETIC VECTOR OF WAYS OF INDUCING RECURSIONS
TYPE(I)=THE TYPE OF SEQUENCE I (0, 2, 3, 4, 5)
TYPE I (STRUCTURAL) P(I) EXACTLY 1.0
TYPE 2 (MESSAGE) P(I) WITHIN EPS OF 1.0
TYPE 3 (MESSAGE) P(I) BETWEEN 1.0 - EPS AND SEPAR
TYPE 4 (MESSAGE) P(I) BETWEEN SEPAR AND EPS
TYPE 5 (NOISE) P(I) BETWEEN EPS AND 0.0

VDIR: CCNTRUSL ASCENDING OR DESCENDING DIRECTION OF
CONSIDERATION OF M*S IN RECODE -1=DESCENDING +1=ASCENDING
VSTART: BEGIN SCAN IN RECODE AT TIME OF MBEST FOR EACH SENT
FOR (CODE=2, ONLY) O=NO 1=YES
WAYS=ALPHABETIC VECTOR OF CODING SUBROUTINES
WCV=1=USE DEFINITIONAL APPROACH TO FIND RECURSIONS
2=USE UVNX APPROACH
WF: EFFECT OF SATISFYING CRITERION OR FINDING COMPLETE
RESOLUTION FOR A PARTICULAR M
O=NO EFFECT 1=CUT-OUT AND USE THIS M
WX: INCLUDE IDENTITY RULES (AND FACTOR OF M IN H)
O=NO 1=YES
Y(I)= LARGE SAMPLE OF CONCATENATED SENTENCES
YNWT(I)= NEW Y STRING CREATED BY RECODING
YP(R)=STRING CONTAINING RULES OF PRODUCTION IN CONTIGUOUS
FORM, CONTAINS YARRAW BETWEEN ANTECEDENT AND CONSEQUENT
SIDES, CONTAINS YRPUNC BETWEEN RULES.
YRMPX=DIMENSIONED MAX # OF SYMBOLS IN YRP

SUBROUTINES ....

BEST FINDS BEST H
CCNV CONVERTS A NATURAL NUMBER INTO A C-ARY SEQUENCE
FZERO=ZEROES FLOATING POINT VECTOR
GEND READS INITIAL SYMBOL STRING FROM CARDS
GENZ IS SPECIAL UNARY SEQUENCE GENERATOR
GETMCK GENERATES MASKS
GRAPH PRINTS GRAPH OF P(I)'S USED IN RECODING
GRID PRINTS OUT MASKS
HUFF IS HUFFMAN CODE SUBROUTINE
IEQUAL=SETS VECTORS EQUAL
IMATCH=MATCHES INTEGER VALUES
ISB=DOES MATCHING
ISUM FINDS SUM OF VECTOR
IXSPKX=SPLAYS INTEGER CONSTANT
LCC LOCATES SEQUENCES THAT OCCUR IN SYMBOL STRING
LOOK KEEPS LIST OF NEW NON-TERMINAL SYMBOLS AT EACH LEVEL
MCRE CREATES MASK FROM TWO SEQUENCES (PRE AND POST)
MTES TESTS GRAMMATICAL TYPE OF MASK
PHI CONVERTS A C-ARY SEQUENCE INTO THE NATURAL NUMBER
RAND IS RANUCM INTEGER GENERATOR
RECODE DOES RECODING FROM ONE STRING TO STRING OF HIGHER LEVEL
SPRAY SETS VECTOR TO CONSTANT
SYMPOL PRODUCES SYMBOL STRING FOR PRINTING
TLOC DETERMINES WHICH SEQUENCES APPEAR AT A GIVEN TIME IN Y
XFSNWT SORTS BLOCK OF REALS WITHIN ARRAY
ZERO SETS VECTOR TO ZEROS

INPUT AND OUTPUT ...
5=INPUT CARDS
6=OUTPUT PRINTER
7=OUTPUT FILE CONTAINING SUMMARY OF PRINTER OUTPUT
DIMENSIONED SIZES OF VECTORS

NS=500  
M2MAX=20  
MTSIZE=500  
N2=26  
N40=40  
RPMAX=50  
YRPMAX=300  
MNDC=500  
MSG=15

MAIN LOOP

110 CONTINUE  
WRITE(6,790)  
790 FORMAT (1*)

READ CONTROL PARAMETERS

INPUT CARDS...

1. CONTROL CARD:
   415: L3, M1, M2, VDIR
   2F5.2: LAMBDAA, LAMR
   2512: CONT, MODE, GRAM, KIT, ALLOW, STRICT, FR, PR1, PR2, PGR, PRS,
        EXK, MCH, EARLY, LALN, VSTART, HSEL, WOF, WCV, WXL,
        KRECUR

2. TERMINAL ALPHABET VECTOR (FORMAT 15, 75A1)
   C1=SIZE OF NON-TERMINAL ALPHABET
   CARD COLUMNS 6... CONTAIN THE C1 TERMINAL SYMBOLS
   IF MODE=2, LAST COLUMN AFTER THE TERMINALS CONTAINS
   THE INITIAL PUNCTUATION MARK (.)

3. NON-TERMINAL ALPHABET VECTOR (FORMAT 15, 75A1)
   N2=SIZE OF VECTOR IN FORMAT 15
   LETTER=ALPHABET IN FORMAT 75A1

4. IF CONT=1, GEN1 READS CARD CONTAINING LENGTH OF Y (FORMAT 15)
   IF CONT=2, GEN2 READS CARD (FORMAT 1615)

5. IF CONT=1, GEN1 READS IN CARDS UNTIL N SYMBOLS HAVE BEEN
   READ IN UNDER AN 80A1 FORMAT

READ (5,401C) L3, M1, M2, VDIR, LAMBDAA, LAMR, CGNT, MODE, GRAM, RIT, ALLOW, S

1TRICT, FR, PR1, PR2, PGR, PRS, EXK, MCH, EARLY, LALN, VSTART, HSEL, WOF, WCV, WX

2L, KRECUR

4010 FORMAT (4I5,2F5.2,2512)

6. WRITE (6,4020)

7. WRITE (7,4020)

4020 FORMAT (1*)

WRITE (6,4030) L3, M1, M2, VDIR, LAMBDAA, LAMR, CONT, MODE, GRAM, RIT, ALLOW,

1STRICT, FR, PR1, PR2, PGR, PRS, EXK, MCH, EARLY, LALN, VSTART, HSEL, WOF, WCV, W
2XL, KRECUR

4030 FORMAT (1X, 415, 2FS.2, 2512)
C
C
WRITE (6, 4040) L3, M1, M2, DIRECT(VDIR+2), LAMBDA, LAMR
WRITE (7, 4040) L3, M1, M2, DIRECT(VCIR+2), LAMBDA, LAMR

4040 FORMAT ('0')
8  ** MAXIMUM NUMBER OF LEVELS TO BE TRIED     **, I7 7
4  ** SMALLEST M TO BE TRIED     **, I7 /
5  ** LARGEST M TO BE TRIED     **, I7 /
8  ** DIRECTION OF CONSIDERNG M IN RECODING     **, 6X, A0 /
9  ** LAMBDA -- COEFFICIENT OF PARSIMONY     **, F10.5 /
9  ** LAMR -- COEFFICIENT OF RECURSIVE PARSIMONY     **, F10.5 /
)
WRITE (6, 4050) SOURCE(CONT), ORI(MODE), NAME(GRAM+1), ANTS((RTT+1), ALL
10W, ANSW(STRIC+1), FR, ANSW(PR1+1), ANSW(PR2+1), TRIGR+1), ANSW(PSR+1
2W, WAYS(EWR), MCH, ANSW(EARLY+1), ANSW(LAWL+1), ANSW(VSTART+1), HSEL
WRITE (7, 4050) SOURCE(CONT), ORI(MODE), NAME(GRAM+1), ANTS((RTT+1), ALL
10W, ANSW(STRIC+1), FR, ANSW(PR1+1), ANSW(PR2+1), TRIGR+1), ANSW(PSR+1
2W, WAYS(EWR), MCH, ANSW(EARLY+1), ANSW(LAWL+1), ANSW(VSTART+1), HSEL

4050 FORMAT (1)
2  ** SOURCE OF INITIAL SYMBOL STRING     **, 6X, A4 /
3  ** MODE -- INITIAL PUNCTUATION MARK     **, 6X, A4 /
2  ** TYPE OF GRAMMAR DESIRED (MASKS TO BE TRIED)     **, 6X, A8 /
1  ** RIT -- CONTROLS ANTECEDENT SIDE OF RULES     **, 6X, A8 /
1  ** WURST TYPE OF P(I) ALLOWED TO ENTER CODE     **, 17 /
8  ** USE OF BEST M STRICTLY     **, 6X, A4 /
2  ** FR -- RACIX OF MASK     **, 17 /
6  ** PRINT CONTROL -- PRINT UNASCERTED P(I)     **, 6X, A4 /
1  ** PRINT CONTROL -- PRINT SORTED P(I)     **, 6X, A4 /
1  ** PRINT CONTROL -- PRINT GRAPGH OF P(I)-S     **, 6X, A4 /
5  ** PRINT CONTROL -- PRINT SEQUENCE NUMBERS     **, 6X, A4 /
2  ** SUBROUTINE USED FOR RECODING     **, 6X, A8 /
2  ** OVERRIDE VALUE FOR MBEST MBEST (MCH)     **, 17 /
1  ** EARLY ELIMINATION OF UNALLOWABLE P(I)     **, 6X, A4 /
7  ** INCLUDE ALL-PREDICTED MASK     **, 6X, A4 /
5  ** START RECOD AT TIME OF MBEST     **, 6X, A4 /
7  ** HSEL -- METHOD FOR COMPUTING H     **, 17 /
8  )
WRITE (6, 4060) ANSW(WDF+1), TTQ(KCV), ANSW(WXL+1), ANSW(KRECUR+1)
WRITE (7, 4060) ANSW(WDF+1), TTQ(KCV), ANSW(WXL+1), ANSW(KRECUR+1)

4060 FORMAT (1)
1  ** CUT-OUT CN FINDING GOOD M     **, 6X, A4 /
2  ** METHOD CF FINDING RECURSIONS     **, 6X, A8 /
3  ** INCLUDE IDENTITY RULES     **, 6X, A4 /
4  ** TEST FOR RECURSIN     **, 6X, A4 /
5  )
C
C
READ IN TERMINAL ALPHABET, INITIAL PUNCTUATION MARK (IF ANY),
C
READ (5, 407C) CI, [CHAR(I)], I=4, 78)

4070 FORMAT (15, 75A1)
NCHAR=C+3+(MODE=1)
C
C
PRINT OUT TERMINAL ALPHABET
NCX=NCHAR-(MODE=1)
WHITE (6, 4080) CI, [CHAR(I)], I=4, NCX)
WRITE (7, 4080) CI, [CHAR(I)], I=4, NCX)
4080 FORMAT (1)}
1 ' SIZE OF TERMINAL ALPHABET (INITIAL STRING) ', 17 /
2 ' TERMINAL ALPHABET: ', 75A1
4 )
IF (MODE.EQ.2) WRITE (6,4090) CHAR(NCHAR)
IF (MODE.EQ.2) WRITE (7,4090) CHAR(NCHAR)
4090 FORMAT ( ' INITIAL PUNCTUATION MARK (MODE=2) ',A1)
C
C
C
READ IN NCN-TERMINAL ALPHABET
C
READ (5,407C) N26,(LETTER(I),I=1,N26)
C
PRINT OUT BASIC NCN-TERMINAL ALPHABET
C
WRITE (6,410U) N26,(LETTER(I),I=1,N26)
WRITE (7,410U) N26,(LETTER(I),I=1,N26)
4100 FORMAT ( ' ',I7'/
1 ' SIZE OF BASIC NON-TERMINAL ALPHABET ', 17 /
2 ' SYMBOLS OF THE BASIC NON-TERMINAL ALPHABET: ', 75A1 )
C
C
ESTABLISH CONVENTIONS FOR REPRESENTING SYMBOLS
C
C
SYMBOLIC NUMERIC
C FORM FORM
( ALPHA ) ( Y )
C
# C DON'T CARE SYMBOL
1 1 CONTEXT PART
2 2 PREDICTED PART
C 3...C1+2 THE C1 TERMINAL SYMBOLS
C C1+3 THE INITIAL PUNCTUATION MARK, IF ANY
C
DO 115 I=1, 3
CHAR(I)=MARKER(I)
115 CONTINUE
C=NCFAK
DONT=0
CTX=1
PERC=2
PUNC=NCHAR-1
YARR=DON'T
YRPUNC=PERC
C
C
C
OBTAIN INITIAL Y FOR LEVEL 1
C
C
GO TO (130,146,150), CCNT
C
C GEN1: READ SYMBOL STRING FROM CARDS
130 CONTINUE
CALL GEN1 (Y,N,NS,ALPHA,CHAR,NCHAR)
GO TO 140
C
C GEN2: SPECIAL BINARY SEQUENCE GENERATOR
140 CONTINUE
C CALL GEN2 ( STRING,N,NS,MN,MODE,NCHAR,PUNC)
GO TO 180
C
C GEN3: GENERAL SEQUENCE GENERATOR
150 CONTINUE
GO TO 180
CONTINUE

INITIALIZE FOR ALL LEVELS

LEVEL=1
CATCH=1
NRP=0
NTP=0
NY=0

CONSIDER VARIOUS VALUES OF LEVEL=1,...,L3

CONTINUE

INITIALIZE FOR THIS LEVEL

MINM=0
NST=0
NZ=0
NDC=0
DO 192 I1=1, MNDC
HDJ(I1)=0.0
192 CONTINUE

PRINT OUT Y

CW=C-3-(MODE-1)
WRITE(6,200) LEVEL, N, CW
WRITE(7,200) LEVEL, N, CW

FORMAT(//1----------------------------- LEVEL*, I2, 9 , ' '  --------------------------- / )
2 ' SYMBOL STRING OF LENGTH ', I3, 3 ' AND USING ALPHABET OF SIZE ', I3 / )
CALL SYMBOL (Y,ALPHA,N, MNA,CHAR,NCHAR,LETTER,N26)
WRITE (6,220) (ALPHA(I11),I11=1,MNA)

FORMAT ('0', '/1X,60A1')
WRITE (7,220) (ALPHA(I11),I11=1,MNA)

COMPUTE CRITERION FOR THIS LEVEL

FC=ALOG(2.0)
CX=C-3-(MODE-1)
CRIT=-1.0  *ALOG(1.0/CX)/FC
WRITE (6,225) CRIT

FORMAT(///0CRITERION = ', F10.5 / )

CONSIDER VARIOUS M = M1 ... M2

M=M1

CONTINUE

WRITE (6,240) LEVEL,M

FORMAT ('1', 'LEVEL=',I4,5X, 'M=',I4,14///)
IF ( PK1 .NE. 0 ) WRITE(6,544)

INITIALIZE FOR THIS M

NTB=0
H(M)=9999
NSTRU=0
NEPS=0
NT3=0
NT4=0
NNOISY=0
CALL ZEROS(TBCNT,NS)
CALL SPARAY(TENEXT,NS,-1)
CALL SPARAY(TGPHI,NS,-1)

C
C
EPS AND SEPAR
SEPAR=0.50
EPS=1.0/M
IF(EPS.GE.34) EPS=.34

C
C
CCONSIDER VARIOUS MASKS
C
CONVENTIONS FOR MASK, PRE, POST VECTORS
# = DON'T CARE SYMBOL  _ = CONTEXT PART  % = PREDICTED PART
C
MASK:  # _ %
PRE:    Y Y %
POST:   # _ Y
C
NMASK=0
SIND=0
262 CONTINUE
CALL GETMSK(Y, FR, GRAM, NMASK, CTXT, PERC, DONT, M2MAX,
1  MASK, IPART, 6421)
263 CONTINUE
C
C
CONSIDER ALL T FROM T=M TO T=N
FOR CURRENT VALUE OF M, CONSIDER M-SEQUENCES STARTING AT
TIME T=T-M+1 TO TIME T=T
C
DO 400 T=M,N
C
FORM PKE AND POST SEQUENCES
C
DO 350 L=1,M
JD=L
TMJD=T-M+JD
K=M+1-L
C
IF M-SEQUENCE CONTAINS PUNCTUATION MARK, SKIP OVER IT
IF (MODE.EQ.2.AND.Y(TMJD).EQ.PUNC) GO TO 390
IF ( MASK(K).EQ. DCNT ) PRE(L)=Y(TMJD)
IF ( MASK(K).EQ. DCNT ) POST(L)=DCNT
IF ( MASK(K).EQ. CTXT ) PRE(L)=Y(TMJD)
IF ( MASK(K).EQ. CTXT ) PCST(L)=CTXT
IF ( MASK(K).EQ. PERC ) PRE(L)=PERC
IF ( MASK(K).EQ. PERC ) POST(L)=Y(TMJD)
350 CONTINUE
C
GET PHI'S FOR EACH SEQUENCE
CALL PHI(PRE,M,C,XPKE)
CALL PHI(PCST,M,C,XPPOST)
C
INSERT INTO TABLES
IF (INTB.EQ.0) GO TO 370
DO 360 I=1,NTB
IF (XPRE.NE.TEPHI(I)) GO TO 360
IF (XPOST.NE.TBNEXT(I)) GO TO 360
TBCNT(I)=TBCT(I)+1
GO TO 380
360 CONTINUE
370 CONTINUE
NTB=NTB+1
TBCT(NTB)=1
TSPHI(NTB)=XPRE
TBNEXT(NTB)=XPOST
380 CONTINUE
390 CONTINUE
C
END OF LOOP FOR THIS T
C
400 CONTINUE
C
END OF LOOP FOR THIS MASK
C
IF(SIND.EQ.1) GO TO 429
GO TO 262
C
ALSO USE THE ALL-PREDICTED MASK
C
421 CONTINUE
IF(LAWL.NE.1) GO TO 428
SIND=1
NMASK=NMASK+1
DO 422 IZ=1, M
MASK(IZ)=PERC
422 CONTINUE
GO TO 263
428 CONTINUE
429 CONTINUE
C
CCMPUTHE TPI'S
C
IF(NTB.EQ.0) GO TO 545
NZ=0
C
MAIN LOOP FOR I=1, NTB
C
DO 540 I=1,NTB
IF (TBCT(I).LE.0) GO TO 540
C
NEW AND DIFFERENT CONTEXT DISCOVERED
CCMPUTE TT-COUNT OF ITS OCCURRENCES
C
NDC=NDC+1
TT=0.0
DO 430 J=1,NTB
C
CCONSIDEERTABLE ENTRIES ONLY ONCE
IF (TBCT(J).LE.0) GO TO 430
C
MATCH CONTEXTS
IF (TBPHI(I) .NE. TBPHI(J)) GO TO 430

CONTINUE

PREVENT FURTHER CONSIDERATION OF PREDICTED PARTS (OTHER THAN FIRST) ASSOCIATED WITH THIS CONTEXT

TBCNT(J) = TBCNT(J)

CONTINUE

COMPUTE THE SQCTXT, SQPRED, SQLEN, ETC. VALUES FOR THIS PARTICULAR CONTEXT. THAT IS, CONSIDER EACH PREDICTED PART ASSOCIATED WITH THIS CONTEXT.

ILCLEN(NDC) = 0
DO 520 J = 1, NTB
CONSIDER TABLE ENTRIES ONLY ONCE
IF (TBCNT(J) .GE. 0) GO TO 520

MATCH CONTEXT
IF (TBPHI(I) .NE. TBPHI(J)) GO TO 520
NST = NST + 1
NZ = NZ + 1
P(NST) = TBCNT(J)/TT
H0J(NDC) = H0J(NDC) - P(NST) * ALOG(P(NST))
TBCNT(J) = 0
ILCLEN(NDC) = ILCLEN(NDC) + 1
SQLEN(NST) = M
SQPRED(NST) = T6NEXT(J)
SQCTXT(NST) = TBPHI(J)

CLASSIFY SEQUENCES AS TO TYPE (1, 2, 3, 4, OR 5)

IF (P(NST) .GE. 1.0) GO TO 440
IF (P(NST) .GE. 1.0 - EPS) GO TO 450
IF (P(NST) .GE. SEPARK) GO TO 460
IF (P(NST) .GE. EPS) GO TO 470
IF (P(NST) .GE. 0.0) GO TO 480

TYPE 1: P(I) EXACTLY 1.0 (STRUCTURE)
CONTINUE
NSTRU = NSTRU + 1
TYPE(NST) = 1
GO TO 490

TYPE 2: P(I) WITHIN EPS OF 1.0
CONTINUE
NEPS = NEPS + 1
TYPE(NST) = 2
GO TO 490

TYPE 3: P(I) BETWEEN 1.0 - EPS AND SEPARK
CONTINUE
NT3 = NT3 + 1
TYPE(NST) = 3
GO TO 490
C TYPE 4: PI(I) BETWEEN SEPAR AND EPS
C
470 CONTINUE
NT4=NT4+1
TYPE(NST)=4
GO TO 490
C
C TYPE 5: PI(I) LESS THAN EPS (NOISE)
C
480 CONTINUE
NNOISY=NNOISY+1
TYPE(NST)=5
GO TO 490
C
C DELETE PI(I)'S HERE THAT ARE TOO SMALL
C
490 CONTINUE
C IF (.NOT.(EARLY.EQ.1.AND.TYPE(NST).GT.ALLOW)) GO TO 495
NZ=NZ-1
NST=NST-1
C
500 CONTINUE
C PRINT UNSCRIBED TABLE OF NON-ZERO ANC DEFINED P'S
C
510 IF (PIEQ.EQ.0) GO TO 510
CALL CONV (SCRJ, M, C, SQTXT(NST))
CALL SYMBOOL (SCRJ, SCRJAL, M, MM, CHAR, NCHAR, LETTER, N26)
CALL CONV (SCRJ, M, C, SQPRED(NST))
CALL SYMBOOL (SCRJ, SCRJAL, M, MM, CHAR, NCHAR, LETTER, N26)
CALL GRID ( SCRJ, SCRJ, M, DCNT, CTAT, PERC, SCMAL, MARKER )
JJ=NST
MJ=M
WRITE(0,710) JJ, LEVEL, SQLENJ, PI(JJ), TYPE(JJ),
1 (SCRJAL(JJ), JJ=1, MJ), BLANKS, (SCRJAL(KKJ), KKJ=1, MM), BLANKS,
2 (SCRJAL(JKL), JKL=1, MM ))
C
520 CONTINUE
C
C PRINT STATISTICS ON TYPES OF SEQUENCES
C
540 CONTINUE
WRITE(6,550) ASTRU, NEPS, NT3, NT4, NNOISY
C
550 FORMAT (1 ' NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) ', I7 /
6 ' NUMBER OF TYPE 2 SEQUENCES (MESSAGE) ', I7 /
6 ' NUMBER OF TYPE 3 SEQUENCES (MESSAGE) ', I7 /
6 ' NUMBER OF TYPE 4 SEQUENCES (MESSAGE) ', I7 /
3 ' NUMBER OF TYPE 5 SEQUENCES (NOISE) ', I7 )
C
570 FORMAT (1 ' NUMBER OF DIFFERENT MASKS USED ', I7 /
2 ' NUMBER OF POSSIBLE M-SEQUENCES ', I7 /
2 ' NUMBER OF POSSIBLE PI(I)', I7 /
1 ' NUMBER OF PI(I)', I7 /
3 ' NUMBER OF DIFFERENT CONTEXTS ', I7 )
C
3 'SEPARATION VALUE BETWEEN TYPES 3 AND 4', F10.5 /
4 'EPSILON FOR DEFINING TYPES 2 AND 3', F10.5 /
5 )
C  
C  SCRT THE P(I)'S INTO DESCENDING ORDER  
C  IF ( NTB.EQ.0 ) GO TO 610
DO 542 JK=1, NST
PD(JK)=P(JK)
542 CONTINUE
CALL XFSORT ( PD, MTSIZE, INDEX, 1, NST )
C  PRINT SORTED TABLE OF NON-ZERO AND DEFINED P'S
IF ( PR2 .EQ. 0 ) GO TO 548
WRITE ( 6,544 )
544 FORMAT ( I0, # LEVEL LENGTH PROG TYPE*, 7X,
1 'MASK CONTEXT PREDICTED' / '0' )
DO 546 L=1, NST
JJ=INDEX(L)
MJ=SQLEN(JJ)
CALL CONV ( SCRJ, MJ, C, SQCTX(JJ) )
CALL SYMBOL ( SCRJ, SCRJAL, MJ, MK, CHAR, NCHAR, LETTER, N26 )
CALL CONV ( SCRJ, MJ, C, SQPRED(JJ) )
CALL SYMBOL ( SCRJ, SCRJAL, MJ, MMQ, CHAR, NCHAR, LETTER, N26 )
CALL GRID ( SCRJ, SCRJ, M, DONT, CTXT, PERC, SCRMLAL, MARKER )
WRITE ( 6,710 ) JJ, LEVEL, SQLEN(JJ), P(JJ), TYPE(JJ),
1 ( SCRML(JJH), JH=1, MJ ), BLANKS, ( SCRML(KKJ), KKJ=1, MMQ ),
2 ( SCRJAL(JKL), JKL=1, MJ )
546 CONTINUE
548 CONTINUE
C  ATTEMPT RECODING, COMPUTE H FOR THIS M
C  TRIAL=1
M5=K
CALL RECODE(Y, YNEW, SECON, ALPHA, LLIST, SQCTX, SQLEN, SQPRED,
1 TYPE, P, PD, INDEX, INDEP, CHAR, SCRJ, SCRJAL, SCRJ)
2 SCRJAL, PUST, BI, BII, PRED, MASK, H, FIRST, SECOND, THIRD, LETTER,
3 RPALEN, RPLLEN, RPLEV, RFA, RPC, RPM, YRP, NUMT, ARROW, DONE, DESC,
4, PCX, RPIX, RTIX )
C  COMPUTE NSUM, NMESS, MINM  
C  IF ( ALLOW.EQ.1 ) NSUM=NSTRU
IF ( ALLOW.EQ.2 ) NSUM=NSTRU+NEPS
IF ( ALLOW.EQ.3 ) NSUM=NSTRU+NEPS+NT3
IF ( ALLOW.EQ.4 ) NSUM=NSTRU+NEPS+NT3+NT4
IF ( ALLOW.EQ.5 ) NSUM=NSTRU+NEPS+NT3+NT4+NAQISY
NMESS=NL=NCI=ID=NST=U
IF(MINH.EQ.0) GC TO 600
IF ( MINM.EQ.0 ) MINM=M
600 CONTINUE
C  CHECK FOR (1) ALL SEQUENCES STRUCTURAL SEQUENCES,
C  (2) H(M) IS 0.0, OR (3) H(M) IS LESS THAN CRITERION
IF(DF(N),E,0) GO TO 592
IF(NMT(1),E,0) .AND. NMT(2),E,0 .AND. NMT(3),E,0 .AND. NMT(4),E,0 .AND. NMT(5),E,0 ) GO TO 650
IF(F(RST,M),E,0,0) GO TO 650
IF(H(M),E,CRIT) GO TO 630

592 CONTINUE
C
C END OF LCLP FOR M
C
610 CONTINUE
M=M+1
IF(M,E,M2) GO TO 230
C
C FIND BEST M --- LOCAL MINIMA AMONG THE H(M)
C
CALL BEST(HE,M1, M2, MBEST, MIMAX, FIRST, SECCND, THIRD, M2MAX)
WRITE (6, 620) MBEST, H(MBEST)
WRITE (7, 620) MBEST, H(MBEST)

620 FORMAT (' MBEST M IS ', I4, ' WITH H OF ', F10.5)
GO TO 670
C
C CRITERION SATISFIED
C
630 CONTINUE
MBEST=M
WRITE (6, 640) MBEST, CRIT, H(MBEST)
WRITE (7, 640) MBEST, CRIT, H(MBEST)

640 FORMAT (' CM CF ', I4, ' SATISFIES CRITERION OF ', F10.5, 1 ' WITH H OF ', F10.5)
GO TO 670
C
C ALL P(I) ARE STRUCTURAL
C
650 CONTINUE
MBEST=M
WRITE (6, 660) M, LEVEL
WRITE (7, 660) M, LEVEL

660 FORMAT (' CM CF ', I4, ' GIVES COMPLETE RESOLUTION AT LEVEL', I2/)
C
C OVERRIDE VALUE OF MBEST (FOR LEVEL 1 CALY)
C
670 CONTINUE
IF (.NOT.(MCH,E,0) .AND. LEVEL,E,1)) GO TO 690
MBEST=MCH
WRITE (6, 680) MBEST
WRITE (7, 680) MBEST

680 FORMAT (' CM OF ', I4, ' IS OVERRIDE VALUE ')
690 CONTINUE
C
C PRINT THE SQ-TABLES
C
WRITE (6, 700) LEVEL

700 FORMAT (' 1REGULARITIES FOR LEVEL ', I4)
WRITE(6, 7544)
IF (NST,E,0) GO TO 730
DJ 720 L #1, NST
I=INDEX(I)
M=SQLEN(I)
CALL CONV (SCRJ,M,C,SQCTXT(I))
CALL SYMBOL (SCRJ,SCRJAL,M,MM,CHAR,NCHAR,LETTER,N26)
CALL CONV (SCRJ,M,C,SQLRED(I))
CALL SYMBOL (SCRJ,SCRJAL,M,MM,CHAR,NCHAR,LETTER,N26)
CALL GRID (SCRJ,SCRJ, M,DCNT,CTX,T,PERC,SCRJAL,MARKER)
JJ=I
MJ=M
WRITE (0,710) JJ,LEVEL,SQLEN(IJ),P(JJ),TYPE(IJ),
1 (SCRMA1(IJH),JJH=1,MJ),BLANKS,(SCRMS1(IKJ),KKJ=1,MM),BLANKS,
2 (SCRDA1(IJKL),JKL=1,MM)
710 FORMAT (318,FS2,18,7X,80A1)
720 CONTINUE
730 CONTINUE

DO RECODING WITH M=MBE (BEST M)

TRIAL=2
MBE=MBEST
CALL RECODE(Y,YNE,N,SEQNO,ALPHA,LLIST,SQCTXT,SQLEN,SQLRED,
1 TYPE,P,PD,INDEX,INDEP,CHAR,SCRJ,SCRJAL,SCRJ,
2 SCRDA1,POST,LI,BI,PKE,MASK,HI,FIRST,SECOND,THIRD,LETTER,
3 RPALLEN,RPCLLEN,RPA,RP,RPC,RPAM,YRP,NUMT,ARRW,DONE,...,DESC
4 ,PCX,RPIX,RTIX)

END CF LLCCP FOR THIS LEVEL

GO TO TC(740,760), CATCH
740 CONTINUE
LEVEL=LEVEL+1
IF (LEVEL.LE.L3) GO TO 190
WRITE (6,750)
WRITE (7,750)
750 FORMAT ('ONLY MURE LEVELS TO BE CONSIDERED',/)
CALL SYMBOL (Y,ALPHA,N,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,220) (ALPHA(I),III=1,MMA)
WRITE (7,220) (ALPHA(I),III=1,MMA)

PRINT FINAL RULE OF PRODUCTION

GO TO 760,772
760 CONTINUE
762 CONTINUE
CALL SYMBOL (Y,ALPHA,N,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,770) SSS,ARRW,(ALPHA(I),III=1,MMA)
WRITE (7,770) SSS,ARRW,(ALPHA(I),III=1,MMA)
770 FORMAT ('0'/'','129A1/'','19X,110A1/'))
GO TO 778
772 CONTINUE
L=0
DU 775 T=1,N
L=L+1
YNE(W)=Y(T)
IFY(T).NE.PUNC) GO TO 775
CALL SYMBOL (YNE,ALPHA,L,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,770) SSS,ARRW,(ALPHA(I),III=1,MMA)
WRITE(7,770) SSS, ARROW, (ALPHA(I)), III=1, MMA )
L=0
775 CONTINUE
778 CONTINUE
C
END OF PROCESSING OF THIS DATA SET
C
780 CONTINUE
GO TO 110
END
$COPY RECODE *SINK* \&-CC
SUBROUTINE RECODELY,YNEW,SEQNO,ALPHA, L1ST,SGCXTXT,SQLN,SQRED,
  1 TYPE, P,PD,INDEX,INDEP, CHAR, SCRJ,SCRJAL,SCRD,
  2 SCDAL, POST,BI,BII,PRE,MASK,H,FIRST,SECOND,THIRD, LETTER,
  3 RPAREN, RCPLEN,RPLEV,RPA,RPC,RPAM,YRF,NUMT,ARROW,DONE,DESC
  4, PCX, RPIX, RTIX )

SUBROUTINE FOR RECODING SAMPLE Y

IMPLICIT INTEGER (A-Z)

COMMON STORAGE

COMMON C1,CONT,L3,M1,M2,PR1,PR2,FR,LAMBDA,RIT,
  1 MCH,MODE,ALLOW,STRICT,EARLY,GRAM,NCHAR,
  2 N26,PRG,EXR,PRS,LAWL,VSTART,VDIR,HSEL,WDF,
  3 WCV, WX2, KRECUR, LAMR
  4, NS,NMBE,C,MTSIZE,M2MAX,NQ,TRIAL,CCNT,PUNC,LEVEL,N40,FC,NS,
  5 PERC,ICTXT,TN,NTM,RP MAX,NRP, N Y, YRP MAX, NRECUR, YRABN, YR PUNC
  1, NTP

VECTORS OF SIZE NS
INTEGER YNS(NS)
INTEGER YNEW(NS)
INTEGER SEQNC(NS)
INTEGER ALPHA(NS)
INTEGER DONE(NS)

VECTORS OF SIZE MTSIZE
INTEGER SGCXTXT(MTSIZE)
INTEGER SOLLEN(MTSIZE)
INTEGER SQRNL(MTSIZE)
INTEGER TYPO(MTSIZE)
INTEGER INDEX(MTSIZE)
INTEGER INDEP(MTSIZE)
REAL P(MTSIZE)
REAL P0(MTSIZE)

VECTORS OF SIZE NCHAR
INTEGER CHAR(NCHAR)

VECTORS OF SIZE N40
INTEGER SCRJ(N40)
INTEGER SCRJAL(N40)
INTEGER SCRDN(N40)
INTEGER SCRJAL(N40)

VECTORS OF SIZE M2MAX
INTEGER POST(M2MAX)
INTEGER B1(M2MAX)
INTEGER BII(M2MAX)
INTEGER PRE(M2MAX)
INTEGER MASK(M2MAX)
REAL H(M2MAX)
REAL FIRST(M2MAX)
REAL SECOND(M2MAX)
REAL THIRD(M2MAX)

C VECTORS OF SIZE N26
INTEGER LETTER(N26)

C VECTORS OF SIZE RMAX
INTEGER LLIST(RMAX)
INTEGER RPSET(RMAX)
INTEGER RPKLEN(RMAX)
INTEGER RPK2LEN(RMAX)
REAL PDX(RMAX)
INTEGER RP2X(RMAX)
INTEGER RPIX(RMAX)
INTEGER RTIX(RMAX)
INTEGER DESC(RMAX)

C ARRAYS OF SIZE RMAX * M2MAX
INTEGER RPA(RMAX,M2MAX)
INTEGER RFC(RMAX,M2MAX)
INTEGER RPAM(RMAX,M2MAX)

C VECTORS OF SIZE YRMAX
INTEGER YRP(YRMAX)
OTHER VECTORS
DIMENSION NUMT(5)
INTEGER ARROW(12)

C REAL CONSTANTS AND FUNCTIONS
REAL HP
REAL ALOG
REAL FC
REAL LAMBDA
REAL LAMR

C SUBROUTINE RECODES SAMPLE ACCORDING TO PARAMETERS
C MAIN PARAMETERS FOR RECODING:
C MBE=UPPER LIMIT ON LENGTH OF REGULARITY TO BE USED IN
C THIS RECODING
C M=M1,...,MBE
C TRIAL=INDICATION OF WHETHER THIS RECODING IS TRIAL "OR"
C ACTUAL RECODING 1=TRIAL 2=ACTUAL
C RIT=WHETHER TO WRITE CONTEXT-FREE RULES
C ALLOW=TYPE OF WORST P(I) ALLOWED IN RECODING
C STRICT=WHETHER TO DO STRICT RECODING
C KRECUR=WHETHER TO TRY RECURRENS
C OTHER PARAMETERS
C VDIR=DIRECTION OF SCAN FOR M
C VSTART=STARTING POINT FOR SCAN
C HSEL=SELECTION OF H MEASURE (FOR TESTING PURPOSES)

C INITIALIZATION

YNL=N
HP=0.0
NTM=0
NCT=0
NQ=0
NRECUR=0
NHER=0
FILL=DUNT
CALL ZERO (SEQNG,N)
CALL ZERO (DONE,NS)
DO 2040 I=1,M1
YNEW(I)=Y(I)
2040 CONTINUE
IF (TRIAL.EQ.2) WRITE (6,2010) LEVEL
IF (TRIAL.EQ.2) WRITE (7,2010) LEVEL
2010 FORMAT ('0-------------------------- RULES CP PRODUCTION ',
9 'FOR LEVEL', 1Z,
1 '--------------------------')
IF (TRIAL.EQ.1) WRITE (6,2020) MBE
2020 FORMAT ('0TENTATIVE RULES OF PRODUCTION FOR M OF ', 13 )
C
COPY INTO TEMPORARY RPIX FROM THE PERMANENT RTIX TABLE
NB: IF TRIAL=1, THE RPA AND OTHER ASSOCIATED TABLES ARE NOT
ALTERED, AND NP IS NOT CHANGED.
IF(NTP.NE.0) CALL IEQUAL (RPIX, NTP, RTIX)
C
C
C
C
LOOP FOR T=M1,...,N
C
LAST=0
QLAST=0
TN=M1
T=M1
2050 CONTINUE
YNEW(TN)=Y(T)
IF (NST.EQ.0) GO TO 2460
IF (VSTART.EQ.1.AND.T-QLAST.LT.MBE) GO TO 2460
C
C
C
C
LOOP FOR LR=1,2
C
FIRST TIME: TRY TO USE SEQUENCES THAT BECOME RECURSIVE
C
PROVIDED KEKCUR IS SET AT 11 ES)
C
SECOND TIME: USE ANY SEQUENCE
C
LR=1
IF (LEVEL.EQ.1) LR=2
IF (KEKUR.EQ.0) LR=2
2060 CONTINUE
C
C
C
C
LOOP FOR LTA=1, ALLOW
C
LTA=1
2070 CONTINUE
C
C
C
C
LOOP FOR M
C
M=M1,...,MBE IF VDIR=+1
C
M=M2,...,M1 IF VDIR=-1
C
M=MBE ONLY IF STRICT=1
C
IF (VDIR.EQ.1) M=M1
IF(VDIR.EQ.-1) M=MBE
IF (STRIC=EQ.1) M=MBE
2080 CONTINUE
DU 2120 LLL=1,NST
L=INDEX(LLL)
IF (SQLEN(L).NE.M) GO TO 2120
IF (TYPE(L).NE.MTA) GO TO 2120
C
MAKE SURE NO PUNCTUATION OCCURS IN M-SEQUENCE OF Y
IF (MODE.EQ.1) GO TO 2100
DO 2090 JD=1,M
TMJDT=T-M+JD
IF (Y(TMJDT).NE.PUNC) GO TO 2090
QLAST=TMJDT
IF (VDIR.EQ.-1) GO TO 2130
GO TO 2460
2090 CONTINUE
2100 CONTINUE
C
RECREATE CONTEXT-SEQUENCE AND PREDICTED-SEQUENCE
CALL CONV (PRE,M,C,SQXTXT(L))
CALL CONV (POST,M,C,SQPPRED(L))
C
MATCH Y AGAINST CONTEXT AND PREDICTED SEQUENCES
DO 2110 JD=1,M
TMJD=T-M+JD
IF (POST(JD).EQ.DONT) GO TO 2110
IF (PRE(JD).EQ.PERC.AND.POST(JD).NE.Y(TMJD)) GO TO 2120
IF (POST(JD).EQ.CTX.T.AND.PRE(JD).NE.Y(TMJD)) GO TO 2120
2110 CONTINUE
C
LOCATED SEQUENCE OF LENGTH M AT TIME T
C
GO TO 2150
2120 CONTINUE
2130 CONTINUE
C
END OF LOC FOR M
C
M=M+VDIR
IF (STRCT.EQ.1) GO TO 2140
IF (VDIR.EQ.-1.AND.M.GE.M1) GO TO 2080
IF (VDIR.EQ.1.AND.M.LE.MBE) GO TO 2080
C
2140 CONTINUE
L=0
2150 CONTINUE
SEQN(T)=L
C
THE SEQUENCE THAT IS FOUND IS
1. OF LONGEST LENGTH FROM M1 TO MBE OF THOSE SEQUENCES
2. ENDING AT TIME T (IF VDIR = -1 )
3. OF SHORTEST LENGTH (IF VDIR = +1 )
4. OF LENGTH MBE (IF STRCT = 1 )
5. OF HIGHEST ALLOWABLE TYPE
6. OF HIGHEST P(I) POSSIBLE
7. OF SIMPLEST GRAMMATICAL TYPE
C
IF (L.EQ.0) GO TO 2450
M=SQLEN(L)
IS THERE ROOM (FROM LEFT) FOR AN M-SEQUENCE
   IF (T+1-M.LE.LAST) GO TO 2450

ARE ALL M POSITIONS AS YET UNRECODED?
DO 2160 JD=1,M
   TMJD=T-M+JD
   IF (CUNY(TMJD).EQ.1) GO TO 2450
CONTINUE
2160

CONTRIBUTION TO H
   IF (P(L).EQ.0.0) GO TO 2170
   NTM=NTM+1
   IF(WXL.EQ.0) MWQ=1
   IF(XXL.EQ.1) MWQ=M
   HP=HP-P(L)*ALOG(P(L)) * MWQ
CONTINUE
2170

CHECK CURRENT L AGAINST LLIST
   QNEW: 1=L IS NEW TO LLIST     0=L IS OLD
   CALL LOOK (LLIST,MTSIZE,NG,L,ILINE,QNEW)
   LAST=T
   TN=TN-M
   KL=Q
   T=T-M

WRITE CONSEQUENT (RIGHT) SIDE OF RULE OF PRODUCTION
   CONSEQUENT SIDE IS OF LENGTH M
   IF (QNEW.NE.1) GO TO 2190
   DO 2180 JP=1,M
      SCRDP(JP)=Y(T+JP)
   CONTINUE
2180
CONTINUE
2190

WRITE ANTECEDENT SIDE OF RULE OF PRODUCTION
   RIT=CONTROLS FORM OF ANTECEDENT (LEFT) SIDE OF
   RULE OF PRODUCTION
   1=(CONTEXT-FREE)= REPLACE ENTIRE PREDICTOR AND PREDICTED
   SEQUENCES WITH ONE NEW NON-TERMINAL SYMBOL
   0=REPLACE ALL CONTIGUOUS PREDICTOR SYMBOLS WITH
   NEW NON-TERMINAL SYMBOL
   IF (RIT.EQ.0) GO TO 2210

CONTEXT-FREE CASE (RIT=1)
   REPLACE ENTIRE ANTECEDENT (LEFT) SIDE WITH 1 NEW NON-TERMINAL

   KRET=1
   IF (LR.EQ.1) GO TO 2810
CONTINUE
2200
   IF (LR.EQ.1.AND.LRX.EQ.0) GO TO 2450
   TN=TN+1
IF (LR.EQ.1 .AND. LRX.EQ.1) YNEW(TN)=R$YM
IF (LR.EQ.2) YNEW(TN)=C-1+LINE*NCT
KL=KL+1
SCRJ(KL)=YNEW(TN)
CALL IXSPRY (DONE, NS, 1, T+1, T+M)
T=T+M
IF(M.GE.2) CALL IXSPRY(YNEW, NS, FILL, TN+1, TN+M-1)
TN=TN+M-1
GO TO 2360

RIT=0 CASE

2210 CONTINUE
CALL CONV (PRE, M, C, SQCTXT(L))
CALL CONV (POST, M, C, SOPRED(L))
CALL MCRC (PRE, POST, M, MASK, DCNT, CTXT, PERC, MODE, PUNC)

C IS MASK STRICTLY CONTEXT-SENSITIVE
C OR STRICTLY UNRESTRICTED Rrewrite

CALL MTST ( MASK, M, PERC, 22305, 2215 )

***************************************************************************
STRCTLY CONTEXT-SENSITIVE CASE
***************************************************************************

FIND CONTEXT ON LEFT, IF ANY

2215 CONTINUE
LEFT=0
RIGHT=0
DO 2220 LEFT=1,M
IF (POST(LEFT).EQ.DCNT.OR.POST(LEFT).EQ.CTXT) GO TO 2230

2220 CONTINUE

2230 CONTINUE
LEFT=LEFT-1
DO 2240 RL=1,M
RLQ=M+1-KL
IF (POST(RLQ).EQ.DCNT.OR.POST(RLQ).EQ.CTXT) GO TO 2250

2240 CONTINUE

2250 CONTINUE
RIGHT=RL-1

BRANCH TO RECURSIVE TEST, IF LR=1

KRET=2
IF (LR.EQ.1) GO TO 2810

2260 CONTINUE
IF (LR.EQ.1 .AND. LRX.EQ.0) GO TO 2450
CALL IXSPRY (DONE, NS, 1, T+1, T+M)

LEFT END (IF ANY)

IF (LEFT.EQ.0) GO TO 2280
DO 2270 KT=1, LEFT
T=T+1
TN=TN+1
YNEW(TN)=Y(T)
KL=KL+1

GO TO 2230

2810 CONTINUE
J=J+1
CALL IXSPRY (DONE, NS, FILL, TN+1, TN+M-1)
TN=TN+M-1

GO TO 2210
2270 CONTINUE
2280 CONTINUE
C
C REPLACE 'MID' CONTIGUOUS SYMBOLS IN Y WITH 1 NEW NON-TERMINAL
C
TN=TN+1
IF (LR.EQ.1 .AND. LRX.EQ.1) YNEW(TN)=RSYM
IF (LR.EQ.2) YNEW(TN)=C-1+ILINE+NCT
KL=KL+1
SCRJ(KL)=YNEW(TN)
MID=M-RIGHT-LEFT
IF (MID.GE.2) CALL IXSPRY(YNEW,NS,FILL, TN+1, TN+MID-1 )
TN=TN+MID-1
T=T+MID
C
C RIGHT END (IF ANY)
IF (RIGHT.EQ.0) GO TO 2300
DO 2290 KP=1,RIGHT
T=T+1
TN=TN+1
YNEW(TN)=Y(T)
KL=KL+1
SCRJ(KL)=YNEW(TN)
2290 CONTINUE
2300 CONTINUE
GO TO 2340
C
C ***********************************************
C STRICTLY UNRESTRICTED-REWRITE CASE
C ***********************************************
C
2305 CONTINUE
C ***RPAK,RECUR,DCNE
NB=-1
KA=0
2310 CONTINUE
KA=KA+1
IF (KA.GT.N) GO TO 2350
IF (MASK(KA).EQ.NB) GO TO 2330
NB=MASK(KA)
KRET=3
IF (LR.EQ.1) GO TO 2810
2320 CONTINUE
IF (LR.EQ.1 .AND. LRX.EQ.0) GO TO 2450
TN=TN+1
IF (LR.EQ.1 .AND. LRX.EQ.1) YNEW(TN)=RSYM
IF (LR.EQ.2) YNEW(TN)=C-1+ILINE+NCT
NCT=NCT+1
KL=KL+1
SCRJ(KL)=YNEW(TN)
T=T+1
GO TO 2310
2330 CONTINUE
IF (MASK(KA).EQ.PERC) GO TO 2310
T=T+1
TN=TN+1
YNEW(TN)=Y(T)
KL=KL+1
SCRJ(KL)=YNEW(TN)
2350 CONTINUE
   GO TO 2360
C
C SAVE ANTECEDENT (LEFT) SIDE OF RULE OF PRODUCTION IN RPA
C
2360 CONTINUE
   IF (QNEW .NE. 1) GO TO 2410
   NHER=NHER+1
   NRP=NRP+1
   RPLEV(NRP)=LEVEL
   PDX(NRP)=0.0
   IF (LRX.EQ.0) PDX(NRP) = P(L)
   IF (LRX.EQ.2 .OR. LRX.EQ.1 .AND. LTRX.EQ.0) RPIX(NRP) = -2
   NB: RPIX CAN ALSO BE SET IN RECURSIVE ROUTINE
   RPALEN(NRP)=KL
   DD 2370 JG=1, KL
   RPAP(NRP,JG)=SCRJ(JG)
2370 CONTINUE
C
C SAVE CONSEQUENT (RIGHT) SIDE OF RULE OF PRODUCTION IN RPC
C
2380 CONTINUE
   IF (RIT.EQ.0) GO TO 2390
   RPAM(NRP,1)=PERC
   GO TO 2410
2390 CONTINUE
   DU 2400 JB=1, M
   RPAM(NRP,JB)=MASK(JB)
2400 CONTINUE
2410 CONTINUE
C
C PRINT RULES OF PRODUCTION ONTO 6, AS THEY ARE PRODUCED
C
C SCRJ=ANTECEDENT SIDE OF RULE OF PRODUCTION
C SCR=CONSEQUENT SIDE OF RULE OF PRODUCTION
C
IF (QNEW .NE. 1) GO TO 2430
   CALL SYMBOL (SCRJ,SCRDAL,M,MMQ,CHAR,NCHAR,LETTER,N26)
   CALL SYMUGL (SCRJ,SCRJAL,KL,MP,CHAR,NCHAR,LETTER,N26)
   WRITE (6,2420) (SCRJAL(J),J=1,MMQ),ARRROW,(SCRDAL(JJ),JJ=1,MMQ)
2420 FORMAT ('O',29A1/(1X,19X,110A1/))
2430 CONTINUE
C
C IF NEW RULE WAS FOUND, OR NEW RECURSIVE RULE WAS FORMED,
C TRY TO APPLY IT THROUGHOUT STRING Y
C
I=NRP
   MG=RPCLEN(I)
   IF (STRICT.EQ.1 .AND. MG.NE. MBE) GO TO 2450
3538 CONTINUE
   NGL=0
C
C LOOP FOR TX=M1, ...
C
LASTX=0
QLASTX=0
TNX=M1
TX=M1
3550 CONTINUE
IF (VSTART.EQ.1.AND.TX=QLASTX.LT. MBE) GO TO 3630

C
IS THERE RGCM (FFCM LEFT) FOR AN M-SEQUENCE
IF (TX+1-MG.EQ.1.LT. LASTX) GO TO 3630

C
ARE ALL M POSITIONS AS YET UNRECODED?
DO 3570 JD=1, MG
TMJD=TX-MG+JD
IF (DONE(TMJD).EQ.1) GO TO 3630
3570 CONTINUE

C
MAKE SURE NO PUNCTUATION OCCURS IN M-SEQUENCE OF Y
IF (MODE.EQ.1) GO TO 3590
DO 3580 JD=1, MG
THJD=TX-MG+JD
IF (Y(TMJD).NE.PUNC) GO TO 3580
QLASTX=TMJD
GO TO 3630
3580 CONTINUE
3590 CONTINUE

C
MATCH Y WITH CONSEQUENT (RIGHT) SIDE OF RULE I
DO 3600 JD=1, MG
IF (RPAM(I,J).EQ.DUNT) GO TO 3600
TMJD=TX-MG+JD
IF (RPC(I,J)+EQ.Y(TMJD)) GO TO 3600
IF (LRX.EQ.1.AND.(Y(TMJD).EQ.SMID .AND. RPC(I,J).EQ.RSYM))
1 GO TO 360C
GO TO 3630
3600 CONTINUE

C
LOCATED SEQUENCE OF LENGTH M AT TIME T
SEQNO(TX)=L
NGL=NGL+1
LASTX=TX

C
CONTRIBUTION TO M
IF (POXI(I).EQ.0.0) GO TO 3610
ATM=NTM+1
IF (XW.L.EQ.0.0) MKC=1
IF (XW.EQ.1.0) MWQ=M
HP=HP-POXI(I)*ALGQ(PDX(I))*MWQ
3610 CONTINUE

C
RECODE
TNX=TNX-MG
TX=TX-MG
MAA=RPALEN(I)
DO 3620 JD=1, MAA
TNX=TNX+1
YNEW(TNX)=RPA(I,JD)
3620 CONTINUE

IF (MG.GT.MAA) CALL IXSPRAY(YNEW,NS,FILL,TNX+1, TX+MG-MAA)
TNX=TNX+MG-MAA
CALL IXSPRAY(IDNE,NS,1, TX+1, TX+MG)
TX=TX+MG
YKJ(NY)=SCRD(JP)
CONTINUE
NY=NY+1
YRP(NY)=YRPUNG
CALL SYMBOL (SCRJ,SCRJAL,MA,MM,CHAR,NCHAR,LETTER,N26)
CALL SYMBOL (SCRD,SCRDAL,MC,MM,CHAR,NCHAR,LETTER,N26)
WRITE (7,2420) (SCRJAL(J),J=1,MM),ARRCW,(SCRDAL(JJ),JJ=1,Mc)
CONTINUE

COPY INTO PERMANENT RTIX THE TEMPOARY VECTOR RPIX
IF (NRP.EQ.0) GO TO 2510
NTP=NRP
CALL IEQUAL (RTIX,NRP,RPIX)
CONTINUE
CONTINUE
PRINT OUT ORIGINAL STRING Y
WRITE (6,2720)
CALL SYMBOL (Y,ALPHA,N,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,2730) (ALPHA(III),III=1,MMA)
REMOVE 'FILL' SYMBOLS FROM YNEW
CALL REMOVE (YNEW,YNEW,YNL,YNL,FILL)
PRINT OUT NEW STRING YNEW
WRITE(6,2732)
CALL SYMBOL (YNEW,ALPHA,YNL,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,2730) (ALPHA(III),III=1,MMA)

TRY TO RE-US ALL RULES DEVELOPED IN ALL POSSIBLE WAYS
IF(NRP.EQ.0) GO TO 2652
CONTINUE
NGL=0
LM=1,2
FIRST TIME: TRY TO USE SEQUENCES THAT ARE RECURSIVE
SECOND TIME: TRY ANY OTHER SEQUENCE
LM=1
IF(KRECUR.EQ.0) LM=2
CONTINUE
LOOP FOR T=M1,...
LAST=0
QLAST=0
T'=M1
T=M1
CONTINUE
IF (VTART.EQ.1.AND.T-QLAST.LT.MBE) GO TO 2631
IF (YNEW(TN).EQ. FILL ) GO TO 2631
DO 2630 I=1, NRP
M=RPCLEN(I)
IF (STRICT.EQ.1.AND. M.NE. MBE) GO TO 2630
CONTINUE
2538
2540
2550
C WHEN LM=1, PROCESS ONLY RECURSIVE RULE (RPIX > 0)
   IF (LM.EQ.1 .AND. RPIX(I) .LE. 0) GO TO 2630
C WHEN LM=2, PROCESS NON-RECURSIVE RULE (RPIX=-2)
   IF (LM.EQ.2 .AND. RPIX(I) .NE. -2) GO TO 2630
C DEACTIVATED RULES ARE SKIPPED
C IS THERE RCCM FROM LEFT) FOR AN M-SEQUENCE
   IF (T+1-M.LT-LAST) GO TO 2630
C MAKE SURE NO PUNCTUATION OCCURS IN M-SEQUENCE OF Y
   IF (MODE.EQ.1) GO TO 2590
   DO 2580 JD=1,M
      TMJD=T-M+JD
      IF (YNEW(TMJD)*NE. PUNC) GO TO 2580
      QLAST=TMJD
      GO TO 2630
   2580 CONTINUE
C 2590 CONTINUE
C MATCH Y WITH CONSEQUENT (RIGHT) SIDE OF RULE
   DO 2600 JD=1,M
      IF (RPAM(I,JD).*EQ. DCNT) GO TO 2600
      TMJD=T-M+JD
      IF (RPCI(I,JD)*NE. YNEW(TMJD)) GO TO 2630
   2600 CONTINUE
C LOCATED SEQUENCE OF LENGTH M AT TIME T
C NGL=NGL+1
C LAST=T
C CONTRIBUTION TO H
C IF (POXI(I).*EQ.0.0) GO TO 2610
   NT=M+1
   IF (WXL*EQ.0) MWQ=1
   IF (WXL*EQ.1) MWQ=M
   HP=HP-POXI(I)*ALG(POXI(I)) * MWQ
C 2610 CONTINUE
C RECODE
C TN=TN-M
C T=T-M
C MA=RPALEN(I)
C DJ 2620 JD=1,MA
C TN=TN+1
C YNEW(TN)=RP(A,I,JD)
C 2620 CONTINUE
C IF (M.GT. MA) CALL IXSPRY (YNIEW, NS, FILL, TN+1, TN+M-MA)
C TN=T+M-MA
C CALL IXSPAY (DONE,NS,1,T+1,T+M)
C T=T+M
C 2630 CONTINUE
C END CF LCCP FOR T AND TN
C 2631 CONTINUE
T=T+1
TN=TN+1
IF(T.LE. YNL ) GO TO 2550
C END OF LCCP FOR LM=1,2
C
LM=L*M+1
IF(LM.LE.2) GO TO 2540
2650 CONTINUE
C
GO THROUGH LIST OF RULE OF PRODUCTION UNTIL NO APPLICATIONS ARE POSSIBLE
C
IF (AGL.NE.0) GO TO 2538
C IF TRIAL=1, DELETE NHER RULES FROM TABLES
C
2652 CONTINUE
IF (TRIAL.EQ.1) NRP=ARP-NHER
C
IDENTITY RULES
C (APPROXIMATE COUNT)
C
NIR=0
IF(WXL.EQ.0) GO TO 2659
CZ=0
XG=1
DO 2658 IJ=1, N
IF(Y(IJ).EQ.PUNC) GO TO 2653
IF(DCNE(IJ).EQ.1) GO TO 2657
XG=0
CZ=CZ+1
IF(CZ.LT. MBE ) GO TO 2658
2653 CONTINUE
NIR=NIR+1
NTM=NTM+1
XG=1
CZ=0
GO TO 2658
2657 CONTINUE
IF(XG.EQ.0) GO TO 2653
XG=1
CZ=0
2658 CCONTINUE
2659 CCONTINUE
C
CCOMPUTE H
C
FIRST(MBE)=2499. 
GO TO (2660,2670,2680,2690), HSEL
2660 CONTINUE
IF (NTM.NE.0) FIRST(MBE)=HP/(FC*NTM)
GO TO 2700
2670 CONTINUE
IF (NTM.NE.0) FIRST(MBE)=(HP*NQ)/(FC*NTM)
GO TO 2700
2680 CONTINUE
FIRST(MBE)=(HP*MBE)/(FC*M1)
GO TO 2700
2690 CONTINUE
FIRST(MBE)=HP/FC
GJ TO 2700
2700 CONTINUE
SECOND(MBE)=NQ*LAMBDA
SECOND(MBE)=(NQ*NIR)*LAMBDA
THIRD(MBE)=LAMBDA*NRECUR
H(MBE)=FIRST(MBE)+SECOND(MBE)+THIRD(MBE)
WRITE (6,2710) FIRST(MBE),SECOND(MBE),THIRD(MBE),H(MBE),NQ,NRECUR,
1 NIR, NTH
2710 FORMAT ('0' /
2 * ENTRPETY TERM        F10.5 /
3 * PARSIMONY TERM       F10.5 /
4 * RECURSIVE PARSIMONY TERM       F10.5 /
5 * 'VALUE OF H FOR THIS REDUCTION...........F10.5 /
6 /
7 * NUMBER OF RULES OF PRODUCTION      17 /
8 * NUMBER OF RECURSIVE RULES        17 /
9 * NUMBER OF IDENTIY RULES           17 /
10 * NUMBER OF TIMES RULES ARE APPLIED 17 /
)
C PRINT OUT ORIGINAL STRING Y
C WRITE (6,2720)
2720 FORMAT ('UCURRENT STRING Y' / '0')
CALL SYMBOL (Y,ALPHA,N,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,2730) (ALPHA(III),III=1,MMA)
2730 FORMAT ('0', /IX, 60A1) )
C AGAIN REMOVE ALL 'FILL' SYMBOLS FROM YNEW
C CALL REMOVE ( YNEW, YNEW, YNL, YNL, FILL )
C PRINT OUT NEW STRING YNEW
C WRITE(6,2732)
2732 FORMAT ('ONEW STRING ' / '0')
CALL SYMBOL (YNEW,ALPHA,YNL,MMA,CHAR,NCHAR,LETTER,N26)
WRITE (6,2730) (ALPHA(III),III=1,MMA)
C PRINT OUT STRING YRP
C C PRINT GRAPH OF P(I) ACTUALLY USED IN RECODING
C IF (.NOT.(PRG.EQ.2.CEPRG.EQ.1.AND.TRIAL.EQ.2)) GO TO 2790
IF (N1.EQ.0) GO TO 2790
CALL ZERO (NUMT,9)
DO 2740 I=1,NQ
L=LIST(I)
PD(I)=P(L)
K=TYPE(L)
NUMT(K)=NUMT(K)+1
2740 CONTINUE
IF (PRG.EQ.3) GO TO 2760
CALL XFSORT (PD,MTSIZE,INDEP,1,NQ)
DO 2750 I=1,NQ
DESC(I)=TYPE(LIST(INDEP(I)))
2750 CONTINUE
CALL GRAPH (PC,NQ,NUMT,LEVEL,M6E,DESC)

CONTINUE

PRINT STATISTICS ON P(I)'S ACTUALLY USED IN RECODING

WRITE (6,2770) (NUMT(IIM),IIM=1,5)

FORMAT (
1 ' NUMBER OF TYPE 1 SEQUENCES (STRUCTURAL) ', I7 /
6 ' NUMBER OF TYPE 2 SEQUENCES (MESSAGE) ', I7 /
6 ' NUMBER OF TYPE 3 SEQUENCES (MESSAGE) ', I7 /
6 ' NUMBER OF TYPE 4 SEQUENCES (MESSAGE) ', I7 /
3 ' NUMBER OF TYPE 5 SEQUENCES (NOISE) ', I7 ) /
NZZ=ISUM(NUMT,5)
WRITE (6,2780) NZZ

FORMAT (
1 ' NUMBER OF P(I) ', I7 )

CONTINUE

IF TRIAL=2, SUBSTITUTE RECODED YNEW INTO Y
NEW SIZE OF ALPHABET
NEW LENGTH OF Y
SUBSTITUTE NEW STRING (YNEW) INTO Y VECTOR

IF (TRIAL.NE.2) GO TO 2800
C=C+NQ+NCT
YNL=YNL
CALL IEQUAL (Y,N,YNEW)

RETURN

CHECK FOR RECURSION

CONTINUE

RULE 1 IS OF PREVIOUS LEVEL

IF (RPLEV(I).NE.LEVEL-1) GO TO 2900

CONSEQUENT (RIGHT) SIDE OF RULE 1 FROM PREVIOUS LEVEL, AND
CONSEQUENT SIDE OF CURRENT RULE ARE OF SAME LENGTH
NB: THIS IS PRE-CONDITION FOR ISOCRPHISM (SEE BELOW)

IF (RPCLEN(I).NE.LCC) GO TO 2900
MA=RPCLEN(I)
DO 2890 J=1,MA

SYMBOL 'SMID' IS A 'PREDICTED' SYMBOL ON THE ANTECEDENT (LEFT)
C SIDE OF RULE I, AT PREVIOUS LEVEL
    IF (RPAM(I,J) .NE. PERC) GO TO 2890
    SMID=RPAM(I,J)
    DO 2880 J=1,LC
        C SYMBOL 'SMID' APPEARS ON CONSEQUENT (RIGHT) SIDE OF RULE AT
        THIS LEVEL
        IF (SCRD(JJ) .NE. SMID) GO TO 2880
        C ISOMORPHISM OF CONSEQUENT (RIGHT) SIDE OF RULE I FROM PREVIOUS
        LEVEL, AND CONSEQUENT (RIGHT) SIDE OF CURRENT RULE
        BILEN=0
        BIILEN=0
        DO 2830 KK=1,LC
            CALL LOOK (BI, M2MAX, BILEN, RPC(I, KK), BILEV, BIINEW)
            CALL LOOK (BII, M2MAX, BIILEN, SCRD(KK), BILEV, BIINEW)
            IF (BILEV .NE. BIILEV) GO TO 2880
        CONTINUE
        2830
        C HAVE RECURSION
        LRX=1
        NR=NR+1
        WRITE (6,2840) NR
        WRITE (6,2840) NR
        2840 FORMAT ('**RECURSION NR. ', I3)
    C IDENTIFY RECURSIVE SYMBOL
    RSYM=RPC(I, JJ)
    C ABANDON GENERATION OF CURRENT RULE
    IF (NQ .NE. 0) AQ=AQ-1
    C INSERT RSYM WHEREEVER SMID OCCURRED IN RULES
    DO 2870 IA=1, NMP
        LZ=RPALEN(IA)
        DO 2850 JA=1,LZA
            IF (RPAM(IA, JA) .EQ. SMID) RPAM(IA, JA)=RSYM
        CONTINUE
        2850
        LZ=RPALEN(IA)
        DO 2860 JA=1,LZC
            IF (RPC(IA, JA) .EQ. SMID) RPC(IA, JA)=RSYM
        CONTINUE
        2860
        CONTINUE
    2870
    C GENERATE A RECURSIVE RULE AND DEACTIVATE RULE I
    RPIX(I)=0
    RPIX(NRP+1)=I
    DO 2872 JP=1, M
        SCRD(JP)=RPC(I, JP)
    CONTINUE
    2872
    DO 2874 JG=1, MA
        SCRJ(JG)=RPAM(I, JG)
    CONTINUE
    2874
    CONTINUE
    GO TO 2910
    C END OF LDMP FOR JJ
    2880 CONTINUE
    C END OF LDMP FOR J
    2890 CONTINUE
END OF LOOP FOR I

2900 CONTINUE

2910 CONTINUE
   GO TO (2200,2260,2320), KRET

END
SUBROUTINE best ( H, M1, M2, HBEST, MBEST, MINM, FIRST, SECOND,
1 THIRD, M2MAX )
IMPLICIT INTEGER (I-Z)
REAL H(M2MAX)
REAL FIRST(M2MAX)
REAL SECOND(M2MAX)
REAL THIRD(M2MAX)
REAL HBEST

C
C SUBROUTINE TO FIND BEST H
C OUT: HBEST, MBEST
C
M0=M1
IF ( M0 .LT. MINM ) M0 = MINM
HBEST=H(M0)
MBEST=M0
DO 10 I=M0, M2
IF ( HBEST .LE. H(I) ) GO TO 10
HBEST=H(I)
MBEST=I
10 CONTINUE
RETURN
END

SUBROUTINE GETMSK (M,FR,GRAM,NMASK,CTXT,PERC,DONT,M2MAX,MASK,IPART
1,*)
IMPLICIT INTEGER(A-Z)
INTEGER MASK(M2MAX)
IF (NMSK.NE.0) GO TO 2010
JJZ=2
XKZ=1
LATCH=1
MLIM=M
III=0
IEEG=0
NTOT=FR*M-2
IPART=1
2010 CONTINUE
C
C LOOP FOR IPART
C
2020 CONTINUE
C
2030 DO 2030 IG=1,M
MASK(IG)=CTXT
2030 CONTINUE
GO TO (2040,2050,2110,2140), IPART
C
C TYPE 3 MASK (REGULAR)
2040 CONTINUE
IF (M.GT.2) GO TO 2190
IREG=IREG+1
IF (IREG.GT.2) GO TO 2190
C LEFT END
IF (IREG.EQ.1) MASK(M)=PERC
C RIGHT END
IF (IREG.EQ.2) MASK(1)=PERC
GO TO 2180

C RIGHT-SENSITIVE AND LEFT-SENSITIVE
2050 CONTINUE
IF (M.LE.2) GO TO 2190
GO TO (2060, 2090), LATCH

C RIGHT-SENSITIVE
2060 CONTINUE
MLIM=MLIM-1
IF (MLIM.GT.0) GO TO 2070
LATCH=2
MLIM=M
GO TO 2050
2070 CONTINUE
DO 2080 JM=1, MLIM
MASK(JM)=PERC
2080 CONTINUE
GO TO 2180

C LEFT-SENSITIVE
2090 CONTINUE
MLIM=MLIM-1
IF (MLIM.LE.0) GO TO 2190
DO 2100 JL=1, MLIM
JJZ=M+1-JL
MASK(JJZ)=PERC
2100 CONTINUE
GO TO 2180

C TYPE 1 MASK (CONTEXT-SENSITIVE)
2110 CONTINUE
IF (M.LE.2) GO TO 2190
KZ=KZ+1
IF (KZ.LE.K-1) GO TO 2120
JJZ=JJZ+1
IF (JJZ.GT.K-1) GO TO 2190
KZ=JJZ
2120 CONTINUE
DO 2130 LK=JJZ, KZ
MASK(LK)=PERC
2130 CONTINUE
GO TO 2180

C TYPE 0 MASK (UNRESTRICTED REWRITE)
2140 CONTINUE
IF (M.LE.2) GO TO 2190
1111=1111*1
IF (1111.GT.MTGT) GO TO 2190
CALL CONV (MASK, 1111, M, FR, 1111)
DO 2150 IJK=1, M
IF (MASK(IJK).EQ.0) MASK(IJK)=CTX
IF (MASK(IJK).EQ.1) MASK(IJK)=PERC
2150 CONTINUE
C CALL MTEST (MASK, M, PERC, 0, 2180, 2140)
2180 CONTINUE
NMASK=NMASK+1
RETURN
2190 CONTINUE
IPART=IPART+1
IF(IPART.LE. 4-GRAM ) GO TO 2020
RETURN 1
END
SUBROUTINE MTEST(MASK,M,PERC,*,*)
IMPLICIT INTEGER(A-Z)
INTEGER MASK(M)
C
TEST MASK FOR GRAMMATICAL TYPE
C
RETURN 1: UNRESTRICTED RE-WRITE
C
RETURN 2: CONTEXT-SENSITIVE
NB=-1
CCC=-1
GO 2170 IK=1,M
IF(MASK(IK).EQ.NB) GO TO 2170
NB=MASK(IK)
CCC=CCC+1
2170 CONTINUE
IF(CCC.GE.3 .OR. (CCC.EQ.2 .AND. MASK(1).EQ.PERC)) RETURN 1
RETURN 2
END
SUBROUTINE GRID ( PRE, POST, M, DCNT, CTXT, PERC, SCR,G, ZLM)
IMPLICIT INTEGER(A-Z)
INTEGER PRE(M)
INTEGER POST(M)
INTEGER SCR(G)
INTEGER ZLM(3)
C
# DON'T CARE SYMBOL ___=CONTEXT PART ___=PREDICTED PART
DO 100 I=1,M
K=I
IF(POST(I).EQ.DCNT) SCR(K)=ZLM(1)
IF(POST(I).EQ.CTXT) SCR(K)=ZLM(2)
IF(PRE(I).EQ.PERC) SCR(K)=ZLM(3)
100 CONTINUE
RETURN
END
SUBROUTINE GRAPH ( X,N, NUMT, LEVEL, MBE, TY )
IMPLICIT INTEGER(A-Z)
REAL X(N)
INTEGER NUMT(5)
INTEGER SPUG(5)
INTEGER IMAGE(51,101)
INTEGER TY(N)
REAL COMA
REAL GAID
DATA BLANK/* /DATA VERT/*I*/DATA HORIZ/*-*/DATA PLUS/*+*/DATA STAR/*#*/
C
GRAPH PRINTS GRAPH GF P11'S USE IN RECODING
C
WRITE(6,1300) LEVEL, MBE
1800 FORMAT('1GRAPH OF P11 USED IN RECODING FOR LEVEL',I2,'1
AND M GF',I4,'/'' )
C
IF(N.LE.101) GO TO 150
COMP=N/100.
NWIDE=101
GO TO 170
CONTINUE
NWIDE=N
COMP=1.0

CONTINUE
C
DO 200 J=1, NWIDE
DO 200 I=1, 51
IMAGE(I,J)=BLANK
CONTINUE
C
VERTICAL LINES AT RIGHT AND LEFT EDGE
DO 235 I=2, 50
IMAGE(I,1)=VERT
IMAGE(I,NWIDE)=VERT
CONTINUE
C
FIND LOCATIONS OF VERTICAL LINES DIVIDING GRAPH
IS=0
DO 220 K=1,5
IS=IS+NUMT(K)
SPOT(K)=IS/CMP
IF(SPOT(K) .EQ. 0) SPOT(K)=1
CONTINUE
C
INSERT VERTICAL LINES DIVIDING GRAPH
DO 230 K=1,5
DO 230 I=2, 50
IF(I .EQ. 26) GO TO 230
IMAGE(I,SPOT(K))=VERT
CONTINUE
C
HORIZONTAL LINES
DO 240 II=1,3
I=II*(II-1)*25
DO 240 J=1, NWIDE
IMAGE(I,J)=HCR
IF(J .EQ. 1 .OR. J .EQ. NWIDE ) IMAGE(I,J)=PLUS
CONTINUE
C
DO 238 L=1, 5
IF(J .EQ. SPOT(L) ) IMAGE(I,J) = PLUS
CONTINUE
C
INSERT POINTS TO BE PLOTTED INTO GRAPH
DO 300 J=1, 51
JJ=COMP*J
IF(JJ .EQ. 0) JJ=1
II=1+ 50*#(JJ)
IMAGE(II,JJ)=STAR
CONTINUE
C
PRINT GRAPH
C
DO 500 I=1, 51
I=I+1
ORDIN=.02*(I-1)
WRITE(6,430) ORDIN, (IMAGE(II,J), J=1,NWIDE )
CONTINUE
C
FORMAT ( ', F4.2, 1X, 101A1 )
PRINT TYPES OF THE P(I)'S ACROSS BOTTOM OF GRAPH
WRITE(600, 600) (TY(I), I=1, NWIDE)
FORMAT (6X, 10111)
RETURN
END

SUBROUTINE SYMBOL (Y, ALPHA, N, K, CHAR, NCHAR, LETTER, N26)
IMPLICIT INTEGER (A-Z)
INTEGER Y(N)
INTEGER ALPHA(N)
INTEGER CHAR(NCHAR)
INTEGER LETTER(N26)

CONVERTS NUMERICAL STRING INTO ALPHABETIC SYMBOL STRING

IN:
Y=STRING OF NUMBERS
0,1,2,...
N=LENGTH OF Y
NCHAR=NUMBER OF SYMBOLS IN VECTOR CHAR
CHAR=VECTOR OF TERMINAL SYMBOLS
INCLUDING INITIAL PUNCTUATION, IF ANY
ALSO INCLUDING THE SYMBOLS OF ZLM
LETTER=VECTOR OF NON-TERMINAL SYMBOLS
N26=SIZE OF VECTOR LETTER

OUT:
K=LENGTH OF ALPHA STRING PRODUCED
ALPHA=RESULTING STRING OF CHARACTERS
ALPHA MUST BE DIMENSIONED TO AT LEAST 4
N26: ALPHA AND Y MUST BE DISTINCT
IF MORE THAN N26 NON-TERMINALS ARE NEEDED

DATA LP /*<*/
DATA RP /*>**/
DATA NBLANK /* */

ALPHA(1)=NBLANK
ALPHA(2)=NBLANK
ALPHA(3)=NBLANK
ALPHA(4)=NBLANK
K=0
IF(N.NE.0) GO TO 5
K=4
GO TO 999
CONTINUE
DO 15 I=1, N

TERMINAL SYMBOLS (INCLUDING INITIAL PUNCTUATION, IF ANY)
INCLUDING ZLM
IF ( .NOT. (Y(I).LT.NCHAR) ) GO TO 6
K=K+1
ALPHA(K) = CHAR(Y(I) + 1)
GO TO 15

FIRST N26 NON-TERMINAL SYMBOLS (SINGLE SYMBOL)
CONTINUE
IF ( .NOT. (Y(I).LT.NCHAR+N26) ) GO TO 7
K=K+1
ALPHA(K) = LETTER(Y(I) *1-NCHAR)
GO TO 15

C LATER NON-TERMINALS (PARENTHESES AND 2 NON-TERMINALS)
7 CONTINUE
L1=Y(I)-NCHAR/ N26
L2= Y(I) - NCHAR - L1*N26 + 1
K=K+1
ALPHA(K)= LP
K=K+1
ALPHA(K)=LETTER(L1)
K=K+1
ALPHA(K)=LETTER(L2)
K=K+1
ALPHA(K)=RP
15 CONTINUE
999 CONTINUE
RETURN
END
SUBROUTINE MCRE(PRE,PPOST,M,MASK, DONT,CTXF,PERC,MODE,PUNC )
IMPLICIT INTEGER(A-Z)
INTEGER PRE(M)
INTEGER POST(M)
INTEGER MASK(M)
C CREATES MASK FROM PRE AND POST SEQUENCES
DO 860 JD=1, M
K=M+1-JD
IF(PRE(JD) .EQ. PERC ) MASK(K) = PERC
IF(POST(JD) .EQ. CTXF ) MASK(K) = CTXF
IF( POST(JD) .EQ. DONT ) MASK(K) = DONT
860 CONTINUE
RETURN
END
SUBROUTINE GEN1 ( Y, N, NS, ALPHA, CHAR, NCHAR )
IMPLICIT INTEGER(A-Z)
INTEGER Y(NS)
INTEGER ALPHA:NS)
INTEGER CHAR(NCHAR)
READ ( 5, 12 ) N
12 FORMAT(15)
READ ( 5, 20 ) (ALPHA(I), I=1, N )
20 FORMAT( 80A1 )
DO 30 I=1, N
CALL IMATCH(CHAR, NCHAR, ALPHA(I), Y(I) )
Y(I)=Y(I)-1
30 CONTINUE
RETURN
END
SUBROUTINE LCC (NS,N,M,STRIC,M8E,SQLEN,TYPE,ALLOW,PRE,C,Y,MTSIZE
1,M2MAX,PST,SEQPRE,SEQ,WONT,PUNC,INDEX,SEQNO,NS1,SEQ_TX,PERC,CTXF
2,VDIR,USED,CLAST,VSTART,M2)
IMPLICIT INTEGER(A-Z)
INTEGER Y(NS)
INTEGER SEQ(ns)
INTEGER SQ_TX(MTSIZE)
INTEGER SQLEN(MTSIZE)
INTEGER SEQPRE(MTSIZE)
INTEGER TYPE(MTSIZE)
INTEGER INDEX(MTSIZE)
INTEGER USED(MTSIZE)
INTEGER POST(M2MAX)
INTEGER PRE(M2MAX)

SEARCH Y FOR ALLOWABLE SEQUENCES
IN:
  MBE=VALUE OF M TO BE USED IN THIS RECODING
OUT:

IF (NST.EQ.0) GO TO 2141
VARIOUS M = M1 ... MBEST
IF (VDIR.EQ.1) M=M1
IF (VDIR.EQ.1) M=M2
IF (STRUCT.EQ.1) M=MBE
CONTINUE

CCNSIDER T = M, ... N

T=M

IF (VSTART.EQ.1.AND.T=QLAST.LT.MBE) GO TO 2137
CONTINUE

DO 2135 LJ=1,2
DO 2134 LTA=1, ALLOW
DO 2133 LLL=1,NST
L=INDEX(LLL)
IF (LJ.EQ.1.AND.USED(L).EQ.0) GO TO 2133
IF (LJ.EQ.2.AND.USED(L).EQ.1) GO TO 2133
IF (OLEN(L).NE.M) GO TO 2133
IF (TYPE(L).NE. LTA ) GO TO 2133
IF (MODE.EQ.1) GO TO 2129
DO 2127 JD=1,M
TMJD=T-M+JD
IF (Y(TMJD).NE.PUNC) GO TO 2127
QLAST=TMJD
GO TO 2137

CONTINUE

CALL CCNV (PRE,M,C,SQCTXT(L))
CALL CCNV (POST,M,C,SQPRED(L))
DO 2131 JD=1,M
TMJD=T-M+JD
IF (MODEL.EQ.2.AND.Y(TMJD).EQ.PUNC) GO TO 2137
IF (POST(JD).EQ.0) GO TO 2131
IF (PRE(JD).EQ.PHIC.AND.POST(JD).NE.Y(TMJD)) GO TO 2133
IF (POST(JD).EQ.YTXT.AND.PRE(JD).NE.Y(TMJD)) GO TO 2133
CONTINUE

LOCATED SEQUENCE OF LENGTH M AT TIME T
SEQNL(T)=THE ROW (L) IN STHI,STNEXT,STLEN TABLES OF THE
LONGEST SEQUENCE OF ALLOWABLE TYPE ENDING AT TIME T
USED(L)=1
SEQNO(T)=L
GO TO 2137

CONTINUE
CONTINUE
CONTINUE
T = T + 1
IF (T .LE. N) GO TO 2125
M = M + 1
IF (STRONG .EQ. 1) GO TO 2139
IF (VDIR .EQ. -1 .AND. M .GE. M1) GO TO 2123
IF (VDIR .EQ. 1 .AND. M .LE. MBE) GO TO 2123
CONTINUE
CONTINUE
RETURN
END
SUBROUTINE TLOC (NS, M1, STRONG, MBE, SLEN, TYPE, ALLOC, PRE, C, Y, MTSIZE, 
M2MAX, POST, SQRT, MODE, DON'T, PUNC, INDEX, NST, SQRTXT, PERC, CTXT, VSPEC, 
2SPEC, USED, T, Y, VSTART, VDIR, QLAST, LAWL, M2)
IMPLICIT INTEGER (A-Z)
INTEGER YNS
INTEGER USED(MTSIZE)
INTEGER SQRTXT(MTSIZE)
INTEGER SLEN(MTSIZE)
INTEGER SQRT(MTSIZE)
INTEGER TYPE(MTSIZE)
INTEGER INDEX(MTSIZE)
INTEGER POST(M2MAX)
INTEGER PRE(M2MAX)

SEARCH Y FOR ALLOWABLE SEQUENCES
GIVEN T

IN:
MBE = VALUE OF M TO BE USED IN THIS RECODING
OUT:

VSPEC = 0
SPEC = 0
IF (NST .EQ. 0) GO TO 1639
IF (VSTART .EQ. 1 .AND. T .GT. QLAST .LT. MBE) GO TO 1639

VARIOUS M = M1 ... MBEST
IF (VDIR .EQ. 1) M = M1
IF (VDIR .EQ. -1) M = M2
IF (STRONG .EQ. 1) M = MBE

CONTINUE
DO 1633 LJ = 1, 2
DO 1632 LTA = 1, ALLOW
DO 1631 LLL = 1, NST
L = INDEX(LLL)
IF (LJ .EQ. 1 .AND. USED(L).EQ.0) GO TO 1631
IF (LJ .EQ. 2 .AND. USED(L).EQ.1) GO TO 1631
IF (SLEN(L) .NE. M) GO TO 1631
IF (TYPE(L) .NE. LTA) GO TO 1631
IF (MODE .EQ. 1) GO TO 1627
DO 1625 JD=1,M
TMJD=Y-M+JD
IF (Y(TMJD).NE.PUNC) GO TO 1625
QLAST=TMJD
GO TO 1639
1625 CONTINUE
1627 CONTINUE
CALL CGNV (PRE,M,C,SQCTX(L))
CALL CGNV (POST,M,C,SQREDIL))
DO 1629 JD=1,M
TMJD=T-M+JD
IF (MODE.EQ.2.AND.Y(TMJD).EQ.PUNC) GO TO 1635
IF (POST(JD).EQ.DUNT) GO TO 1629
IF (PRE(JD).EQ.PERC.AND.POST(JD).NE.Y(TMJD)) GO TO 1631
IF (POST(JD).EQ.CTXT.AND.PRE(JD).NE.Y(TMJD)) GO TO 1631
1629 CONTINUE
C  LOCATED SEQUENCE OF LENGTH M AT TIME T
IF (LJ.EQ.1) VSPEC=L
IF (LJ.EQ.2) SPEC=L
1631 CONTINUE
1632 CONTINUE
1633 CONTINUE
1635 CONTINUE
M=M+VOIR
IF (STRICM.EQ.1) GO TO 1637
IF (VOIR.EQ.0.AND.M.GE.M1) GO TO 1623
IF (VOIR.EQ.1.AND.M.LE.MBE) GO TO 1623
1637 CONTINUE
C
1639 CONTINUE
L=0
IF (SPEC.NE.0) L=SPEC
IF (VSPEC.NE.0) L=VSPEC
IF (L.NE.0) USED(L)=1
C
RETURN
END
SUBROUTINE GEN2 (STRING,N,NS,MODE,NCHAR,PUNC)
IMPLICIT INTEGER (A-Z)
INTEGER STRIN(N)
REAL PRB(N)
REAL YFL
C
SPECIAL BINARY SEQUENCE GENERATOR FOR SENTENCES OF UNIFORM
LENGTH
C
TERMINAL SYMBOLS: 0,1 (S0, S1)
C
RULES OF PRODUCTION:
C
S --> XA
C
A --> YA
C
A --> ZA
C
A --> WA
C
X --> 11
C
Y --> 01
C
Z --> 10
W --> 00

IN:
N5=DIMENSIONED SIZE OF STRING
NCHAR=NUMBER OF TERMINAL SYMBOLS (INCLUDING PUNCTUATION, IF ANY)
MGOE=DETERMINES METHOD OF INITIAL PUNCTUATION
OUT:
N=SIZE OF STRING PRODUCED
STRING=THE STRING PRODUCED
READ IN FROM CARD:
NEAR=APPROXIMATE SIZE OF STRING DESIRED
BEGIN=SEED FOR RAND
N8=SENTENCE LENGTH
DIV: 2=USE NON-TERMINALS Y AND Z  3=USE Y, Z, AND W

S0=3
S1=4

READ ONE CONTROL CARD

READ (5, 13) NEAR, BEGIN, N8, DIV
FORMAT (1615)

DO 8 L=1, DIV
8 PROB(L)=0

N=0

NEW SENTENCE

CONTINUE
IF ( N .GT. NEAR ) GO TO 30

LETTER X=11
SENTENCE ALWAYS BEGINS WITH AN X
N=N+1
STRING(N)=S1
N=N+1
STRING(N)=S1
J=2

CONTINUE
CALL RANDU(BEGIN, IY, YFL)
BEGIN=IY
IND= YFL*DIV+1.0
PROB(IND) = PROB(IND) +1.0
IF ( J .GE. N8 ) GO TO 28
GO TO ( 10, 20, 29), IND

LETTER Y=01

CONTINUE
N=N+1
STRING(N)=SC
N=N+1
STRING(N)=S1
J=J+2
GO TO 5

LETTER Z=10

20 CONTINUE
N=N+1
STRING(N)=S1
N=N+1
STRING(N)=S0
J=J+2
GO TO 5

LETTER W=00

25 CONTINUE
N=N+1
STRING(N)=S0
N=N+1
STRING(N)=S0
J=J+2
GO TO 5

END OF SENTENCE
INCLUDE INITIAL PUNCTUATION MARK, IF MODE=2

28 CONTINUE
IF ( MODE .NE. 2 ) GO TO 4
N=N+1
STRING(N)=PUNC
GO TO 4

PRINT OUT PROBS AS CHECK

30 CONTINUE
DO 36 L=1, DIV
P[L]=P[L]/N*2.0
36 CONTINUE
WRITE (6, 99) ( PROB(I), I=1, DIV)
99 FORMAT (10F10.8)

RETURN
END

SUBROUTINE HUFF( M, C, M2, P, LEN, KDE, PC, ACT, TAB, LMIN, MKD, LIST )

IMPLICIT INTEGER ( A-Z )
REAL P(M)
REAL PC(M2)
INTEGER LEN(M)
INTEGER KDE(M, MKD)
INTEGER ACT(M2)
INTEGER TAB(M,C)
INTEGER LMIN(C)
INTEGER LIST(M)
REAL SUM
REAL MIN

HUFFMAN CODING SCHEME
STARTS WITH M MESSAGES OF PROB P(I)
CODES THEM INTO SEQUENCES OF C SYMBOLS

M= # OF MESSAGES
M2 = 2 * M
C C = # OF SYMBOLS IN ALPHABET FOR MESSAGE
C P(I) = PROB OF MESSAGE I
C MKD = MAX LENGTH OF ANY ENCODED MESSAGE (2ND SUB OF KODE)
C OUT:
C KCODE(I,J) = CODE FOR MESSAGE I (USES LEN(I) SYMBOLS)
C LEN(I) = LENGTH OF CODE FOR ORIGINAL SEQ I
C SCRATCH VECTORS:
C PC(I) = ORIGINAL VECTOR P PLUS ADDITIONAL LINES
C ACT(I) = 0 IMPLIES LINE IS INACTIVE, 1 IMPLIES ACTIVE
C TAB(I,J) = THE LINES USED TO CREATE LINE M+1 IN TABLE
C LINE M+1 HAS ONLY MG ENTRIES. OTHER LINES HAVE DEL=C
C LMIN(I) = LOCATION OF ONE OF THE K SMALLEST PC'S
C LIST = C-ARY NUMBER USED IN TREE SEARCH
C
C***************************************************************************************
C CALLING SEQUENCE
C***************************************************************************************
C NEED NAME, CC, PB
C REAL PC(400)
C DIMENSION KODE(200, 20)
C DIMENSION TAB(200, 2)
C DIMENSION LEN(200)
C DIMENSION ACT(400)
C DIMENSION LIST(200)
C DIMENSION LMIN(10)
C MKD = 20
C
C RECODING
C IN TO HUFF:
C NAME = NUMBER OF MESSAGES TO BE RECODED
C CC = NUMBER OF SYMBOLS
C PB = PROBABILITY VECTOR
C MKD = DIMENSIONED SIZE OF 2ND DIM OF KODE
C M KK = TWICE NAME
C PRODUCED BY SUBROUTINE: KODE, LEN
C KCODE(I, J) = CODE FOR MESSAGE I (USES LEN(I) SYMBOLS)
C LEN(I) = LENGTH OF CODE FOR ORIGINAL SEQ I
C SCRATCH FOR HUFF: TAB, ACT, LIST, LMIN
C IF (EXR .NE. 2) GO TO 980
C IF (NAME .EQ. 0) GO TO 980
C MKD = 2 * NAME
C CALL HUFF(NAME, CC, M KK, PB, LEN, KODE, PC, ACT, TAB, LMIN, MKD, LIST)
C 980 CONTINUE
C***************************************************************************************
C
C
C INITIALIZE
DO 7 I = 1, M
ACT(I) = 1
PC(I) = P(I)
LEN(I) = 0
7 CONTINUE
DO 9 L = 1, C
DO 9 I = 1, M
TAB(I,L) = 0
9 CONTINUE
C
C FIND MO
IF ( C .EQ. 2 ) GO TO 20
CL=C-1
DO 10 MU=2,C
NU= M - MD
Q= NU / CL
IF ( Q * CL .EQ. NU ) GO TO 30
10 CONTINUE
20 CONTINUE
MO=2
30 CONTINUE
C MAIN SECTION
C FIRST TIME DEL=MG, THEN DEL=C
C
DEPTH=M
TEDPHT=0
NACT=M
DEL=MD
C FIND DEL SMALLEST PROBS
C GET SUM OF DEL SMALLEST PROBS
C
22 CONTINUE
SUM=0,J
DO 70 L=1, DEL
MIN=9999
DO 60 I=1, DEPTH
IF ( MIN .LE. PC(I) .OR. ACT(I) .NE. 1 ) GO TO 60
MIN = PC(I)
60 LMIN(I) = I
70 CONTINUE
II=LMIN(J)
ACT(II) =0
SUM=SUM + PC(II)
CONTINUE
NACT=NACT - DEL
C CREATE NEW LINE IN PC-TABLE EQUAL TO SUM OF THE DEL SMALLEST
C
DEPTH=DEPTH +1
TEDPHT=TEDPHT+1
ACT(DEPTH) =1
PC(DEPTH) = SUM
DO 90 J=1, C
TAB( TEDPHT, J ) = LMIN(J)
90 CONTINUE
C IF ( NACT .LT. 1 ) GO TO 100
NACT=NACT+1
DEL = C
GO TO 22
100 CONTINUE
C CREATE CODE SEQUENCES
C
DO 120 I=1, M
LIST(I) =1
120 CONTINUE
C TREE SEARCH
125 CONTINUE
   II=TDEPTH
   K=0
130 CONTINUE
   K=K+1
   II = TAB( II, LIST(K) )
   IF ( II .LE. M ) GO TO 140
   II=II - M
   GO TO 130
C
C REACHED ENDPOINT OF TREE
140 CONTINUE
   IF ( II .EQ. 0 ) GO TO 190
   DO 150 LL =1, K
   KDDE(II,LL) = LIST(LL) -1
150 CONTINUE
   LEN(II) = K
190 CONTINUE
   LIST(K) = LIST(K) +1
   IF ( LIST(K) .LE. C ) GO TO 125
   LIST(K) =1
   K=K-1
   IF ( K .GT. 0 ) GO TO 190
C
C PRINT CODES
C
PRH=0
   IF ( PRH .EQ. 0 ) GO TO 361
   DO 360 I=1, M
      JJ= LEN (I)
      WRITE (6, 341 ) I, ( KDDE(I,J), J=1, JJ )
341 FORMAT ( 1X, I4, 5X, 120I1 )
360 CONTINUE
361 CONTINUE
RETURN
END
SUBROUTINE REMOVE ( IN, OUT, NIN, NOUT, FILL )
 IMPLICIT INTEGER(A-Z)
 INTEGER IN(ININ)
 INTEGER OUT(ININ)
 TG=0
 DO 10 T=1, AIN
   IF ( IN(T) .EQ. FILL ) GO TO 10
   TG=TG+1
   OUT(TG)=IN(T)
10 CONTINUE
   NOUT= TG
RETURN
END
SUBROUTINE ZERO(INVEXT,N)
 ENTRY ZER0(INVEXT,N)
 INTEGER NVECT(N)
C
SUBROUTINE TO ZERO VECTOR
 DO 10 I=1,N
   NVECT(I)=0
10 CONTINUE
RETURN
END
SUBROUTINE FZERO(INVEXT,N)
REAL NVECT(N)

SUBROUTINE TO ZERO VECTOR
DO 10 I=1,N
NVECT(I)=.0
10 CONTINUE
RETURN
END

SUBROUTINE SPRAY(NV, N, NVAL)
ENTRY ISPRAY(NV,N, NVAL)
INTEGER NVECT(N)

SUBROUTINE TO SET VECTOR TO A CONSTANT
DO 10 I=1, N
NVECT(I)=NVAL
10 CONTINUE
RETURN
END

FUNCTION ISUM(NV, N)
INTEGER NVECT(N)

ISUM = FINDS SUM OF VECTOR
ISUM=0
DO 10 I=1, N
ISUM=ISUM+NVECT(I)
10 CONTINUE
RETURN
END

SUBROUTINE PHI ( DIGIT, M, C, SUM )
IMPLICIT INTEGER (A-Z)
INTEGER DIGIT(M)

PHI ASSIGNS AN INDEX NUMBER TO A GIVEN M-SEQUENCE
PHI=NATURAL NUMBER CORRESPONDING TO M DIGITS MODULO C

IN: DIGIT, M, C
DIGIT=VECTOR OF M DIGITS MOD C
OUT: PHI
PHI=NATURAL NUMBER PRODUCED

SUM=0
E=1
DO 120 L=1,M
SUM=SUM+DIGIT(M+1-L)
E=E*C
120 CONTINUE
RETURN
END

SUBROUTINE CONV ( DIGIT, M, C, NUM )
IMPLICIT INTEGER (A-Z)
INTEGER DIGIT(M)

CONVERTS NUMBER 'NUM' TO M DIGITS MOD C

IN: NUM, M, C
NUM=NATURAL NUMBER
OUT: DIGIT
DIGITS= M DIGITS MOD C

N=NUM
DO 10 L=1, M
Q=N/C
DIGIT ( M-L+1 ) = N-Q*C
N=0
10 CONTINUE
RETURN
END
SUBROUTINE XFSORT ( V, N, INDEX, N1, N2 )
ENTRY FXSORT ( V, N, INDEX, N1, N2 )
REAL V(N)
REAL T
INTEGER INDEX(N)

INDEXED SORTING ROUTINE.
SORTS BLOCK WITHIN VECTOR V INTO HIGH...LOW ORDER
IN:
V=VECTOR OF LENGTH N OF VALUES TO BE SORTED
N=DIMENSIONED SIZE OF V
N1=INDEX OF BEGINNING OF BLOCK WITHIN V TO BE SORTED
N2=INDEX OF END OF BLOCK WITHIN V TO BE SORTED
OUT:
INDEX(I) = SUBSCRIPT OF I-TH LARGEST ELEMENT OF V
CHANGED:
THAT PART OF VECTOR V WHICH IS SORTED
NB: IN MAIN, INDEX VECTOR V BY 2, AND ALL OTHER ASSOCIATED
VECTORS BY INDEX(I)

DO 324 KJ=N1, N2
   INDEX(KJ) = KJ
IF ( N1 .GE. N2 ) GO TO 999
   NL= N2-1
   DO 300 L= N1, NL
      IF ( V(L) .GE. V(L+1) ) GO TO 300
C
   V(L+1) GREATER
   LLL= L+1 - N1
   DO 160 J=1, LLL
      K=L+1 - J
      IF ( V(K) .GE. V(K+1) ) GO TO 200
T=V(K)
V(K) = V(K+1)
V(K+1) = T
N=INDEX(K)
INDEX(K) = INDEX(K+1)
INDEX(K+1) = N
100 CONTINUE
200 CONTINUE
300 CONTINUE
999 CONTINUE
RETURN
END
SUBROUTINE LOOK ( LLIST, LDIM, LEN, ITEM, LEVEL, NEW )
IMPLICIT INTEGER (A-Z)
INTEGER LLIST(LDIM)

ITEM=CURRENT ELEMENT
LDIM=DIMENSIONED SIZE OF LLIST
LEN=ACTUAL LENGTH OF LLIST
LEVEL=INDEX OF PLACE WHERE ITEM IS FOUND IN LLIST
LLIST(I)=LIST OF ITEMS
NEW: 0=ITEM WAS IN LLIST ALREADY

1=ITEM IS NEW, AND WAS ADDED TO LLIST
C
C INITIALIZE: LEN BEFORE FIRST ENTRY
C
C
IND=1
GO TO 50
ENTRY
ENTRY 1
LDCKN( LLIST, LDIM, LEN, ITEM, LEVEL, NEW)
IND=0
50 CONTINUE
IF (LEN.EQ.0) GO TO 150
DO 100 I=1, LEN
100 CONTINUE
IF (LLIST(I).EQ. ITEM) GO TO 200
C CONTINUE
C NO MATCH
150 CONTINUE
NEW=1
IF (IND.EQ.0 .OR. LEN.LT. LDIM) GO TO 168
PRINT 166, LDIM
166 FORMAT ('Too MANY ELEMENTS IN LIST OF LENGTH ', I6 )
C NB: LEVEL SET TO LAST POSSIBLE LEVEL IN THIS CASE
LEVEL=LDIM
GO TO 250
168 CONTINUE
LEN=LEN+1
LLIST(LEN)=ITEM
LEVEL=LEN
GO TO 250
C CURRENT *ITEM* MATCHES WITH EXISTING ELEMENT OF LLIST
200 CONTINUE
NEW=0
LEVEL=1
250 CONTINUE
RETURN
END
SUBROUTINE ISEEK ( NVECT, N, NOW, KCODE )
INTEGER NVECT(N)
DO 10 I=1, N
IF ( NVECT(I) .EQ. NOW ) GO TO 20
10 CONTINUE
KCODE=0
RETURN
20 CONTINUE
KCODE=1
RETURN
END
SUBROUTINE IXSPRINVECT, NDIIM, NVAL, I1, I2 )
INTEGER NVECT(NDIM)
DO 10 I=I1, I2
NVECT(I)=NVAL
10 CONTINUE
RETURN
END
SUBROUTINE IEQUAL ( JVECT, N, KVECT )
INTEGER JVECT(N)
DO 10 I=1, N
JVECT(I)=KVECT(I)
10 CONTINUE
RETURN
END
SUBROUTINE MATCH (NVECT, N, NOW, KODE)
ENTRY MATCH(NVECT,N,NOW,KODE)
INTEGER NVECT(N)
DO 10 I=1, N
IF (NVECT(I) .EQ. NOW ) GO TO 20
10 CONTINUE
PRINT 15, NOW
15 FORMAT ( 'OUNABLE TO MATCH:', A4)
KODE=0
RETURN
20 CONTINUE
KODE=1
RETURN
END

SUBROUTINE RANDU (IX, IY, YFL)

DESCRIPTION OF PARAMETERS
IX - FOR THE FIRST ENTRY THIS MUST CONTAIN ANY ODD INTEGER
NUMBER WITH NINE OR LESS DIGITS. AFTER THE FIRST ENTRY,
IX SHOULD BE THE PREVIOUS VALUE OF IY COMPUTED BY THIS
SUBROUTINE.

IY - A RESULTANT INTEGER RANDOM NUMBER REQUIRED FOR THE NEXT
ENTRY TO THIS SUBROUTINE. THE RANGE OF THIS NUMBER IS
BEWEEN ZEAG AND 2**31

YFL - THE RESULTANT UNIFORMLY DISTRIBUTED FLOATING POINT,
RANDOM NUMBER IN THE RANGE 0 TO 1.0

REAL YFL
IX=65549
DO 1 K=1, 10
CALL RANDU ( IX, IY, YFL )
1 CONTINUE

REMARKS
THIS SUBROUTINE IS SPECIFIC TO SYSTEM/360 AND WILL PRODUCE
2**29 TERMS BEFORE REPEATING. THE REFERENCE BELOW DISCUSSES
SEEDS (65539 HERE), RUN PROBLEMS, AND PROBLEMS CONCERNING
RANDOM DIGITS USING THIS GENERATION SCHEME. MCLAREN AND
MARSAGLIA; JACM 12, P. 83-89 DISCUSS CONGRUENTIAL
THE RANDU TYPE, ONE FILLING A TABLE AND ONE PICKING FROM THE
TABLE, IS OF BENEFIT IN SOME CASES. 65549 HAS BEEN
SUGGESTED AS A SEED WHICH HAS BETTER STATISTICAL PROPERTIES
FOR HIGH ORDER BITS OF THE GENERATED DEVIATE.
SEEDS SHOULD BE CHOSSEN IN ACCORDANCE WITH THE DISCUSSION
GIVEN IN THE REFERENCE DELCM. ALSO, IT SHOULD BE NOTED THAT
IF FLOATING POINT RANDOM NUMBERS ARE DESIRED, AS ARE
AVAILABLE FROM RANDU, THE RANDOM CHARACTERISTICS OF THE
FLOATING POINT DEVIATES ARE MODIFIED AND IN FACT THESE
DEViates HAVE HIGH PROBABILITY OF HAVING A TRAILING LOW
ORDER ZERO BIT IN THEIR FRACTIONAL PART.
POWER RESIDUE METHOD DISCUSSED IN IBM MANUAL C20-8011,
RANDU NUMBER GENERATION AND TESTING

INTEGER NUMBERS ARE AL CDD

IY=IX*65539
IF (IY).LT.6
5 IY=IY+2147483647+1
6 YFL=IY
YFL=YFL*4656613E-9
IX=IY
RETURN
END
BIBLIOGRAPHY


Miller, G. A., "The Magic Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information" *Psychological Review* 1956, 63, 81-97


Riordan, John, *An Introduction to Combinatorial Analysis*, Wiley 1958


Smith, L. D. *Cryptography*, (1943) Dover 1955


