ERROR-CORRECTING PARSING FOR SYNTACTIC PATTERN RECOGNITION

S. Y. Lu
and
K. S. Fu

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE 77-31
August 1977

This work was supported by the AFOSR Grant 74-2661.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR-77-1253
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ERROR-CORRECTING PARSING FOR SYNTACTIC PATTERN RECOGNITION
5. TYPE OF REPORT & PERIOD COVERED: Interim rept.
6. PERFORMING ORG. REPORT NUMBER: TR-EE 77-31
7. AUTHOR(s): S. Y. Lu, K. S. Fu
8. CONTRACT OR GRANT NUMBER(s): AFOSR-74-2661
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Purdue University, School of Electrical Engineering, West Lafayette, IN 47907
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB DC 20332
12. REPORT DATE: Aug [redacted] 77
13. NUMBER OF PAGES: 309
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 324 p.
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 292 000
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers

ERROR-CORRECTING PARSING FOR SYNTACTIC PATTERN RECOGNITION[†]

S. Y. Lu and K. S. Fu

TR-EE 77-31

August, 1977

[†]This work was supported by the AFOSR Grant 74-2661.

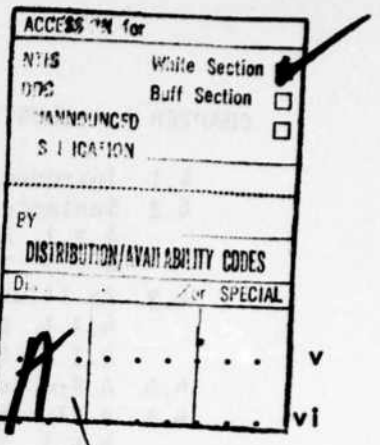


TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

Figure

Figure

Appendix Figure

ABSTRACT

The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers are employed to determine the distance between an input syntactic pattern and a formed cluster, or a language. Finally, using the error-correcting parsing techniques, real data examples on texture modeling and discrimination are presented.

CHAPTER I

INTRODUCTION

1.1 Purpose

During the past decade, there has been an increasing interest in pattern recognition. Most of the developments in the theory and applications of pattern recognition use the statistical approach [1-3]. In order to represent the structural information contained in the patterns, the syntactic or structural approach has been proposed [4-5]. This approach draws an analogy between the structure of patterns and the syntax of languages. The precision of syntactic specification provides the recognition procedure not only the capability of classifying patterns but also the capacity of describing patterns. However, one of the weaknesses of this approach is the problem of recognizing noisy patterns. Several approaches have been used in dealing with noisy patterns, namely; stochastic grammars or discriminant grammars, sequential parsing or partial parsing methods, language transformations, and error-correcting parsers. The purpose of this research is to develop error-correcting parsing algorithms suitable for syntactic pattern recognition.

1.2 The Recognition of Noisy and Distorted Patterns

Using a syntactic approach to pattern recognition, a set of training patterns is first analyzed. A pattern, according to its structure, is divided into subpatterns. Subpatterns can be further divided into sub-subpatterns, and so on. The basic element is called a "pattern primitive."

Linguistic notations are used to describe a pattern in terms of primitives and relations between them as in a sentence. The set of sentences corresponding to the set of training patterns can be specified by a generative grammar called a "pattern grammar." Non-terminals in the pattern grammar represent the subpatterns and terminals represent primitives and possibly some relational symbols. The structure of patterns is characterized by production rules of the grammar.

In a recognition procedure, after preprocessing, segmentation and primitive extraction, an input pattern is represented as a sentence, then a parser is employed as a pattern recognizer. A parser is an algorithm based on a given pattern grammar, G, that can produce a complete syntactic description in the form of a parse tree of an input sentence if it belongs to L(G), the language generated from G. A block diagram of a syntactic pattern recognition system is given in Figure 1-1.

In a pattern classification problem, parsers are used to determine the membership of an input pattern. Grammars are constructed to characterize each of the classes of patterns. An input pattern is then parsed with respect to the pattern grammars one by one to decide which language (class) it belongs to.

In practical applications, there often exists pattern distortion and measurement noise causing segmentation and primitive extraction errors which ultimately result in a noisy representation (sentence), that is, it cannot be successfully analyzed by the parser. The following are situations that may cause the representation of a pattern to be noisy; (1) unpredictable distortions and variations, (2) simplified grammars.

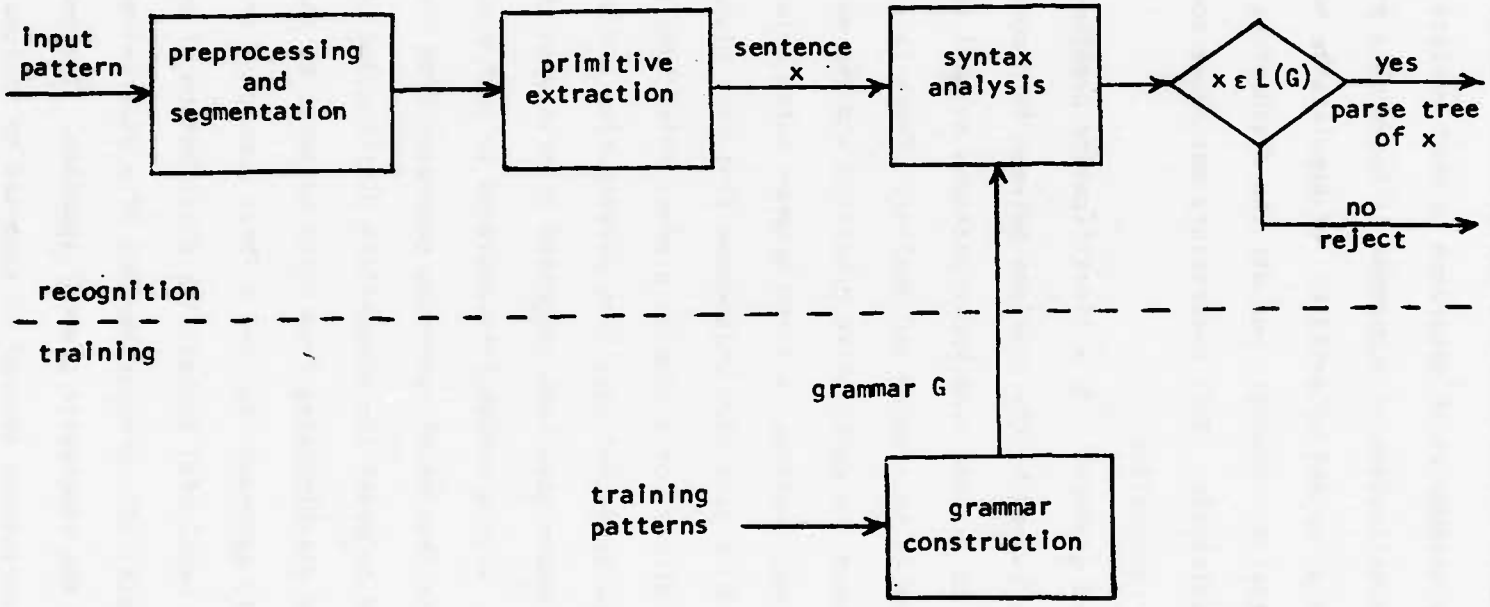


Figure 1.1 Block diagram of a syntactic pattern recognition system.

(1) Unpredictable distortions and variations. Normally, one would like to construct a grammar which generates as much variety in patterns as possible. The construction of a grammar is based on a priori knowledge available; e.g. the given set of patterns, or predictable noise, distortion or variation of patterns. However, not all the distortions and variations of patterns are predictable. This uncertainty may cause some patterns to be rejected during recognition.

(2) Simplified grammars. In a classification problem, in order to avoid any ambiguity caused by the overlaps between languages, grammars may be constructed to exclude some known patterns as well as some expected distortions in order to be simpler and smaller. There is a decision which has to be made between the descriptive precision and the analysis efficiency of a grammar. One may construct a large grammar (with a large number of production rules) which generates a language that very closely yields the given set of patterns, or a simpler grammar which does not generate some of the known patterns but uses less parsing time and storage space.

Stochastic grammars have been suggested in resolving the uncertainty of patterns [6-8]. With a probability assigned to each production rule, the probability distribution of sentences generated from the stochastic grammar can be used to model the probability distribution of patterns. Normal patterns are discriminated from noisy patterns by their associated probabilities. This approach requires a large amount of training data in order to make a meaningful probability distribution of patterns.

Page and Filipski [9], propose the use of a discriminant grammar which is an extension of the stochastic grammar approach. The generated language from a discriminant grammar is supposed to include all the classes

of patterns under consideration. In a discriminant grammar, a number is associated with each production rule. By adding the numbers corresponding to each use of a production rule in a derivation of a sentence, a number associated with the sentence is derived. The language is partitioned into decision regions by comparing this number to a predetermined cut point. A discriminant grammar can also be used in making a Bayes' decision between two stochastic languages. In this case, the number assigned to each production rule is obtained from the probability distribution of sentences. Therefore, the construction of discriminant grammars faces a similar problem to that of stochastic grammars, namely, that a very large amount of training data is necessary in order to make a meaningful probability distribution.

An interesting feature that stochastic grammars and discriminant grammars have is the sequential parsing. Persoon and Fu [10] proposed an algorithm of sequential parsing for stochastic context-free grammars. Using a stopping rule, only part of the input string needs to be scanned when a decision is made. The classification rule used is Bayes' decision rule. Page and Filipski [9] also proposed a scheme for sequential parsing of discriminant grammars based on the sequential probability ratio test. Using the sequential parsing method, since only the left part of a string is involved in the decision making process, one may construct the pattern grammars in such a way that the most informative, distinctive subpatterns and primitives are generated first. Consequently, the sequential classification schemes demonstrate an error tolerance capability to some extent. The error tolerance of sequential parsing is investigated in Chapter 2 of this thesis.

Ali and Pavlidis [11] used a similar idea for the construction of parsers for hand written numerals. Finite state grammars are used for the characterization of each class (numeral). A set of finite state automata is designed to read primitives considered to be discriminanting, while unimportant primitives are neglected. Since the recognizer ignores some irrelevant details which are extracted by pattern description algorithms it can thus reduce recognition errors.

The use of language transformation for the representation of special types of pattern distortions, such as scaling, rotation and replacement, etc. was suggested by Fu and Bhargava [12]. This concept also appears in Aho and Peterson's paper [13] where error transformation for substitution, deletion and insertion errors are defined. Aho and Peterson further expand the original grammar to include error transformations as production rules. Based on the expanded grammar an error-correcting parsing algorithm was formulated for substitution, deletion and insertion errors in general. The correction satisfies the minimum-distance criterion.

The approach of using error-correcting parsers is the method used in this research. A parser constructed on a given grammar, G, performs the function of analysing an input string, x. The analysis result is a complete parse of x, if x is in L(G), the language generated from G. From the parse, a derivation tree which represents the structure of x can be reconstructed. Suppose that x is not in L(G). Then the parser can, at most, generate a partial parse. Therefore, for a given gramar, G, a parser can be used to answer the membership problem, it can also be used to describe the syntax structure if the input sentence is in L(G) [14]. An error-correcting parser is designed to generate a complete parse even

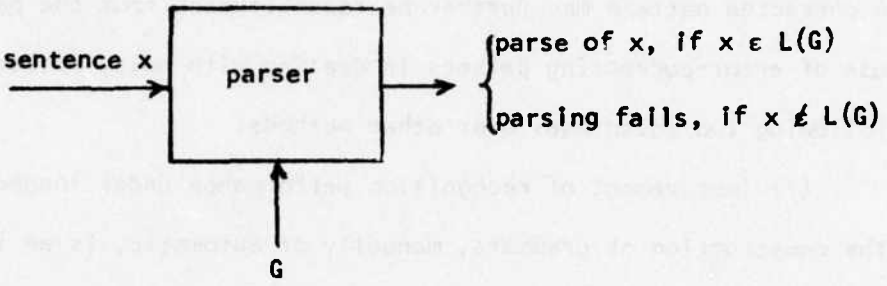
if the input sentence is not in L(G). Hence, using an error-correcting parser in a pattern recognition problem, a noisy pattern can be successfully analyzed and recognized. Block diagrams are given in Figures 1-2(a) and (b) to illustrate the function of a parser and an error-correcting parser. A corrected pattern may further be reconstructed from the parse. The use of error-correcting parsers in dealing with noisy patterns has the following two advantages over other methods.

(1) Improvement of recognition performance under inadequate training. The construction of grammars, manually or automatic, is an important part in the design of a syntactic pattern recognition system. An elaborate design gives better recognition performance, but such a design is certainly data dependent. Inadequate inference procedures or insufficient training data will result in a poorly constructed grammar, and consequently poor recognition performance. The use of error-correcting parsers as a pattern recognizer will compensate this difficulty in grammar construction. Hence, even with a rather poorly constructed grammar, the use of error-correcting parsers can give satisfactory recognition results.

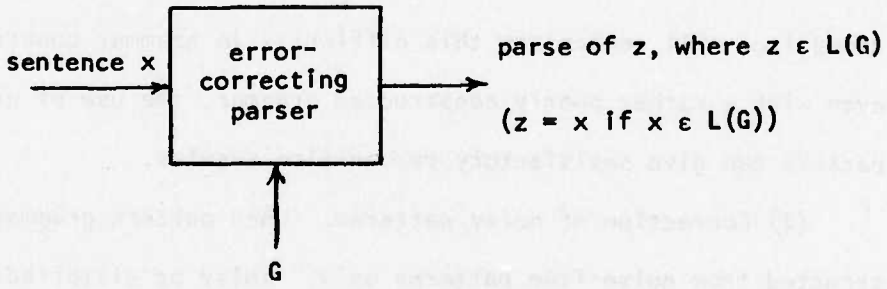
(2) Correction of noisy patterns. When pattern grammars are constructed from noise-free patterns only, noisy or distorted pattersn can be corrected by using error-correcting parsers.

1.3 Survey of Error-Correcting Parsing

The idea of using syntax rules in correcting program errors or punctuation errors arose with the design of syntax-directed compilers. Because the syntactic specifications are precise, syntax analysis not only plays a central role in the organization of compilers, it also provides error detection and recovery capability within the compiler.



(a)



(b)

Figure 1.2 Syntax analysis using a parser (a) and an error-correcting parser (b)

Most error-recovery strategies take the point where parsing fails to continue as the point of detection of errors, [15-21]. A recovery action is then applied to suppress the error so that parsing can be resumed. Diagnostic information or corrections may also be generated at this point. In [15], Ironsattempts to supply some recovery action at each point where an inconsistency is detected by a top-down parser. Part of the input sentence is replaced based on the context of the error and productions of the grammar to allow parsing to continue. Similar to Iron's idea, Gries [16] proposes more sophisticated error-recovery strategies for top-down parsers. Using a precedence grammar, an error is detected when no precedence relation exists between the incoming terminal and the symbol at the top of the stack, or the phrase to be reduced has no equivalent right hand side of a production. Wirth [17] proposes an error recovery algorithm by scanning tables of error rules for an entry which applies to the erroneous condition. The table of error rules is based on designers' knowledge of common programming errors and appropriate recovery actions. Leinius [18] discusses strategies of isolating the smallest potential phrase and then makes the required replacement. Graham and Rhodes [19], suggest a weighted minimum distance measure for finding a "closest fit" local corrections when an error recovery routine faces numerous choice of next move.

Error-correcting for compilers emphasizes early detection of errors and generating accurate diagnosis. Nevertheless, error detection may arbitrarily be delayed. In a study by Aho and Ullman [23], they concluded that a precedence parser will not always detect an error as soon as a corresponding LR(1) parser. Repairing techniques for LR and LL parsers are still subjects to be studied.

An alternative to the heuristic approach for error-correcting parsing is the minimum-distance error-correcting techniques. All the potential errors and their corrections are recorded during parsing. After the entire string has been processed, a derivation that satisfies the minimum-distance criterion will be generated. Aho and Peterson formulate substitution, deletion and insertion errors in terms of error transformations [13]. Distance between two sentences is defined as the least number of transformations used to derive one from the other. The original grammar is then expanded to include error transformations in the set of production rules such that its generated language contains all the possible erroneous sentences. The parsing algorithm is a modified Earley's parser [34] with provisions added for the bookkeeping of the number of error transformations used. During parsing, the potential derivation that uses the least number of error production rules is placed in the parse list. By the time a parsing is completed, the minimum-distance correction is also achieved. Instead of formulating error production rules, Lyon proposes a scheme that puts all the possible corrections such as the substitutions, deletions and insertions of the currently scanned symbol in the parse list [24]. Setting limitations on the number of local errors and global errors are also suggested by Lyon to decrease the parsing time and memory storage.

Although the minimum-distance error-correcting parsers (MDECP) use a similarity criterion in their searching for the syntactically correct sentence they are considered impractical from compiler design point of view. Compiler designers' interests are in methods that generate accurate diagnostic information and continue parsing more than that of automatic correction, since part of the debugging and correction can be left to the programmer.

A spelling error correction technique is proposed by Morgan [25]. He uses a heuristic method to search for a good match of the input string from a table of code words. A more rigorous approach in finding a best match from a finite set of strings for an input string is to use the algorithm given by Wagner and Fisher [26]. Error-correcting parsers for regular language are proposed by Wagner [27] and Thomason [28] respectively.

The second interesting application of error-correcting parsing is in syntactic decoding systems where errors are caused from noisy communication channels [29-32]. In modeling the randomness of noisy channels, it is essential that the designed probabilistic model can be applied to the syntactic processing of linguistic information. Bahi and Jelinek [30], proposed a first order Markov chain model for noisy channels in which an input sequence, $a_1a_2...a_n$ can produce output sequence $b_1b_2...b_m$ of varying lengths. This is done by associating with each input symbol a_k a probabilistic finite state machine. A null transition, self loop transitions and transitions producingoutput other than a_k are added to the finite state machine for modeling deletion, insertion and substitution errors. In Fung and Fu's probabilistic deformation model, context-free languages with substitution errors are considered [31]. Let $x = a_1a_2...a_n$ be a string, the error occurred on a symbol, a_k is assumed to be independent from its context in x. Therefore, the probability that $y = b_1b_2...b_n$ is an error deformed string of x is defined as follows

$$q(y|x) = \prod_{i=1}^{n} q(b_i|a_i)$$

where $q(b_i|a_i)$ is the probability that b_i substitutes for a_i. Let $L(G_1)$ and $L(G_2)$ be two languages, the maximum-likelihood decision rule proposed by Fung and Fu is

$$\max_{x \in L(G_1)} q(y|x) \gtrless \max_{x \in L(G_2)} q(y|x) \text{ decide } \begin{matrix} y \in L(G_1) \\ y \in L(G_2) \end{matrix}$$

Given a grammar G, the parser designed for the searching of x that satisfies the maximum-likelihood criterion, $\max_{x \in L(G)} q(y|x)$, where y is an input string, is called maximum-likelihood error-correcting parser (MLECP). A modified Cocke-Younger-Kasami parser is used by Fung and Fu. Thompson [32] formulates error-correcting parsers for stochastic grammar in Greibach normal form (GNF). The approach of using expanded grammars in which substitution, deletion, insertion and transposition errors are added as error production rules is also used here. The four types of errors are treated seperately. Four separate algorithms each of which copes with one type of error are then combined to correct simultaneous errors. Although Thompson gives no discussion on complexity of this ECP, he points out that the original top-down parser for GNF has already exponential growth in computational complexity. Ambiguity caused by the four expanded grammars is expected to increase this complexity. The combined algorithm would further compound time complexity with the enormity of the bookkeeping problem. It is suggested to combine the algorithm serially to simplify the process. However, the use of serial combination may not always give maximum-likelihood correction.

The idea of using error-correcting parsers in the syntactic recognition of noisy patterns has also been suggested [31,33]. Since pattern recognition systems handle a variety of types of input data, such as pictorial data [35-39], waveforms [40-43], speech patterns, program schemes [44-45], or data files [46] etc., the metalanguages used to describe patterns can be sets of strings, trees or graphs. In addition to type I,

type 2, and type 3 grammars, programmed grammars [47-48], transition networks [49], tree grammars [50], graph grammars [45,51], web grammars [52,53], array grammars [54-56], and many others have been used for pattern analysis. An error-correcting parsing scheme for context-sensitive grammars is proposed by Tanaka and Fu [57].

Error-correcting parsing for syntactic pattern recognition is still at a beginning stage. Most existing pattern grammars do not have their corresponding error-correcting parsers. Similarity measure is a key point in designing such a parser for a pattern recognition system. The idea of using language transformations for the modeling of primitive extraction and segmentation errors, and constructing a parser based on the expanded grammar which includes transformation rules, provides a global distance measurement. We shall use this approach in formulating error-correcting parsers for stochastic and non-stochastic context-free languages and tree languages. A minimum-distance error-correcting parser for context-free program grammar is also presented.

1.4 Thesis Organization

In Chapter 2, the distance between two strings is measured in terms of the three defined transformations, namely, substitution, deletion and insertion transformations. This measurement provides a similarity measure between syntactic patterns. Error-correcting parsers are formulated based on a similarity criterion; e.g. the minimum-distance criterion. Definitions on distance between a string and a language are proposed. A minimum-distance classification system using a modified error-correcting parsing algorithm is presented. A similar approach is applied to a stochastic model where stochastic languages, deformation probabilities and maximum-likelihood criterion are used.

The problem of error-correcting syntax analysis for tree languages is studied in Chapter 3. Syntax errors on trees are defined in terms of five types of transformation, namely, substitution, deletion, stretch, split and branch. In the formulation of error-correcting tree automata (ECTA), transformations made on each terminal symbol are added to the automata in the form of transition functions. Two types of ECTA are proposed: one for substitution errors called SPECTA and one for all five types of errors called GECTA. Real data examples of using SPECTA for LANDSAT data interpretation and GECTA for character recognition are presented.

Chapter 4 and Chapter 5 describes two potential applications of error-correcting parsers in the area of pattern recognition. As the distance between a sentence (a syntactic pattern) and a language (a group of syntactic patterns) is defined, and its computation is implemented by using error-correcting parsers, a clustering procedure for syntactic patterns is proposed in Chapter 4. A character recognition experiment is given as an illustrative example.

A syntactic model for the generation and the discrimination of structured textures is described in Chapter 5. A texture pattern is first divided into fixed-sized windows. Windowed patterns belonging to the same class of texture are then characterized by a tree grammar. The uncertainty existing in texture patterns; e.g. local noise, structure distortion, makes them impossible to be fully characterized by the constructed grammars. Therefore, SPECTA's are used as texture discriminators. Texture patterns generated by tree grammars are illustrated. Discrimination results are also given.

A short summary of this research and suggestions on further work can be found in Chapter 6.

CHAPTER 2

ERROR-CORRECTING PARSING FOR STRING LANGUAGES

2.1 Introduction

In this chapter, a distance between two strings is first defined and then extended to the distance between a string and a language. The distance between two strings is defined in terms of the minimum number of error transformations used to derive one from the other by Aho and Peterson [13]. When the error transformations are defined in terms of substitution, deletion and insertion errors, the distance measurement coincides with the definition of Levenshtein metric [61]. In Section 2.2, error transformations are applied to weighted Levenshtein metric. Also, a new metric, simply called weighted metric, which would reflect the difference of the same type of error made on different terminals is proposed. This extension provides a similarity measure between two sentences more closely related to the similarity of their corresponding patterns.

For a given input string y and a given grammar G, a minimum-distance error-correcting parser (MDECP) is an algorithm that searches for a sentence z in L(G) such that the distance between z and y, d(z,y) is the minimum among the distances between all the sentences in L(G) and y. The algorithm also generates the value of d(z,y). We simply define this value to be the distance between L(G) and y and denote it as $d_1(L(G),y)$.

When a given grammar is a context-free grammar (CFG), its MDECP can be implemented by modifying an Earley's parsing algorithm. We also extend

the definition of the distance between L(G) and y, $d_1(L(G),y)$, to the definition of $d_K(L(G),y)$, the average distance between y and the K sentences in L(G) that are the nearest to y. The computation of $d_K(L(G), y)$ can be impiemented by further modification of the algorithm of MDECP. In Section 2.2.3 a minimum-distance decision rule is proposed for classification of syntactic patterns.

An aigorithm of MDECP for context-free programmed grammars (CFPG) is given in Section 2.3. This algorithm is restricted to Levenshtein's distance oniy. In pattern recognition, the context-free programmed grammars are considered having higher descriptive power than the context-free grammar. It is proved by Rosenkrantz, that the set of ianguages generated by CFPG's properiy contains the set of context-free languages, and is properly contained within the set of context-sensitive languages [60]. A CFPG generating a context-sensitive language may be selected in order to describe the patterns effectively. Aithough a context-sensitive grammar (CSG) may as weii be used, the parsing based on a CFPG has better anaiysis efficiency than a CSG.

In Section 2.4, the stochastic deformation modei of substitution errors proposed by Fung and Fu [31] is first extended to inciude deietion and insertion errors. Based on the deformation model, the deformation probabiiities can be estimated from the observations of these errors. Similar to the use of error transformations proposed in [13], the stochastic deformation model is introduced into the original stochastic context-free grammar (SCFG). The Eariey's parser is modified for the searching of the most likely error-correction based on the maximum-likelihood criterion. We shaii call this algorithm the maximum-likelihood error-correcting parser

(MLECP). For an input sentence y and a given SCFG, G_s, a MLECP generates the most likely correction of y, x. Let p(x) be the probability of x in $L(G_s)$. The MLECP also computes the value of $q(y|x)p(x)$ where $q(y|x)$ is the deformation probability of y given x. We shall interpret the term, $q(y|x)p(x)$, as the deformation probability of y given $L(G_s)$, denoted as $q(y|G_s')$. If the a priori probability of each grammar is known, Bayes' decision rule is proposed as a decision criterion.

Due to the inefficiency of such an error-correcting parser, we are also interested in the improvement of parsing speed. Persoon and Fu have proposed a sequential classification algorithm (SCA) for stochastic context-free languages [10]. The error-tolerance capability of SCA is investigated in Section 2.4.4. We further modify the SCA to classify noisy sentences using the error-correcting approach. Experimental results illustrate that within a tolerable percentage of misrecognition, the speed of SCA is faster than that of MLECP.

2.2 Minimum-Distance Error-Correcting Parsing for Context-Free Languages

Following the notations used in [14], the definition of grammars and languages is briefly reviewed.

Definition 2.1. A grammar is a 4-tuple

$G = (N,\Sigma,P,S)$ where

(1) N is a finite set of nonterminal symbols

(2) Σ is a finite set of terminal symbols disjoint from N.

(3) P is a finite subset of

$$(N\cup\Sigma)^*N(N\cup\Sigma)^*X(N\cup\Sigma)^*$$

An element (α,β) in P will be written $\alpha \rightarrow \beta$ and called a production.

(4) S is a distinguished symbol in N called the start symbol.

Definition 2.2. The language generated by a grammar G, denoted L(G), is the set of sentences generated by G. Thus,

$$L(G) = \{\omega | \omega \text{ is in } \Sigma^*, \text{ and } S \overset{*}{\Longrightarrow} \omega\}$$

where a relation $\Rightarrow$ on $(N \cup \Sigma)^*$ is defined as follows: if $\alpha\beta\gamma$ is in $(N \cup \Sigma)^*$ and $\beta \rightarrow \delta$ is a production rule in P then $\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$, and $\overset{*}{\Longrightarrow}$ denotes the reflexive and transitive closure of $\Rightarrow$.

If each production in P is of the form $A \rightarrow \alpha$, where A is in N and α is in $(N \cup \Sigma)^*$ then the grammar $G = (N, \Sigma, P, S)$ is classified as context-free grammar. The set of languages that can be generated from context-free grammars are called context-free languages.

2.2.1 A Similarity Measure for Syntactic Patterns

In [13], errors in a string are considered to be the three types: substitution, deletion and insertion errors, and treated as syntax errors by defining transformations from Σ^* to a subset of Σ^*.

Definition 2.3. For two strings, $x, y \in \Sigma^*$, we can define a transformation $T: \Sigma^* \rightarrow \Sigma^*$ such that $y \in T(x)$. The following three transformations are introduced:

(1) substitution error transformation

$$\omega_1 a \omega_2 \overset{T_S}{\longmapsto} \omega_1 b \omega_2, \text{for all } a, b \in \Sigma, a \neq b,$$

(2) deletion error transformation

$$\omega_1 a \omega_2 \overset{T_D}{\longmapsto} \omega_1 \omega_2, \text{for all } a \in \Sigma$$

(3) Insertion error transformation

$$\omega_1\omega_2 \xrightarrow{T_I} \omega_1 a \omega_2, \text{for all } a \in \Sigma$$

where $\omega_1, \omega_2 \in \Sigma^*$

Definition 2.4. The distance between two strings $x, y \in \Sigma^*$, $d^L(x,y)$, is defined as the smallest number of transformations required to derive y from x.

Example 2.1. Given a sentence x = cbabdbb and a sentence y = cbbabbdb, then

$$x = \text{cbabdbb} \xrightarrow{T_S} \text{cbabbbb} \xrightarrow{T_S} \text{cbabbdb} \xrightarrow{T_I} \text{cbbabbdb} = y$$

The minimum number of transformations required to transform x to y is three, thus, $d^L(x,y) = 3$.

The metric defined in Definition 2-4 gives exactly the Levenshtein distance of two strings [61]. A weighted Levenshtein distance can be defined by assigning nonnegative numbers σ, γ and δ to transformations T_S, T_D and T_I respectively. Let $x, y \in \Sigma^*$ be two strings, and let J be a sequence of transformations used to derive y from x, then the weighted Levenshtein distance between x and y, denoted as $d^W(x,y)$ is

$$d^W(x,y) = \min_J \{\sigma \cdot k_J + \gamma \cdot m_J + \delta \cdot n_J\} \qquad (2.1)$$

where k_J, m_J and n_J are the number of substitution, deletion, and insertion transformations respectively in J.

We shall propose a weighted metric that would reflect the difference of the same type of error made on different terminals. Let the weights associated with error transformations on terminal a in a string $\omega_1 a \omega_2$ where

$a \in \Sigma$, ω_1 and $\omega_2 \in \Sigma^*$, be defined as follows:

(1) $\omega_1 a \omega_2 \xvdash{T_S,\ S(a,b)} \omega_1 b \omega_2$ for $b \in \Sigma$, $b \neq a$, where $S(a,b)$ is the cost of substituting a for b. Let $S(a,a) = 0$.

(2) $\omega_1 a \omega_2 \xvdash{T_D,\ D(a)} \omega_1 \omega_2$ where $D(a)$ is the cost of deleting a from $\omega_1 a \omega_2$.

(3) $\omega_1 a \omega_2 \xvdash{T_I,\ I(a,b)} \omega_1 b a \omega_2$ for $b \in \Sigma$, where $I(a,b)$ is the cost of inserting b in front of a.

We further define the weight of inserting a terminal b at the end of a string x to be,

(4) $x \xvdash{T_I,\ I'(b)} xb$, for $b \in \Sigma$.

Let $x, y \in \Sigma^*$ be two strings, and J be a sequence of transformations used to derive y from x. Let $|J|$ be defined as the sum of the weights associated with transformations in J, then the weighted distance between x and y, $d^W(x,y)$ is defined as

$$d^W(x,y) = \min_J \{|J|\} \tag{2.2}$$

Equation (2.2) can be illustrated by a graphical interpretation. From point B to point E, each path in the lattice shown in Figure 2.1 corresponds to a sequence of transformations used to derive y from x. A horizontal branch indicates an insertion transformation, a vertical branch indicates a deletion transformation, and a diagonal branch indicates a substitution or a non-error transformation. The weight assigned to a particular type of transformation on a particular symbol in x is labeled at its corresponding branch. Let J be a path in the lattice, then $|J|$ is the sum of weight associated with each branch in J. The distance,

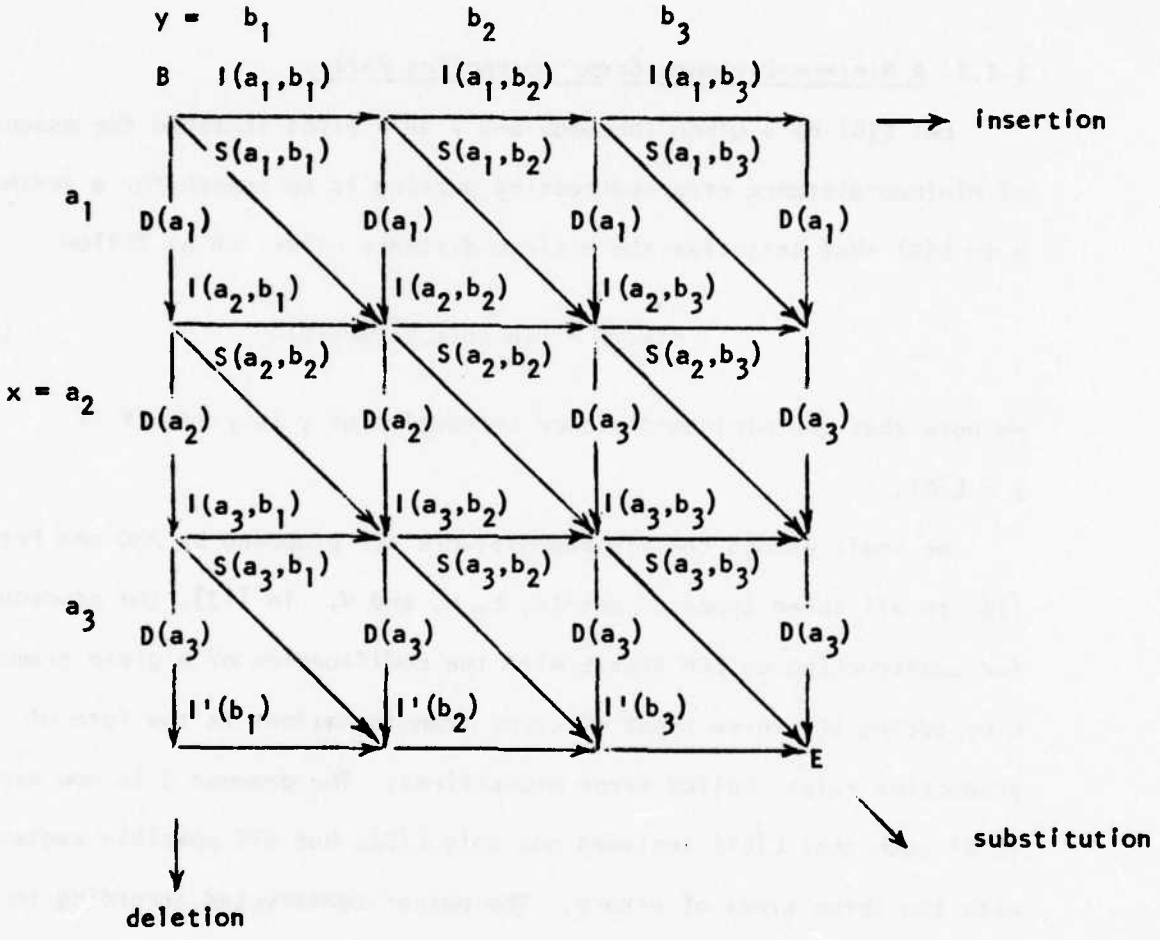


Figure 2.1 A graphic interpretation of metric W

$d^W(x,y)$, is the weight associated with the minumum-weight path.

We shall refer to the Levenshtein distance, weighted Levenshtein distance and weighted distance as metric L, w, and W respectively, and use d(x,y) to denote the distance between x and y based on any of the three metrics.

2.2.2 A Minimum-Distance Error-Correcting Parser

Let L(G) be a given language and y be a given sentence the essence of minimum-distance error-correcting parsing is to search for a sentence x in L(G) that satisfies the minimum distance criterion as follow

$$d(x,y) = \min_{z} \{d(z,y) | z \varepsilon L(G)\} \qquad (2.3)$$

We note that the minimum-distance correction of y is y itself if $y \varepsilon L(G)$.

We shall extend the minimum-distance ECP proposed by Aho and Peterson [13] to all three types of metric; L, w, and W. In [13], the procedure for constructing an ECP starts with the modification of a given grammar G by adding the three types of error transformations in the form of production rules, called error productions. The grammar G is now expanded to G' such that L(G') includes not only L(G), but all possible sentences with the three types of errors. The parser constructed according to G' with a provision added to count the number of error productions used in a derivation is the error-correcting parser for G. For a given sentence y, the ECP will generate a parse, Π, which consists of the smallest number of error productions. A sentence x in L(G) that satisfies the minimum-distance criterion (measured by using Levenshtein distance) can be generated from Π by eliminating error productions. With some modifications, this minimum-distance ECP can easily be extended to the three

metrics proposed in Section 2.2.1. We first give the algorithm of constructing an expanded grammar, in which the nonnegative numbers associated with error-productions are the weights associated with their corresponding error transformations with respect to the metric used.

Algorithm 2.1. Construction of expanded grammar

Input: A CFG $G = (N,\Sigma,P,S)$

output: A CFG $G' = (N',\Sigma',P',S')$ where P' is a set of weighted productions.

Method:

Step 1. $N' = N \cup \{S'\} \cup \{E_a \mid a \in \Sigma\}$, $\Sigma' \supseteq \Sigma$

Step 2. If $A \rightarrow \alpha_0 b_1 \alpha_1 b_2 \ldots b_m \alpha_m$, $m \geq 0$ is a production in P such that $\alpha_1 \in N^*$ and $b_1 \in \Sigma$, then add $A \rightarrow \alpha_0 E_{b_1} \alpha_1 E_{b_2} \ldots E_{b_m} \alpha_m$, 0 to P', where each E_{b_1} is a new non-terminal, $E_{b_1} \in N'$ and 0 is the weight associated with this production.

Step 3. Add the following productions to P'.

Production Rule		weight L	w	W (metric)	
(a)	$S' \rightarrow S$	0	0	0	
(b)	$S' \rightarrow Sa$	1	δ	$1'(a)$	for all $a \in \Sigma'$
(c)	$E_a \rightarrow a$	0	0	0	for all $a \in \Sigma$
(d)	$E_a \rightarrow b$	1	σ	$S(a,b)$	for all $a \in \Sigma$, $b \in \Sigma'$, and $b \neq a$
(e)	$E_a \rightarrow \lambda$	1	γ	$D(a)$	for all $a \in \Sigma$
(f)	$E_a \rightarrow bE_a$	1	δ	$I(a,b)$	for all $a \in \Sigma$, $b \in \Sigma'$

In Algorithm 2.1 the production rules added in Step 3(b), 3(d), 3(e) and 3(f) are called error productions. Each error production corresponds to one type of error transformation on a particular symbol in Σ. Therefore, the distance measured in terms of error transformations can be measured by error productions used in a derivation. The parser is a modified Earley's parsing algorithm with a provision added to accumulate the weights associated with productions used in a derivation. The algorithm is as follow.

Algorithm 2.2. Minimum-distance error-correcting parsing algorithm

input: An expanded grammar $G' = (N', \Sigma', P', S')$ and an input string $y = b_1 b_2 \ldots b_m$ in Σ'^*

output: $I_0, i_1 \ldots I_m$ the parse list for y, and $d(x,y)$ where x is the minimum-distance correction of y.

Method:

Step 1. Set $j = 0$. Then add $[E \rightarrow \cdot\ S', 0, 0]$ to I_j.

Step 2. If $[A \rightarrow \alpha \cdot B\beta, i, \xi]$ is in I_j, and $B \rightarrow \gamma$, η is a production rule in P' then add item $[B \rightarrow \cdot\ \gamma, j, 0]$ to I_j.

Step 3. If $[A \rightarrow \alpha \cdot, i, \xi]$ is in I_j and $[B \rightarrow \beta \cdot A\gamma, k, \xi]$ is in I_i, and if no item of the form $[B \rightarrow \beta A \cdot \gamma, k, \phi]$ can be found in I_j, then add an item $[B \rightarrow \beta A \cdot \gamma, k, \eta+\xi+\zeta)$ to I_j where ζ is the weight associated with production $A \rightarrow \alpha$. If $[B \rightarrow \beta A \cdot \gamma, k, \phi]$ is already in I_j, then replace ϕ by $\eta+\xi+\zeta$ if $\phi > \eta+\xi+\zeta$.

Step 4. If $j = m$ go to Step 6, otherwise $j = j+1$.

Step 5. For each item in I_{j-1} of the form $[A \rightarrow \alpha \cdot b_j\beta, i, \xi]$ add item $[A \rightarrow \alpha b_j \cdot \beta, i, \xi]$ to I_j, go to Step 2.

Step 6. If item $[E \rightarrow S', 0, \xi]$ is in I_m. Then $d(x,y) = \xi$, where x is the minimum-distance connection of y, exit.

In Algorithm 2.2, the string x, which is the minimum-distance correction of y, can be derived from the parse of y by eliminating all the error productions. The extraction of the parse of y is the same as that described in Earley's algorithm.

2.2.3 A Minimum-Distance Classifier for Noisy Patterns

In Section 2.2.1, three metrics are proposed as similarity measures between two strings. We shall define the distance between a string and a given language based on any one of the three metrics as follows.

Definition 2.5. Let y be a sentence, and L(G) be a given language, the distance between L(G) and y, $d_K(L(G),y)$, where K is a given positive integer, is;

$$d_K(L(G),y) = \min\{\sum_{i=1}^{K} \frac{1}{K} d(z_i,y) \mid z_i \in L(G)\} \tag{2.4}$$

In particular, if K = 1, then

$$d_1(L(G),y) = \min\{d(z,y) \mid z \in L(G)\} \tag{2.5}$$

is the distance between y and its minimum-distance correction in L(G).

As the distance between a string (a syntactic pattern) and a language (a set of syntactic patterns) is defined, a minimum-distance decision rule can be stated as follows: suppose that there are two classes of patterns, C_1 and C_2 characterized by grammar G_1 and G_2 respectively. For a given syntactic pattern y with unknown classification,

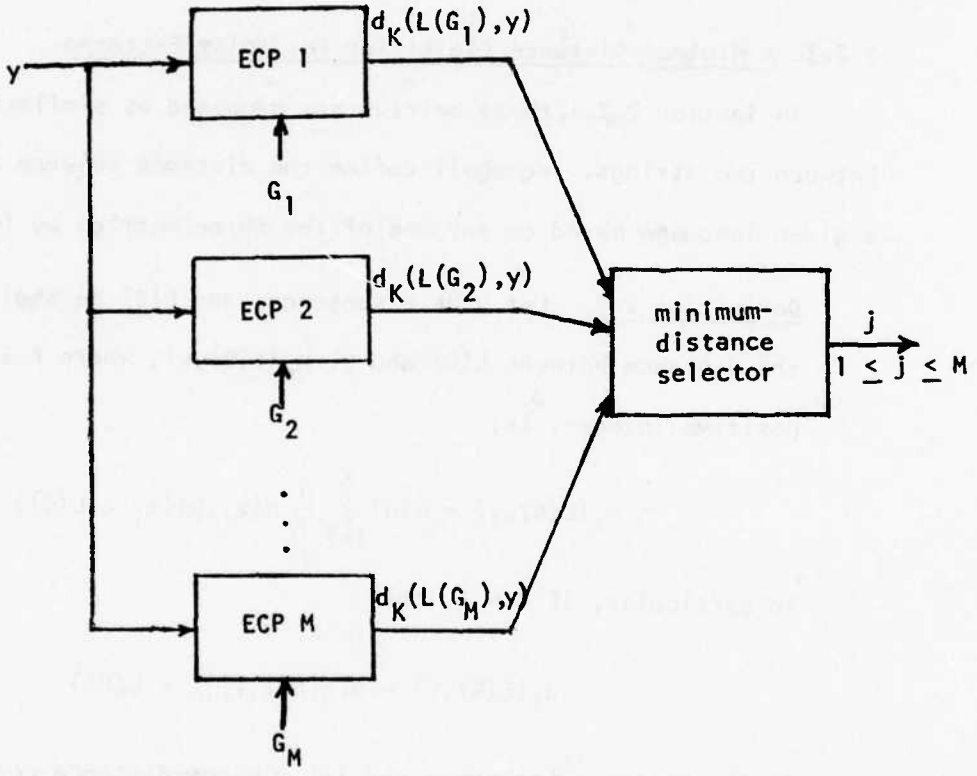


Figure 2.2 A minimum-distance classifier for noisy patterns.

$$\text{decide } y \in \begin{matrix} C_1 \\ C_2 \end{matrix} \text{ if } d_K(L(G_1),y) \lesseqgtr d_K(L(G_2),y) \qquad (2.6)$$

A block diagram of a minimum-distance classification system is given in Figure 2.2.

For a given L(G) and a string $y = b_1b_2\ldots b_m$, the MDECP described in Algorithm 2.2 also generates the distance $d_1(L(G),y)$. For the computation of $d_K(L(G),y)$, $K > 1$, Algorithm 2.2 needs further modification. The following algorithm will generate parse lists in which each item in the list I_j is of the form $[A \to \alpha \cdot \beta, i, (n_1,\pi_1),(n_2,\pi_2)\ldots(n_e,\pi_e)]$ where $e \leq K$. Each pair (n_k,π_k), for $1 \leq k \leq e$, n_k is the weight associated with a derivation of substring $b_{i+1}\ldots b_j$, and π_k is the corresponding correction of $b_{i+1}\ldots b_j$ from this derivation.

Algorithm 2.3. Computation of $d_K(L(G),y)$

Input: An expanded grammar $G' = (N',\Sigma',P',S')$, an input string $y = b_1b_2\ldots b_m$ in Σ', and K, a given positive integer.

output: $d_K(L(G),y)$ and $x_1x_2\ldots x_K$ the K nearest strings to y in L(G).

Method:

Step 1. Add item $[E \to \cdot S',0,\phi]$ to I_0. Set $j = 0$.

Step 2. If $[A \to \alpha \cdot B\beta, i, (n_1,\pi_1)\ldots(n_e,\pi_e)]$ is in I_j and $B \to \gamma$ is a production rule in P', then add item $[B \to \cdot \gamma, j, \phi]$ to I_j.

Step 3. If $[A \to \alpha \cdot B\gamma, h, (n_1,\pi_1)\ldots(n_e,\pi_e)]$ is in I_i and $[B \to \beta \cdot, l, (m_1,\tau_1)\ldots(m_f,\tau_f)]$ is in I_j,

(a) If item of the form $[A \to \alpha B \cdot \gamma, h, (l_1',n_1')\ldots(l_g',n_g')]$ cannot be found in I_j, then add $[A \to \alpha\, B \cdot \gamma, h, (l_1,n_1)\ldots(l_g,n_g)]$ to I_j, where each pair are chosen from the set N,

$N = \{(\xi+n_p+m_q,\ \pi_pX) \mid 1 \leq p \leq e,\ 1 \leq q \leq f$, and $X = B$ if

$B \rightarrow \beta$, ξ is an error production, or $X = \tau_q$, otherwise}, such that $i_1 \leq i_2 \leq \ldots \leq i_g$ are the g smallest left hand side numbers having distinctive right hand side in N. $g = |N|$, if $|N| < K$ or $g = K$ if $|N| \geq K$ where $|N|$ is the number of elements in the set N.

(b) if item of the form $[A \rightarrow \alpha B \cdot \gamma, h, (i_1', \eta_1') \ldots (i_g', \eta_g')]$ is already in I_j, then rearrange the set of pairs $(i_1', \eta_1') \ldots (i_g', \eta_g')$ to be $(i_1, \eta_1) \ldots (i_g, \eta_g)$ such that $\eta_1, \eta_2 \ldots \eta_g$ are g distinctive strings and $i_1 \leq i_2 \leq \ldots \leq i_g$ are g smallest number in the set $N' = NU\{(i_1', \eta_1'), (i_2', \eta_2') \ldots (i_g', \eta_g')\}$ and $g = |N'|$ if $|N'| < K$, or $g = K$ if $|N'| \geq K$.

Step 4. Repeat Step 3 until no new item can be found. Then if j=m, go to Step 6, otherwise j=j+i.

Step 5. For each item $[A \rightarrow \alpha b_j \cdot \beta, i, (n_1, \pi_1) \ldots (n_e, \pi_e)]$ in I_{j-1}, add $[A \rightarrow \alpha b_j \cdot \beta, i, (n_1, \pi_1) \ldots (n_e, \pi_e)]$ to I_j. Go to Step 2.

Step 6. If item $[E \rightarrow S' \cdot , 0, (n_1, \pi_1) \ldots (n_K, \pi_K)]$ is in I_m, then $d_k(L(G), y) = \sum_{i=i}^{K} \frac{i}{K} n_i$.

The pairs (n_k, π_k), $i \leq k \leq K$ in the items of the parse lists are added for bookkeeping of the K nearest strings L(G) to the input y. During the derivation of substring $b_{i+1} \ldots b_j$, the corresponding corrected string are recorded such that identical strings caused by ambiguity of the grammar will not appear in the final set of the K nearest strings.

2.3. Error-Correcting Parsing for Context-Free Programmed Languages

2.3.1. Context-Free Programmed Grammar

Definition 2.6. A context-free programmed grammar (CFPG) is a 5-tuple $G = (N,\Sigma,S,P,J)$ where

(1) N is a finite set of non-terminals

(2) Σ is a finite set of terminals

(3) S is the start symbol in N

(4) P is a finite set of programmed productions

(5) J is a finite set of production lables

Each production in P consists of a label $r \in J$, a core production of the from $A \rightarrow \alpha$ where $A \in N$, $\alpha \in (N \cup \Sigma)^*$, and a success branch field and a failure branch field each consisting of elements from J.

A derivation or generation in G proceeds as follows: the first production is applied to the start symbol S; therefore, if production r is applied to the current sentential form γ to rewrite a nonterminal A, and if γ contains at least one occurrence of A, then the leftmost A is rewritten by the core of production r and the next production label is selected from the success branch field of r; if the current sentential form does not contain x, then the core of production r cannot be used and the next production label is selected from the failure branch field of r; if the applicable branch field is empty, the derivation halts.

Example 2.2. Consider the CFPG $G_p = (N,\Sigma,A,P,J)$ where $N = \{A,B,C\}$, $\Sigma = \{a,b,c\}$, $J = \{1,2,3,4,5\}$ and P:

label	core	S(U)	F(W)
1	A → aBC	2,4	ϕ
2	B → aBB	3	ϕ
3	C → CC	2,4	ϕ
4	B → b	4	5
5	C → c	5	ϕ

$L(G_p) = \{a^n b^n c^n \mid n \geq 1\}$

A syntax analysis procedure for CFPG's has been proposed by Swain and Fu [48]. The algorithm can be explained by using the following example.

Example 2.3. Figure 2.3 is a schematic diagram of the analysis of the string abc with respect to grammar G_p. The notations used are explained as follows:

(1) For any nonterminal $A \in N$, $A_i(\gamma)A_i$ indicates that the string γ was generated from the nonterminal A which was rewritten as γ at the ith step.

(2) A downward arrow indicates a generative step. An upward arrow indicates backtracking. Backtracking is occurred when a generated sentential form is incompatible with input string. That is, the parsing detects at this step that if the analysis continues along the current path (or derivation), the string generated will be different from the input.

(3) The branch labels have the form $\xi_k(\ell) = t$, where ξ is S (success) or F (failure); subscript k indicates the application of the kth production; ℓ indicates the selection of the ℓth branch in the

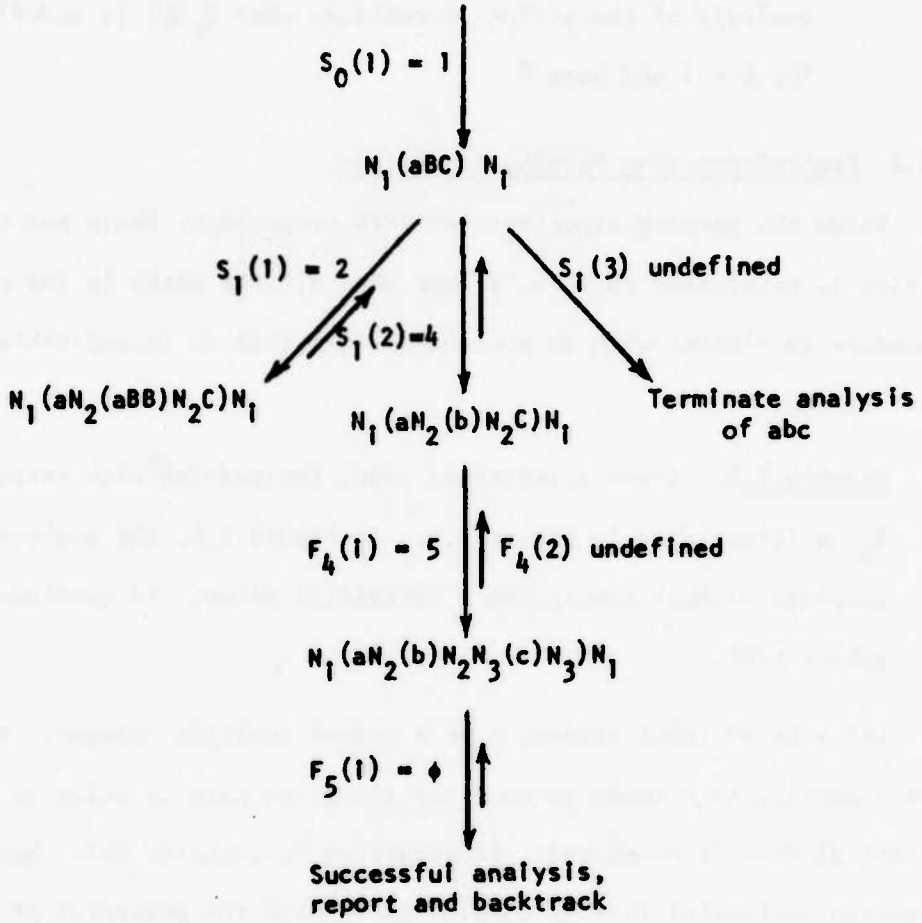


Figure 2.3 Analysis of abc with respect to G_p.

ξ field, which is the branch to the production labeled t. By definition, $\xi_k(\ell) = \phi$ indicates the termination of a path down the tree, which constitutes a successful parsing if the current sentential form is identical to the string being analyzed. The analysis of the string is complete when $\xi_k(\ell)$ is undefined for k = 1 and some ℓ.

2.3.2 Error-Correcting Parsing Algorithms

Using the parsing algorithm for CFPG proposed by Swain and Fu, a parsing is considered to be a failure when all the paths in the analysis procedure terminates with an sentential form that is incompatible with the input string.

Example 2.4. Given a sentence, aabc, the parsing with respect to G_p is illustrated in Figure 2.4. In Figure 2.4, the analysis is complete without generating a successful parse. It concludes that aabc $\notin$ L(G).

Let y be an input string, n be a preset positive integer. Suppose that a parsing is allowed to continue along the path in which an incompatible sentential form is generated. Backtracking is occurred only when a generated sentential form is considered to have the potential of generating a string with at least a distance n from the input string. Using this method, a syntactically correct sentence x can be generated such that $d^L(x,y) \leq n$. A complete parsing procedure starts with n=0. If the analysis based on n fails, then n is increased by one and the analysis procedure is repeated, otherwise parsing is completed and $d^L(x,y) = n$. The algorithm is described as follows.

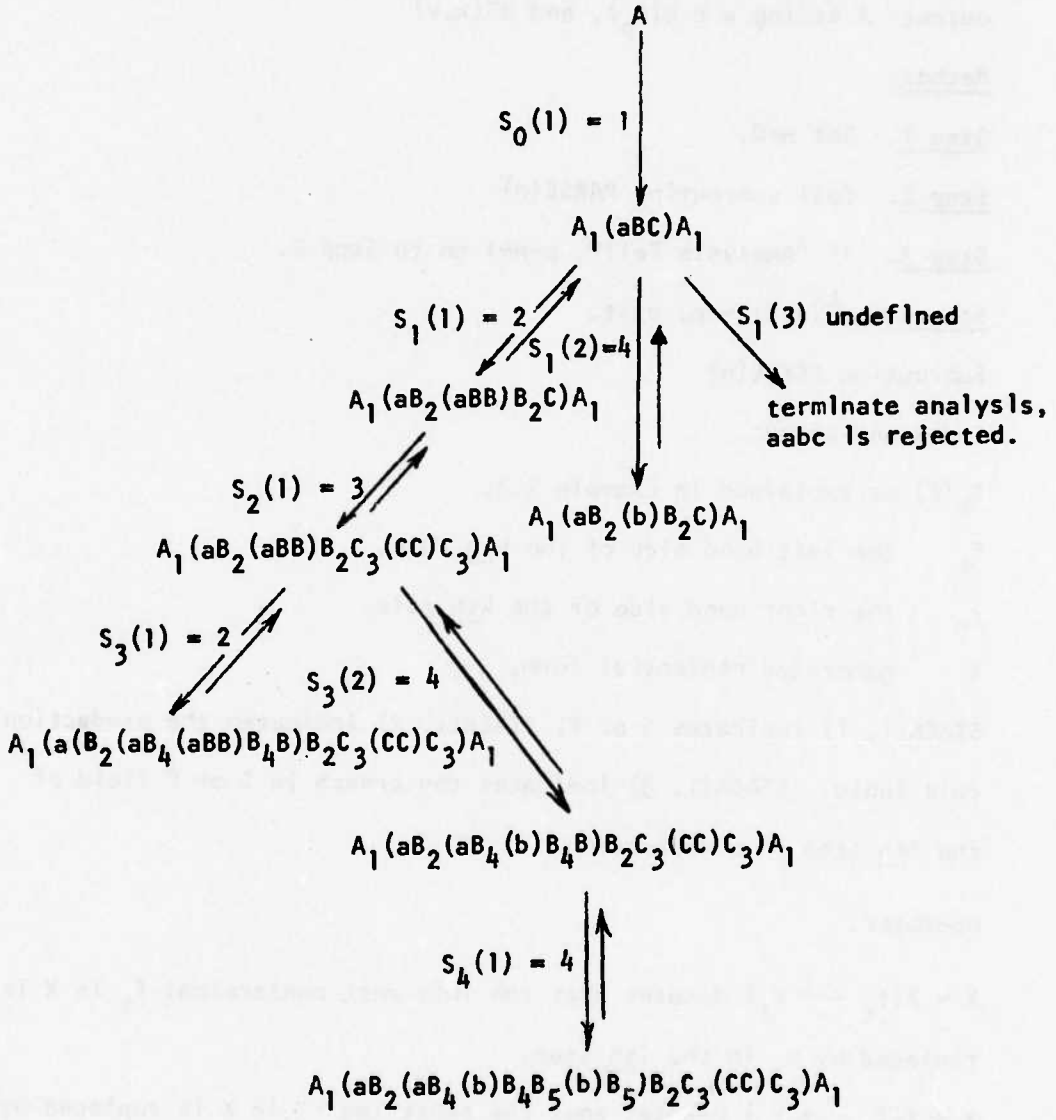


Figure 2.4 The syntax analysis of aabc with respect to G_p.

Algorithm 2.4. MDECP for CFPG

Input: A CFPG $G_p = (N,\Sigma,S,P,J)$ and an input string $y = b_1 b_2 \ldots b_m$.

output: A string $x \in L(G_p)$, and $d^L(x,y)$

Method:

Step 1. Set n=0.

Step 2. Call subroutine PARSE(n)

Step 3. If "Analysis Fail", n=n+1 go to Step 2.

Step 4. $d^L(x,y) = n$, exit.

Subroutine PARSE(n)

index and array;

$\xi_k(\ell)$ as explained in Example 2.3,

f_k the left hand side of the kth rule,

r_k the right hand side of the kth rule,

X generated sentential form,

STACK(i, 1) indicates S or F, STACK(i, 2) indicates the production rule lable. STACK(i, 3) indicates the branch in S or F field of the ith step in a derivation.

operator:

$X = X(f_t \xleftarrow{i} r_t)$ denotes that the left most nonterminal f_t in X is replaced by r_t in the ith step.

$X = X(f_t \xrightarrow{i} r_t)$ denotes that the substring r_t in X is replaced by f_t, where r_t is placed in X at the ith step.

Method:

Step 1. Set k=1, i=0, ℓ=1, ξ=S and $X=r_k$.

Step 2. If $\xi_k(\ell)$ is undefined then go to Step 5, otherwise let $t=\xi_k(\ell)$. If ξ=F, then go to Step 4.

Step 3. If f_t cannot be found in X, then ξ=F, ℓ=1 and $t=F_k(\ell)$.

Step 4. Let i=i+1, STACK(i, 1)=ξ, STACK(i, 2)=k, STACK(i, 3)=ℓ. Let $X = X(f_t \xleftarrow{i} r_t)$. Call subroutine COMPAT(n, X). If "Compatible," and if X is a terminal string, then "Analysis Success" Exit. otherwise k=t, ℓ=i, go to Step 2. If "Incompatible" go to Step 5.

Step 5. If i=0 then "Analysis Fail", exit. Otherwise $X = X(f_t \xrightarrow{i} r_t)$, ξ=STACK(i, 1) k=STACK(i, 2), ℓ=STACK(i, 3)+i, i=i-1 go to Step 2.

Subroutine COMPAT(n, X)

index and array;

M = |X|, the length of sentential form X. Assume that $X = B_1B_2\ldots B_M$. D(i, j), stores the number of potential errors between $B_1B_2\ldots B_i$ and $b_1b_2\ldots b_j$.

Method:

Step 1. D(0, 0)=0

Step 2. Do i=1 to M

D(i, 0) = D(i-1, 0)+1 if B_i is a terminal

D(i, 0) = D(i-1, 0) if B_i is a nonterminal

Step 3. Do j=1 to m

D(0, j) = D(0, j-1)+1

Step 4. Do i=1 to M, D_0 j=1 to m

(a) if B_i is a terminal and $B_i \neq b_j$ then m_1 = D(i-1, j-1)+1 otherwise m_1 = D(j-1, j-1).

(b) if B_i is a terminal then m_2 = D(i-1, j)+1 otherwise m_2 = D(i-1, j).

(c) if B_1 is a terminal then $m_3 = D(i, j-1)+1$ otherwise $m_3 = D(i, j-i)$.

$D(i, j)=\min (m_1,m_2,m_3)$

Step 5. If $D(M, m) > n$ then it is "Incompatible" otherwise "Compatible". Exit.

Example 2.5. Let the string aabc be parsed by Algorithm 2.4. The result of the first analysis (n=0) is shown in Figure 2.4. Since "Analysis Fail" is reported, the algorithm then increases n by one. Figure 2.5 describes the result of the second analysis where an "Analysis Success" is reported. The algorithm generates the corrected string "abc", $d^L(abc, aabc)=1$.

2.4. Error-Correcting Parsing on a Stochastic Model

Basic notations and definitions of stochastic context-free grammar (SCFG) given in [62-63] are briefly reviewed.

Definition 2.7. A SCFG is a 4-tupies $G_s = (N,\Sigma,P_s,S)$ where,

N is a finite set of non-terminals,

Σ is a finite set of terminals,

P_s is a finite set of stochastic productions, each of which is of the form:

$$A_i \xrightarrow{p_{ij}} \alpha_{ij},\ j = 1,2,\ldots,n_i,\ i = 1,2,\ldots,\ell$$

where n_i is the number of productions with A_i at left-hand side, ℓ is the number of non-terminals, $A_i \in N$, $\alpha_{ij} \in (N \cup \Sigma)^*$, and p_{ij} is the probability associated with this production. Furthermore,

$$0 < p_{ij} \leq 1 \text{ and } \sum_{j=1}^{n_i} p_{ij} = 1$$

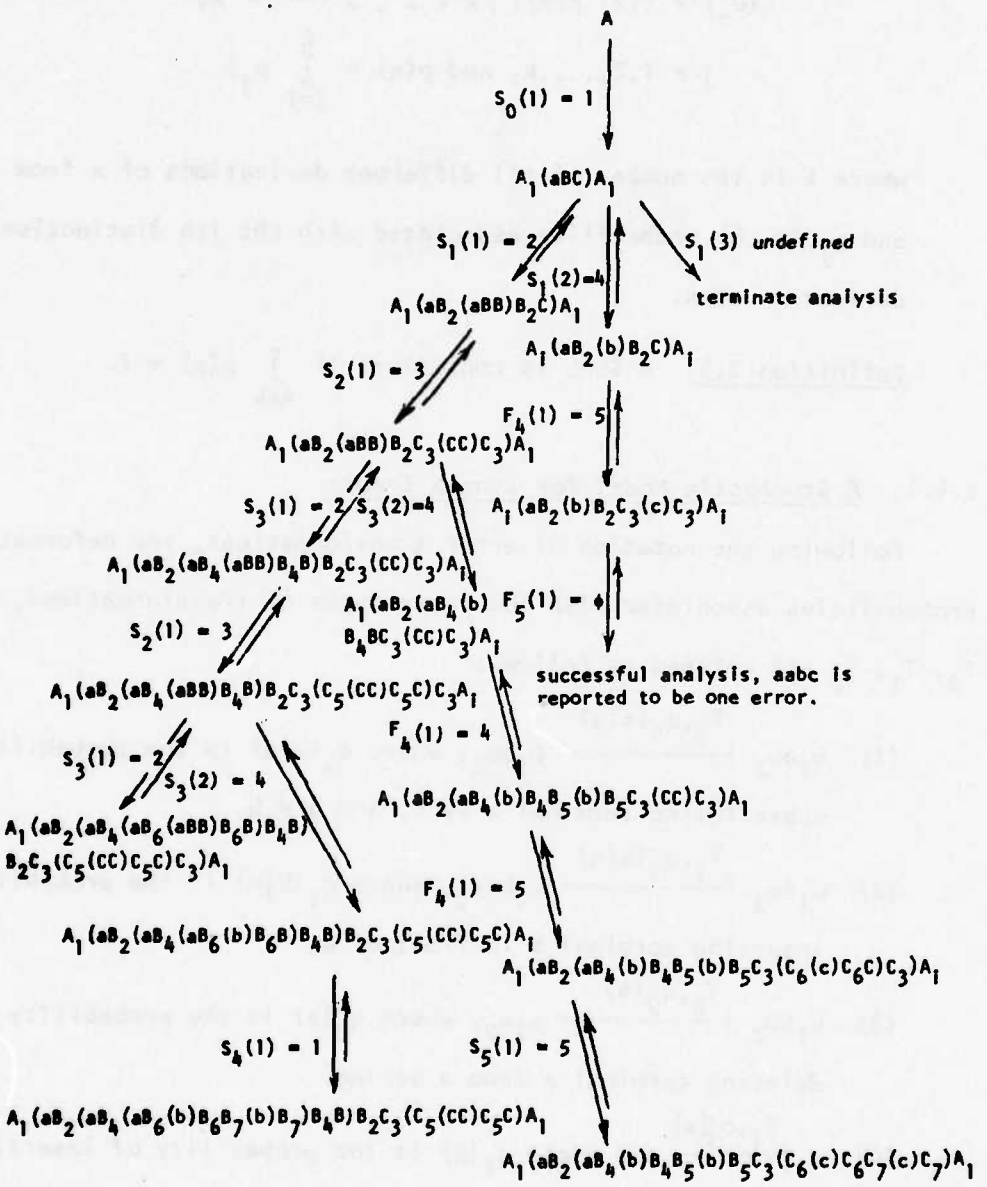


Figure 2.5 The analysis of aabc when n=1

Definition 2.8. The stochastic context-free language (SCFL) generated by SCFG G_s is

$$L(G_s) = \{(x, p(x)) \mid x \in \Sigma^*, S \xrightarrow{p_j} x, j = 1,2,\ldots,k, \text{ and } p(x) = \sum_{j=1}^{k} p_j\}$$

where k is the number of all different derivations of x from S, and p_j is the probability associated with the jth distinctive derivation of x.

Definition 2.9. A SCFL is consistent if $\sum_{x \in L} p(x) = 1$.

2.4.1. A Stochastic Model for Syntax Errors

Following the notation of error transformations, the deformation probabilities associated with the three types of transformations, namely, T_S, T_I, T_D are defined as follows:

(1) $\omega_1 a \omega_2 \xrightarrow{T_S, q_S(b|a)} \omega_1 b \omega_2$, where $q_S(b|a)$ is the probability of substituting terminal a by b, and $a \neq b$,

(2) $\omega_1 a \omega_2 \xrightarrow{T_I, q_I(b|a)} \omega_1 b a \omega_2$, where $q_I(b|a)$ is the probability of inserting terminal b in front of a,

(3) $\omega_1 a \omega_2 \xrightarrow{T_D, q_D(a)} \omega_1 \omega_2$, where $q_D(a)$ is the probability of deleting terminal a from a string.

(4) $x \xrightarrow{T_I, q_I'(a)} xa$, where $q_I'(a)$ is the probability of inserting terminal a at the end of a string.

Let $q_S(a|a)$ be the probability that no error occurs on terminal a, which could be interpreted as the probability of the non-error transformation on a. Assume that for each terminal symbol there can be at

most one error existing. The deformation probabilities on this single-error model is consistent if

$$\sum_{b\in\Sigma} q_S(b|a) + q_D(a) + \sum_{b\in\Sigma} q_I(b|a) = 1 \qquad (2.7)$$

$$\text{for all } a \in \Sigma$$

Let $\alpha \in \Sigma^*$ be a substring the probability that symbol a is deformed to α, denoted $q(\alpha|a)$, is defined as follows:

$$q(\alpha|a) = \begin{cases} q_D(a) & \text{if } \alpha = \lambda \\ \max\{q_S(b|a), q_I(b|a)q_D(a)\} & \text{if } \alpha = b \\ q_I(b_1|a) \ldots q_I(b_{\ell-1}|a) \max[q_S(b_\ell|a), q_I(b_\ell|a)q_D(a)] & \\ & \text{if } \alpha = b_1 \ldots b_\ell,\ \ell > 1 \end{cases} \qquad (2.8)$$

Note that a substitution error can also be considered as an insertion transformation followed by a deletion transformation. The consistency of this multiple-error stochastic model defined in (2.8) can easily be proved from (2.7). Therefore, we have

$$\sum_{\alpha\in\Sigma^*} q(\alpha|a) = 1 \qquad (2.9)$$

The proof of (2.9) is given in Appendix A.

The probability of inserting α, $\alpha \in \Sigma^*$, at the end of a string is defined as

$$q'(\alpha) = \begin{cases} 1 - q'_I & \text{when } \alpha = \lambda \\ (1 - q'_I)\, q'_I(b_1)\, q'_I(b_2) \ldots q'_I(b_\ell) & \\ & \text{when } \alpha = b_1 \ldots b_\ell,\ \ell \geq 1 \end{cases} \qquad (2.10)$$

where $q'_1 = \sum_{a\in\Sigma} q'_1(a)$.

Furthermore,

$$\sum_{\alpha\in\Sigma^*} q'(\alpha) = (1-q'_1)\left[\sum_{i=0}^{\infty} (q'_1)^i\right] = 1 \qquad (2.11)$$

It is also assumed that for any string of symbols $a_1, a_2,\ldots,a_n \in \Sigma$, and strings $\alpha_1, \alpha_2,\ldots,\alpha_n, \alpha_{n+1} \in \Sigma^*$ we have

$$q(\alpha_1\alpha_2\ldots\alpha_{n+1}|a_1a_2\ldots a_n) = q(\alpha_1|a_1)\ldots q(\alpha_n|a_n)\, q'(\alpha_{n+1}) \qquad (2.12)$$

With the deformation probability defined on each terminal of a string, the probability of deforming string x to string y, $q(y|x)$, where $x = a_1a_2\ldots a_n$ is defined as

$$q(y|x) = \max_i\left[\prod_{j=1}^{n} q(\alpha_j^i|a_j)\right]q'(\alpha_{n+1}^i) \qquad (2.13)$$

where $\alpha_1^i\ldots\alpha_n^i\alpha_{n+1}^i$, $|\alpha_j^i| \geq 0$, is a partition of y into n+i substrings, and $1 \leq i \leq r$, r is the number of different ways of partitioning y into n+i substrings.

A graphical interpretation of (2.12) is particularly illustrative. Consider an example, in which $x = a_1a_2a_3$ and $y = b_1b_2b_3b_4$, whose lattice is shown in Figure 2.6. In this lattice, a horizontal branch indicates an insertion transformation, a vertical branch indicates a deletion transformation, and a diagonal branch indicates a substitution transformation or a non-error transformation.

In Figure 2.6, each traverse from point B to point E represents one way of deforming x into y. The heavy line indicates that b_1 is an insertion in front of a_1, b_2 is a substitution for a_1, a_2 is deleted,

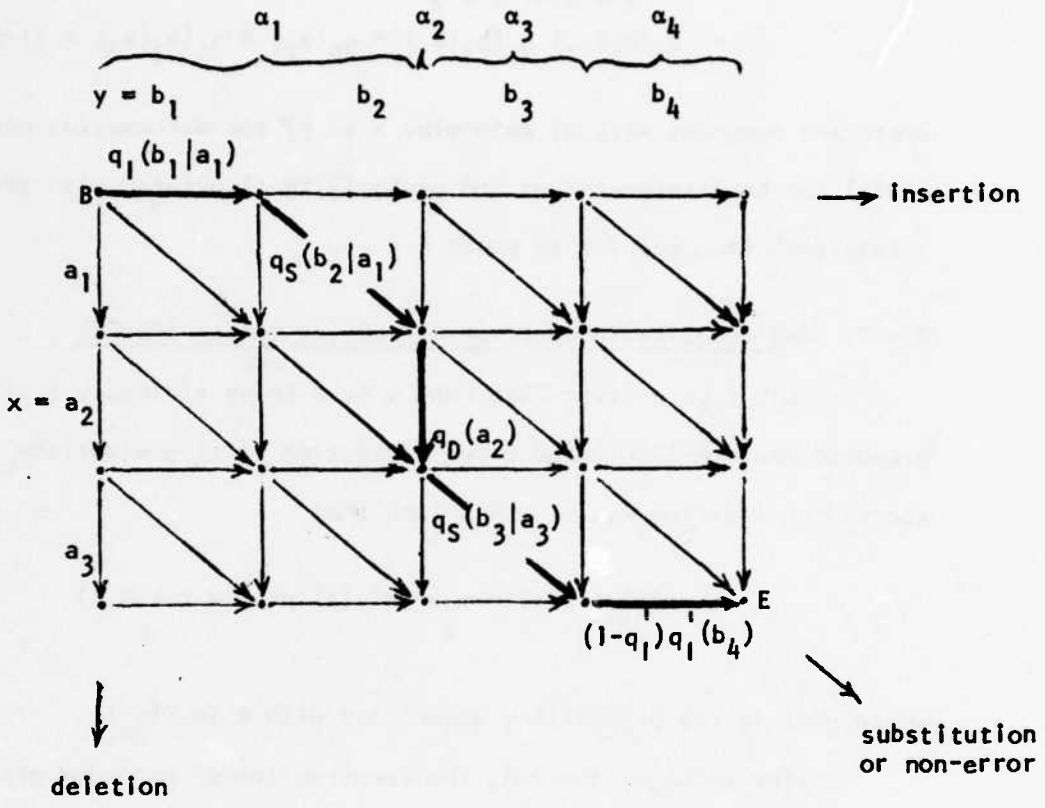


Figure 2.6 The stochastic deformation model described by a lattice

b_3 substitutes a_3, and b_4 is inserted at the end. This deformation is made possible by partitioning y into $\alpha_1\alpha_2\alpha_3\alpha_4$, where $\alpha_1 = b_1b_2$, $\alpha_2 = \lambda$, $\alpha_3 = b_3$, $\alpha_4 = b_4$. We have,

$$q(\alpha_1\alpha_2\alpha_3\alpha_4|a_1a_2a_3)$$
$$= q_1(b_1|a_1)\ q_S(b_1|a_1) * q_D(a_2) * q_S(b_3|a_3) * (1-q'_1)q'_1(b_4)$$

There are numerous ways of deforming x to y, the deformation probability $q(y|x)$ can be interpreted as the probability associated with the most likely path from point B to point E.

2.4.2. Maximum-Likelihood Error-Correcting Parser (MLECP)

Let $L(G_s)$ be a given SCFL, and y be a noisy string, $y \notin L(G_s)$. The proposed maximum-likelihood error-correcting parsing algorithm is to search for a string x, $x \in L(G_s)$ such that

$$q(y|x)\ p(x) = \max_z \{q(y|z)\ p(z)|z \in L(G_s)\} \qquad (2.14)$$

where $p(z)$ is the probability associated with z in $L(G_s)$.

Similar to Algorithm 2.1, the construction of expanded grammars based on a stochastic deformation model is given as follows:

Algorithm 2.5. Construction of stochastic expanded grammar.

input: A SCFG $G_s = (N,\Sigma,P_s,S)$

output: $G'_s = (N',\Sigma',P'_s,S')$ the stochastic expanded grammar.

Method:

Step 1. $N' = N \cup \{S'\} \cup \{E_a|a \in \Sigma\}$.

Step 2. $\Sigma' \supseteq \Sigma$

Step 3. If $A \xrightarrow{p} \alpha_0b_1\alpha_1b_2\alpha_2 \ldots b_mb_m$, $m \geq 0$, is a production in P_s such that α_i is in N^*, and b_i is in Σ, then add the

production $A \xrightarrow{p} \alpha_0 E_{b_1} \alpha_1 E_{b_2} \ldots E_{b_m} \alpha_m$ to P_s', where each E_{b_i} is a new non-terminal, $E_{b_i} \in N'$.

Step 4. Add to P_s' the productions

(a) $S' \xrightarrow{1 - q'_1} S$ where $q'_1 = \sum_{a \in \Sigma'} q'_1(a)$,

(b) $S' \xrightarrow{q'_1(a)} S'a$ for all $a \in \Sigma'$.

Step 5. For all $a \in \Sigma$, add to P_s' the productions

(a) $E_a \xrightarrow{q_S(a|a)} a$

(b) $E_a \xrightarrow{q_S(b|a)} b$ for all $b \in \Sigma'$, $b \neq a$,

(c) $E_a \xrightarrow{q_D(a)} \lambda$

(d) $E_a \xrightarrow{q_I(b|a)} b\,E_a$ for all $b \in \Sigma'$.

Suppose that y is an error-deformed string of x, $x = a_1 a_2 \ldots a_n$. By using productions added to P_s' by Step 3, we have $S \xrightarrow[G_s']{p_i} X$, where $X = E_{a_1} E_{a_2} \ldots E_{a_n}$, if and only if $S \xrightarrow[G_s]{p_i} x$, where p_i is the ith derivation of x in G_s. Applying Step 4(a) first and then repeatedly applying Step 4(b), we can further derive $S' \xrightarrow[G_s']{p_i'} X\,\alpha_{n+1}$ where $p_i' = p_i\, q'(\alpha_{n+1})$. The productions in Step 5 generate $E_{a_i} \xrightarrow[G_s']{q(\alpha_i|a_i)} \alpha_i$ for all $1 \leq i \leq n$, if $\alpha_1, \alpha_2, \ldots, \alpha_{n+1}$ is a partition of y. Step 5(a), (b), (c), and (d) correspond to non-error transformation, substitution transformation, deletion transformation, and insertion transformation which allows multiple insertions, respectively.

Thus, the stochastic language generated by G_s is

$$L(G_s') = \{(y, p(y)) \mid y \in \Sigma^*,\ p(y) = \sum_{x \in G_s} \sum_{i=1}^{r} q^i(y|x)\ p(x)\}$$

where r is the number of distinctive sequence of transformations to derive y from x, and $q^i(y|x)$ is the probability associated with the ith sequence, $1 \leq i \leq r$.

The consistency of $L(G_s')$ can be proved from equation (2.9), (2.11) and (2.12).

It is proposed to use a modified Earley's parser on G_s' to implement the searching of the most likely correction of a noisy input. The algorithm is essentially Earley's Algorithm with a provision added to keep accumulating the probabilities associated with each step of derivations.

Algorithm 2.6. Maximum-Likelihood Error-Correcting Algorithm

input: A stochastic expanded grammar $G_s' = (V', \Sigma', P_s', S)$ of G_s, and string $y = b_1 b_2 \ldots b_m$ in Σ'^*.

output: Parse lists of y.

Method:

Step 1. Set $j = 0$, and add $[E \rightarrow \cdot S', 0, 1]$ to I_j.

Step 2. (a) If $[A \rightarrow \alpha \cdot B\beta, i, p]$ is in I_j, and $B \xrightarrow{q} \gamma$ is a production in P_s' add item $[B \rightarrow \cdot \gamma, j, i]$ to I_j.

(b) If $[A \rightarrow \alpha \cdot, i, p]$ is in I_j and $[B \rightarrow \beta \cdot A\gamma, k, q]$ is in I_j, and if no item of the form $[B \rightarrow \beta A \cdot \gamma, k, r]$ can be found in I_j, add a new item $[B \rightarrow \beta A \cdot \gamma, k, tpq]$ to I_j, where t is the probability associated with $A \rightarrow \alpha$ in P_s'. If $[B \rightarrow \beta A \cdot \gamma, k, r]$ is already in I_j, then replace r by tpq if $tpq > r$.

Step 3. If $j = m$, go to Step 5. Otherwise, $j = j+1$.

Step 4. For each item in I_{j-1} of the form $[A \rightarrow \alpha \cdot b_j\beta, i, p]$ add item $[A \rightarrow \alpha b_j \cdot \beta, i, p]$ to I_j, go to Step 2.

Step 5. If item $[E \rightarrow S' \cdot, 0, p]$ is in I_m, exit.

The extraction of the most likely correction of y, x, that satisfies equation (2.14) are the same as that in MDECP.

From Step 5, the probability, p, in the item $[E \rightarrow S' \cdot, 0, p]$ of the last parse list, I_m, gives the value of $q(y|x)p(x)$ for some $x \in L(G_s)$ that satisfies equation (2.14) if the stochastic grammar, G_s is unambiguous. Since whenever a substring has more than one derivation, the algorithm chooses the one associated with the largest probability. Consequently, the derived number, p, is $q(y|x)p_j(x)$, where x is the most likely correction of y and $p_j(x)$ is the probability associated with the jth distinctive derivation of x with respect to G_s (refer to Definition 2.8). Therefore, only when x has one derivation can p be interpreted as $q(y|x)p(x)$.

2.4.3. Bayes Classification of Noisy Patterns

Assume that there are two classes of syntactic patterns, C_1 and C_2. Let x be a pattern. Suppose that the probability density function for x in C_i, $p(x|C_i)$ and the a priori probability of C_i, $P(C_i)$, where $i = 1,2$, are known. Using Bayes rule, the aposteriori probability that x is in class j is

$$P(C_j|x) = \frac{p(x|C_j)P(C_j)}{\sum_{i=1}^{2} p(x|C_i)P(C_i)}, \text{ for } j = 1,2 \qquad (2.15)$$

Then, the maximum-likelihood decision rule (or Bayes decision rule) is,

$$\text{decide } x\varepsilon \begin{matrix} C_1 \\ C_2 \end{matrix} \text{ if } P(C_1|x) \gtrless p(C_2|x) \tag{2.16}$$

Let $\tilde{C}_1$, $\tilde{C}_2$ be two set of training patterns for C_1 and C_2 respectively. Two stochastic grammars $G_1 = (N_1,\Sigma_1,P_1,S_1)$ and $G_2 = (N_2,\Sigma_2,P_2,S_2)$ are constructed to characterize $\tilde{C}_1$ and $\tilde{C}_2$ respectively, such that the probability of a sentence in $L(G_i)$ yields the probability of its corresponding syntactic pattern in C_i, for i = 1,2. Let the probability of x in $L(G_i)$ be denoted as $p(x|G_i)$, where x is a given sentence. $p(x|G_i)$ can be taken as $p(x|C_i)$ in equation (2.15). The maximum-likelihood decision rule is then applied for recognition, provided that x is in $L(G_1) \cup L(G_2)$, [64]. We shall rewrite equation (2.15) and (2.16) as follow:

$$\text{decide } x\varepsilon \begin{matrix} C_1 \\ C_2 \end{matrix} \text{ if } p(x|G_1)P(C_1) \gtrless p(x|G_2)P(C_2) \tag{2.17}$$

Note that $p(x|G_i) = 0$ if $x \notin L(G_i)$

The purpose of this section is to provide a recognition rule that minimize error rate for a given string, even if it is not in any of the languages under consideration. Let the deformation probabilities of terminals in G_i and G_2 be known. Let G_1' and G_2' be the stochastic expanded grammars for G_i and G_2 respectively according to the deformation probabilities. Given a string y, we shall interpret the term $q_i(y|x)p(x)$ that satisfies equation (2.14) as the probability that y is an error deformed string of $L(G_i)$, and denote it as $q(y|G_i')$ where $q_1(''|'')$ denotes the deformation probability of terminals in G_i. Then equation (2.14) can be rewriten as

$$q(y|G_i') = \max_{z} \{q_i(y|z)P(z) | z \in L(G_i)\} \qquad (2.18)$$

The use of expanded grammar, G_i', enlarges the probability space from $L(G_i)$ to $L(G_i')$ in which the deformation probabilities are employed to adjust the density function $p(x|G_i)$ for $x \in L(G_i)$. Consequently, the probability density function defined in equation (2.18) for sentences in $L(G_i')$ more closely yields the distribution of syntactic patterns in C_i. Let the recognition rule be as follows:

$$\text{decide } y \in \begin{matrix} C_1 \\ C_2 \end{matrix} \text{ if } q_1(y|G_1')P(C_1) \gtrless q_2(y|G_2')P(C_2) \qquad (2.19)$$

2.4.4. An Illustrative Example

A chromosome pattern classification problem is used as an example. Consider that there are four different types of chromosomes-submedian, median, accrocentric, and telocentric-denoted as C_S, C_M, C_A, and C_T, respectively [64,65]. The typical chromosome patterns of each of the four types are illustrated in Figure 2.8. Let the SCFG's for C_S, C_M, C_A, and C_T be G_S, G_M, G_A, and G_T, respectively. Segmentation errors due to noise and distortion in input patterns are considered as insertion or deletion errors, and, primitive extraction errors are considered as substitution errors. With a set of training samples, we can estimate deformation probabilities based on the segmentation and primitive extraction errors committed by the syntactic pattern recognition system. Then the MLECP's are constructed from the estimated deformation probabilities and SCFG's for each type of chromosomes. A set of samples, generated from the stochastic expanded grammars G_S', G_M', G_A', and G_T' (Algorithm 2.5), is then used as the test samples. They are classified by a Bayes

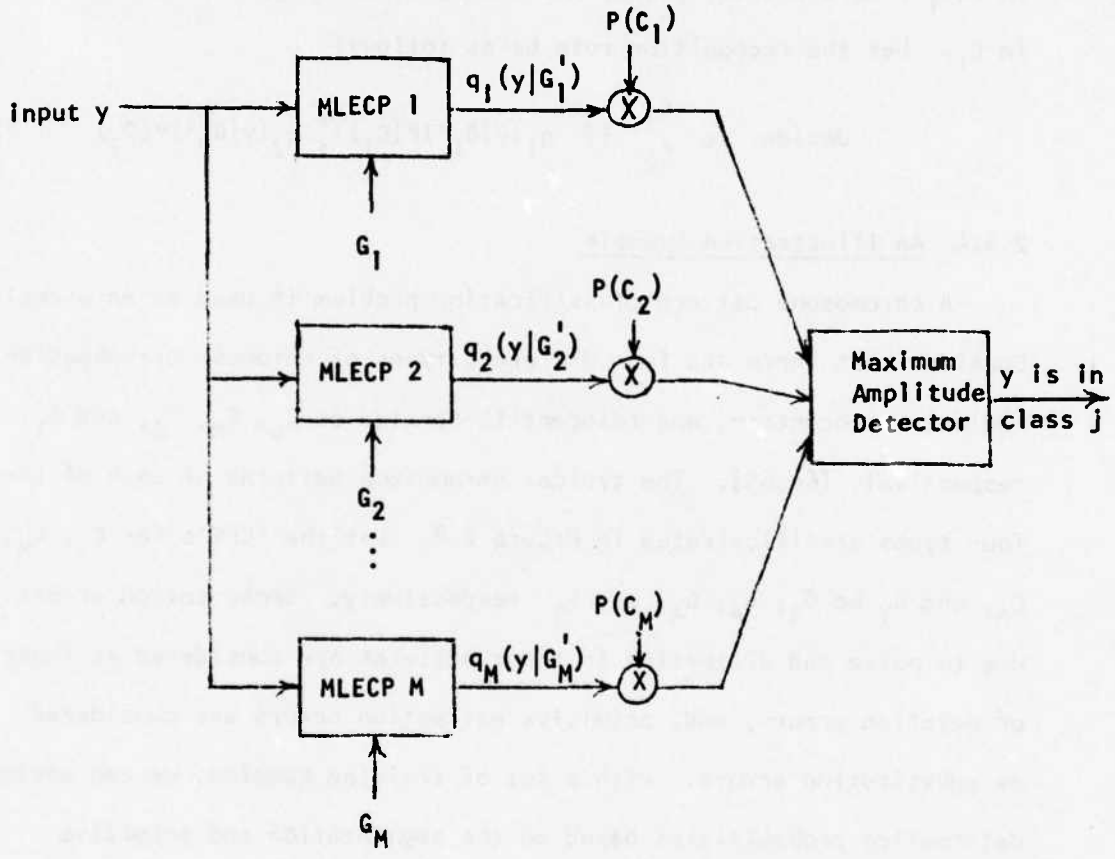
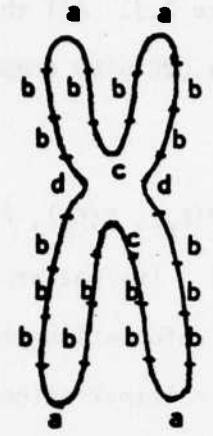


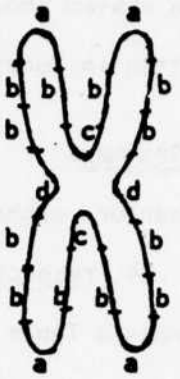
Figure 2.7 Bayes classification system for noisy strings



submedian

cbabbdbbbabb
cbbabbbdbbab

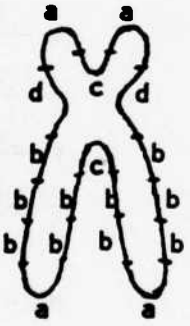
(a)



median

cbabbdbbabcbabbdbbab

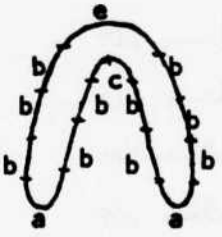
(b)



acrocentric

cadbbbabbcbbabbbda

(c)



telecentric

ebbbabbcbbabbb

(d)

Figure 2.8 The typical chromosome patterns and their string representations

classification system shown in Figure 2.7. All the programs in this paper are written in Fortran IV on a CDC 6500 computer.

(A) Pattern Grammars

Let the apriori probabilities $P(C_S)$, $P(C_M)$, $P(C_A)$ and $P(C_T)$ be .5, .26, .20, and .04, respectively [64]. The pattern grammars G_S, G_M, G_A, and G_T are given in Table 2.1. The deformation probabilities of each terminal are given in Table 2.2. For illustration only, we assume that the deformation probabilities of different classes are the same, the probabilities in Table 2.1 and Table 2.2 are rather arbitrarily assigned.

$G_S = (N_S, \Sigma_S, P_S, S)$

P_S:

$S \xrightarrow{.5} WU$

$S \xrightarrow{.5} UW$

$W \xrightarrow{1.} cM$

$U \xrightarrow{1.} cN$

$M \xrightarrow{1.} bRb$

$N \xrightarrow{1.} bLb$

$R \xrightarrow{.55} bRb$

$R \xrightarrow{.45} aQF$

$Q \xrightarrow{.6} bQb$

$Q \xrightarrow{.4} bdb$

$L \xrightarrow{.55} bLb$

$L \xrightarrow{.45} FQa$

$F \xrightarrow{.4} bFb$

$F \xrightarrow{.6} bab$

$N_S = \{S,W,U,M,N,R,L,Q,F\}$

$\Sigma_S = \{a,b,c,d\}$

$G_M = (N_M, \Sigma_M, P_M, S)$

P_M:

$S \xrightarrow{1.} WW$

$W \xrightarrow{1.} cM$

$M \xrightarrow{1.} bNb$

$N \xrightarrow{.55} bNb$

$N \xrightarrow{.45} aFa$

$F \xrightarrow{.6} bFb$

$F \xrightarrow{.4} bdb$

$N_M = \{S,W,M,N,F\}$

$\Sigma_M = \{a,b,c,d\}$

$G_A = (N_A, \Sigma_A, P_A, S)$

P_A:

$S \xrightarrow{1.} UW$

$U \xrightarrow{1.} ca$

$W \xrightarrow{1.} Qa$

$Q \xrightarrow{1.} dMd$

$M \xrightarrow{.6} bMb$

$M \xrightarrow{.4} bNb$

$N \xrightarrow{.1} aFa$

$F \xrightarrow{.55} bFb$

$F \xrightarrow{.45} bcb$

$N_A = \{S,U,W,Q,M,N,F\}$

$\Sigma_A = \{a,b,c,d\}$

$G_T = (N_T, \Sigma_T, P_T, S)$

P_T:

$S \xrightarrow{1.} eM$

$M \xrightarrow{.6} bMb$

$M \xrightarrow{.4} bNb$

$N \xrightarrow{1.} aFa$

$F \xrightarrow{.55} bFb$

$F \xrightarrow{.45} bcb$

$N_T = \{S,M,N,F\}$

$\Sigma_T = \{a,b,c,e\}$

Table 2.1 G_S, G_M, G_A, G_T for submedian, median, acrocentric, and telocentric, respectively.

$q_S(a\|a)=.97$	$q_S(b\|a)=0.$	$q_S(c\|a)=0.$	$q_S(d\|a)=.008$	$q_S(e\|a)=0.$
$q_S(b\|b)=.963$	$q_S(a\|b)=0.$	$q_S(c\|b)=0.$	$q_S(d\|b)=.012$	$q_S(e\|b)=.003$
$q_S(c\|c)=.965$	$q_S(a\|c)=0.$	$q_S(b\|c)=0.$	$q_S(d\|c)=.003$	$q_S(e\|c)=.01$
$q_S(d\|d)=.955$	$q_S(a\|d)=.01$	$q_S(b\|d)=.01$	$q_S(c\|d)=.003$	$q_S(e\|d)=0.$
$q_S(e\|e)=.965$	$q_S(a\|e)=0.$	$q_S(b\|e)=.003$	$q_S(c\|e)=.01$	$q_S(d\|e)=0.$
$q_D(a)=.01$	$q_D(b)=.01$	$q_D(c)=.01$	$q_D(d)=.01$	$q_D(e)=.01$

$q_I(j|i)=.0024$ for all $i, j, \quad i, j \in \{a,b,c,d,e\}$

$q_I'(i)=0.$ for all $i \in \{a,b,c,d,e\}$

Table 2.2 Probabilities of terminal error transformations.

(B) Parsing-Time Speed-Up Technique

As discussed in [13], the time complexity of minimum-distance error-correcting parsing is $O(N^3)$. In a stochastic model, some of the probabilities associated with items in parse lists are so small that their corresponding derivations hardly ever occur. We may eliminate these unnecessary derivations by comparing the probability of an item with a lower bound before it is added to the parse list.

Let $[A \rightarrow \alpha \cdot \beta, i, p]$ be an item in i_j. We set ξ as a lower bound. Then $[A \rightarrow \alpha \cdot \beta, i, p]$ is allowed to be added to i_j only if $p \geq \xi$. A good selection of ξ should depend on the tolerance of error density of a string. We use $C(A)^{j-i}$ as the lower bound in this paper, where A, C are constants. The value of C should be less than the smallest deformation probability to permit its associated error to exist. The value of A depending on stochastic grammar rules should be small enough to guarantee successful parsing of noisy strings.

For the string y = cbabdbabcbabdbab, Table 2.3 gives the approximate maximum number of errors allowed in a substring of given length when parsed by G_M' with C = .001, A = .5.

To illustrate the amount of parsing time saved by applying this technique, we parse a string both by G_M' with a preset lower bound and without a lower bound. Figure 2.9 shows this comparison in terms of the number of items in each parse list when string cbabbdbbabcbabbdbbab is parsed by G_M' with the lower bound A = .5, C = .001. Total cpu time is 5.4 seconds with the preset lower bound and 34.3 sec. without.

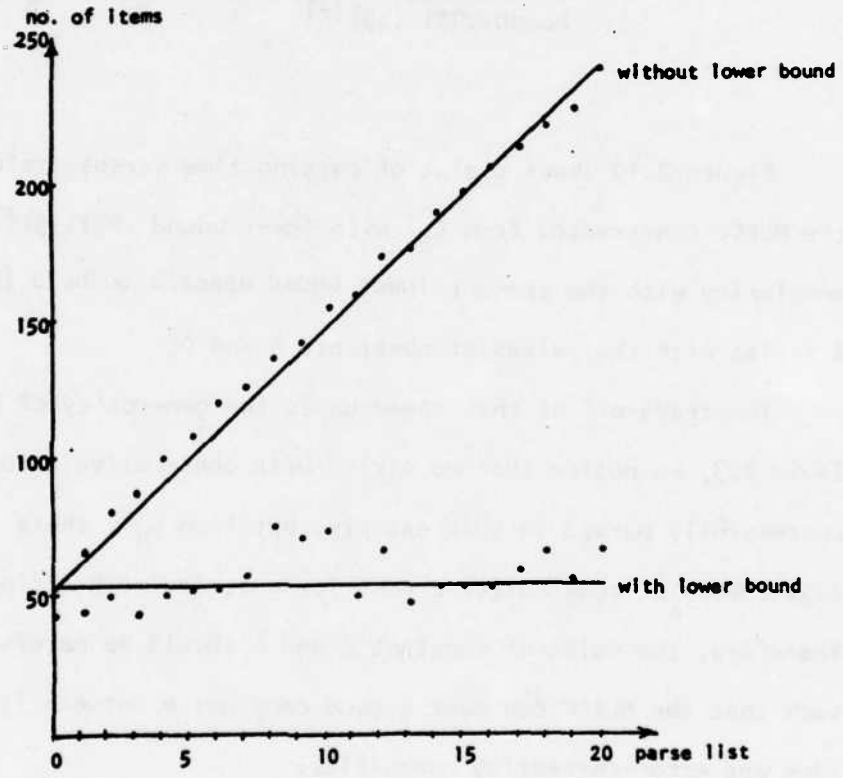


Figure 2.9 Comparison of number of items in parse list of string cbabbdbbabcbabbdbbab parsed by G'_M with and without preset lower bound

length of substring	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
maximum no. of error allowed	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3

Table 2.3 Maximum number of errors allowed in substrings of cbabdbabcbabdbab when parsed by G_M' with lower bound .001 $(.5)^{j-1}$.

Figure 2.10 shows a plot of parsing time versus string length using the MLECP constructed from G_S' with lower bound $.001(.5)^{j-i}$. The time complexity with the present lower bound appears to be $0(n^{\ell})$, $2 \leq \ell \leq 3$ and ℓ varies with the values of constants A and C.

The trade-off of this speed-up is the generosity of MLECP. From Table 2.3, we notice that no string with consecutive errors can be successfully parsed in this example, but from G_M', there is nearly 2.5% chance of consecutive errors for a string with string length 16. Therefore, the value of constant A and C should be carefully chosen such that the MLECP can meet a good compromise between its parsing time and error-correcting capability.

(C) Classification Result

Thirty-two test samples are generated from G_S', G_M', G_A', and G_T' with average string length 27, of which 20 are erroneous. The 32 samples are then tested by the Bayes classifier with lower bound set at $.001(.5)^{j-1}$. The result is that, among 32 samples, 29 are correctly classified. The rest of the three samples are too noisy to be correctly classified due

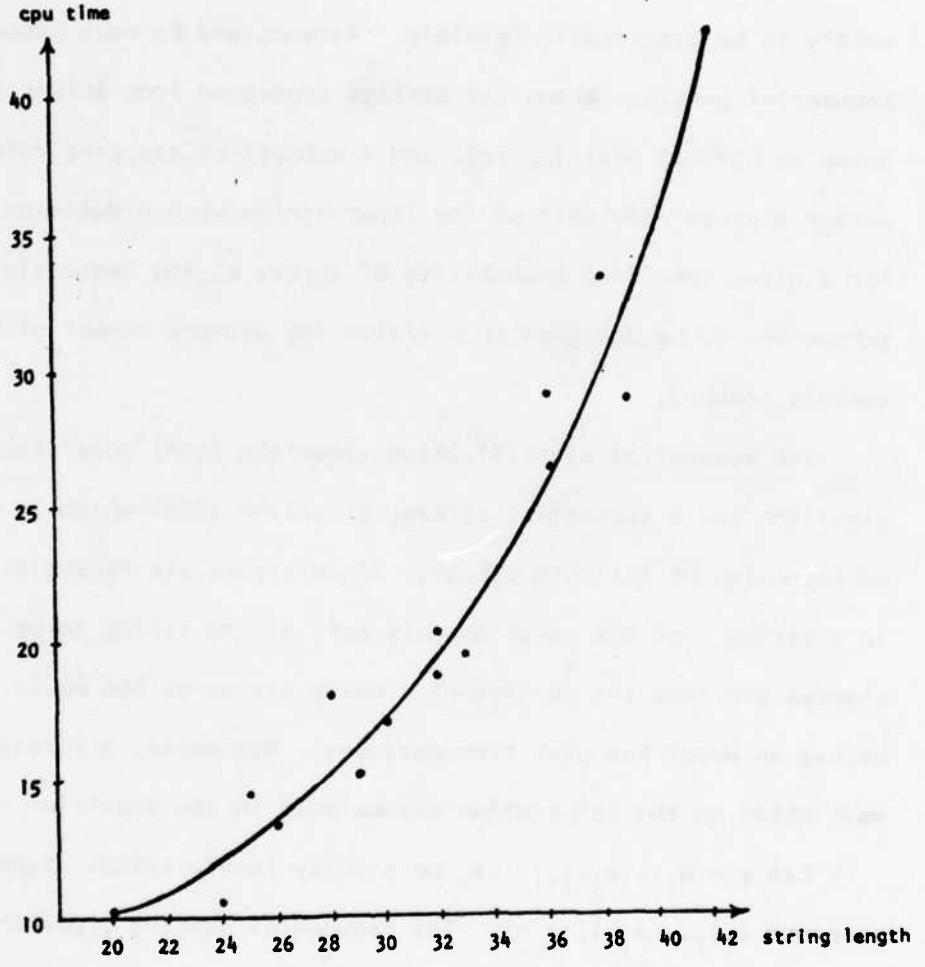


Figure 2.10 cpu time vs. string length

to the use of lower bound of MLECP's. On the average, it takes 32.6 sec. to classify a string with an overall accuracy of 90.6%.

2.4.5. Sequential Classification of Noisy Patterns

In the previous section, we have designed MLECP's for the classification of noisy and distorted patterns. As it was shown, even if the lower bound of ECP are skillfully selected, the use of MLECP's is still too costly to be practically feasible. Persoon and Fu have proposed a sequential parsing scheme for strings generated from SCFG's [10]. By using an optimal decision rule and a suboptimal stopping rule, the parser scanned only part of the input string when a decision is made. For a given specified probability of error, $\hat{e}$, the sequential parsing scheme should be designed to minimize the average number of terminal symbols scanned.

The sequential classification algorithm (SCA) consists of a decision algorithm and a sequential parsing algorithm (SPA) which is essentially an extension of Earley's parser. Since errors are randomly distributed in a string, and SCA requires only part of the string to be parsed, chances are that the parsing of a noisy string by SCA would be successful before an error has ever been detected. Otherwise, a forced decision is made based on the information accumulated in the provision of SPA.

Let $y = a_1 \ldots a_j a_{j+1} \ldots a_n$ be a noisy input string. Suppose that we observed $a_1 a_2 \ldots a_j$ $(j \leq n)$. The sequential parsing algorithm (SPA) computes $p(a_1 a_2 \ldots a_j | C_i)$ which is the probability that $a_1 a_2 \ldots a_j$ is a string in $L(G_i)$, and $p(a_1 \ldots a_j \alpha | C_i)$ which is the total probability of strings in $L(G_i)$ with $a_1 a_2 \ldots a_j$ as their prefix. By using these two quantities, a stopping rule tells when one has to stop observing more terminals, and

a decision rule assigns a class to y once the stopping rule indicates to stop.

The SCA designed for processing erroneous strings is similar to the algorithms presented in [10] for non-error-correcting parsing. A forced decision rule is added to the algorithms such that a decision can be reached where parsing is terminated by illegitimate terminals. The restriction on using λ productions in [10] is removed. The algorithms are given in Appendix B.

Two hundred test samples are generated from G_S', G_M', G_A', and G_T'. Among them, 112 are erroneous. The average string length is 26. The classification results of using SCA with a preset error bound $\hat{e}$ are illustrated in Table 2.4 which shows classification accuracy and parsing time with respect to various $\hat{e}$. In the second experiment, the 200 test samples are passed through a non-sequential Earley's parser constructed from G_S, G_M, G_A and G_T. The result is given at the last row of Table 2.4. For the purpose of comparison, the third experiment uses 200 correct strings generated from SCFG's G_S, G_M, G_A, and G_T as test samples. They are classified by using SCA. The results are shown in Table 2.5. Curves showing accuracy vs. parsing time of Experiment 1, 2, and 3 are given in Figure 2.11.

From Table 2.5, if the parsed strings are error-free, the accuracy of sequential classification increases montonically as the number of symbols scanned increases. One hundred percent accuracy may be reached when the error bound is sufficiently small. Whereas, the curve of accuracy vs. parsing time (or the number of parsed symbols per string) is convex when the parsed strings are noisy. In this case, the

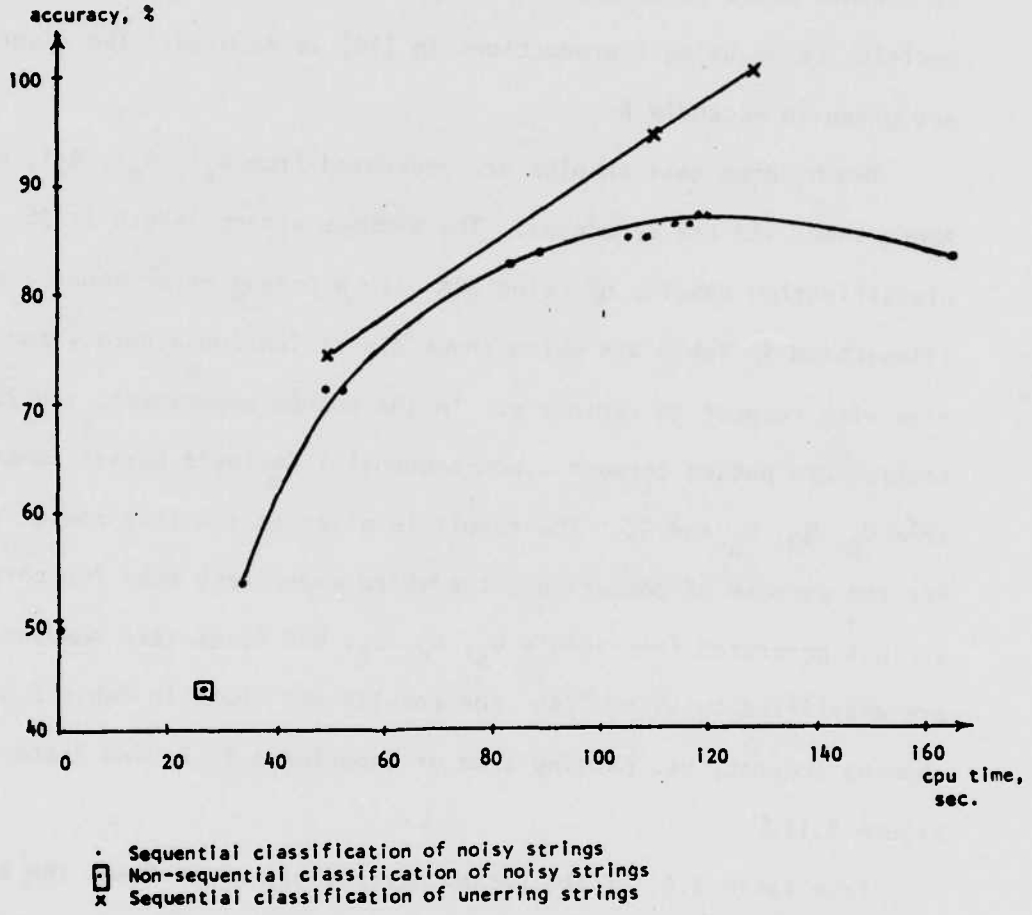


Figure 2.11 Accuracy vs. parsing time of sequential classifier and nonsequential classifier

Sequential Classifier

error bound $\hat{e}$	no. of misclassification	accuracy	total cpu time (sec)	ave. no. of parsed symbol per string
.0	33	83.5%	162.5	16
.03	26	87%	119.0	8
.05	26	87%	118.5	8
.1	27	86.5%	116.5	7
.15	27	86.5%	113.4	7
.18	30	85%	108.2	6
.21	30	85%	104.8	6
.25	32	84%	88.8	5
.3	35	82.5%	82.7	4
.35	57	71.5%	52.4	1
.4	57	71.5%	49.1	1
.5	93	53.5%	37.1	1
.6	101	49.5%	.036	0
Non-sequential Classifier				
	112	44%	27.0	

Table 2.4 Classification results from 200 strings (112 of them are erroneous)

Sequential Classifier

error bound $\hat{e}$	no. of misclassification	accuracy	total cpu time (sec)	ave. no. of parsed symbol per string
.1	0	100%	128.2	8
.2	11	94.5%	110.2	7
.35	51	74.5%	50.0	1

Table 2.5 Classification results from 200 correct strings.

accuracy of the sequential classifier cannot reach beyond a maximum point. In our example, the maximum is 87% accuracy when error bound is .05 with average parsing time .6 sec per string. The interpretation is intuitive, the more symbols are scanned, the higher the accuracy SCA can reach, and the better the chance an error is detected. As soon as an error is announced by the SPA of all the pattern grammars, a forced decision, whose low accuracy plays the counterpart, must be made.

To increase accuracy beyond the maximum accuracy of sequential classifier, we can use a sequential error-correcting parser as the classifier which is a SCA with its SPA constructed from error-induced grammars. The results of classifying 200 noisy strings are given in Table 2.6. When error bound is .15, the sequential error-correcting classifier reaches 94% accuracy with average 19.1 sec per string. Comparing with the result of 90.2% accuracy and average 32.6 sec per string using a non-sequential error-correcting classifier in Section 2.4.3, the sequential version achieved a slightly higher accuracy with less average parsing time.

The effectiveness of using SCA for processing noisy strings is largely pattern grammars dependent. It is noted that, most misclassifications of SCA occur between median and submedian because no distinctive difference appears at the first few symbols of their string representations. Whereas, it takes only two or three symbols to discriminate acrocentric from the other three types with very high accuracy. We may conclude that to increase both accuracy and efficiency using SCA, pattern grammars should be carefully constructed such that informative symbols will appear at the first few positions of derived strings.

error bound $\hat{e}$	no. of misclassification	accuracy	total cpu time (sec)	ave. no. of parsed symbol per string
.15	12	94%	3816	8
.25	27	86.5%	2650	5

Table 2.6 Sequential error-correcting classification.

CHAPTER 3

ERROR-CORRECTING TREE AUTOMATA

3.1 Introduction

In applying syntactic methods to pattern recognition, one-dimensional (string) grammars are sometimes inefficient in describing two- or three-dimensional patterns. For the purpose of effectively describing high-dimensional patterns, high-dimensional grammars such as web grammars, graph grammars, and tree grammars have been proposed [50-52]. Properties of generalized finite automata, called tree automata, which accept finite trees of symbols as its input, have been studied by several authors [66-70]. Brainerd [66] proves that the class of systems which generates exactly the sets of trees accepted by the automata is a regular system. Fu and Bhargava [50] introduced the application of the tree systems into pattern recognition. In practical application, tree grammars and tree automata have been used in the classification of fingerprint patterns [38], the analysis of bubble chamber events [74], and the interpretation of LANDSAT data [39], etc.

The descriptive power of tree languages and the efficient analytical capability of tree automata made the tree system approach to pattern recognition very attractive. This chapter is concerned with the error-correcting version of tree automata. Unlike the string case, where the only relation between symbols is left-right concatenation, a tree structure would be deformed under deletion or insertion errors. The structure-

preserved error-correcting tree automaton (SPECTA) proposed in Section 3.3 takes only substitution errors into consideration. By introducing a blank element, a deletion error can be treated as substitution of a non-blank element by a blank element, and an insertion error becomes a non-blank element in substitution for a blank element. An example of using SPECTA in LANDSAT data interpretation is presented.

In Section 3.4, syntax errors on trees are defined in terms of five error transformations; namely, substitution, stretch, branch, split, and deletion. The distance between two trees is the least cost sequence of error transformations needed to transform one to the other. Based on this tree metric, a generalized error-correcting tree automaton (GECTA), is formulated, where transformations made on each terminal symbol are added to the system in the form of transition rules. An example of hand-printed character recognition is given to demonstrate the operation of GECTA.

3.2 Definitions

In this section, some basic definitions on trees, tree grammars, and tree automata given by Brainerd [66] are briefly reviewed.

Definition 3.1. Let N^+ be the set of positive integers. Let U be the free monoid generated by N^+. Let $\cdot$ be the operation and 0 the identity of U. The depth of $a \in U$ is denoted d(a) and defined as follows: $d(0)=0$, $d(a \cdot i)=d(a)+i$, $i \in N^+$. $a \leq b$ iff there exists $x \in U$ such that $a \cdot x = b$. a and b are incomparable iff $a \not\leq b$ and $b \not\leq a$.

Definition 3.2. D is a tree domain iff D is a finite subset of U satisfying (1) $b \in D$ and $a < b$ implies $a \in D$, and (2) $a \cdot j \in D$ and $i < j$ in N^+ implies $a \cdot i \in D$.

Definition 3.3. A rank is a pair $<\Sigma,r>$ where Σ is a finite set of symbols and $r:\Sigma\to N$. Let $\Sigma_n = r^{-1}(n)$.

Definition 3.4. A tree over Σ (i.e., over $<\Sigma,r>$) is a function $\alpha:D\to\Sigma$, such that D is a tree domain and $r[\alpha(a)] = \max\{i \mid a\cdot i\in D\}$. i.e., the rank of a label at a must be equal to the number of branches in the tree domain at a. The domain of a tree is denoted $D(\alpha)$ or D_α. Let T_Σ be the set of all trees over Σ. The depth of α is defined as, $d(\alpha) = \max\{d(a) \mid a\in D_\alpha\}$.

Definition 3.5. Let $a, b, b'\in U$ such that $b = a\cdot b'$, then $b/a = b'$. b/a is not defined unless $a \leq b$.

Definition 3.6. Let $\alpha\in T_\Sigma$, and $a\in D_\alpha$, $\alpha/a = \{(b,x) \mid (a\cdot b,x)\in\alpha\}$, α/a is the subtree of α at a and α/a occurs at a in α.

Definition 3.7. Let $\alpha\in T_\Sigma$, $a\in U$, then $a\cdot\alpha = \{(b,x) \mid (b/a,x)\in\alpha\}$.

Definition 3.8. Let $a\in D_\alpha$, $\alpha, \beta\in T_\Sigma$, then $\alpha(a\leftarrow\beta) = \{(b,x)\in\alpha \mid b\not\geq a\} \cup a\cdot\beta$. This is the result of replacing the subtree α/a at a by the tree β.

Using postfix notation, the tree $\alpha = \bigcup_{i=1}^{n} i\cdot\alpha_i \cup \{(0,x)\}$ is represented by $\alpha_1\alpha_2\ldots\alpha_n x$.

Definition 3.9. t is a term over $<\Sigma,r>$ iff $t = x\in\Sigma_0$ or $t = t_1t_2\ldots t_n x$ where $x\in\Sigma_n$, and t_i, $1 \leq i \leq n$, is a term, $\bar{T}_\Sigma$ is the set of terms over Σ. There is obviously a one-to-one correspondence between the terms over Σ and the trees over Σ. If the corresponding tree of t is a subtree of α, we say t is a term in α.

Definition 3.10. A regular tree grammar over $<\Sigma,r>$ is a regular system $G_t = (V,r',P,S)$ satisfying the following conditions:

(1) $<V,r'>$ is a finite-ranked alphabet with $\Sigma\subseteq V$, and $r'|\Sigma = r$. The elements in V and $V-\Sigma$ are called terminal and nonterminal symbols, respectively.

(2) P is a finite set of productions of the form $\phi \rightarrow \psi$, where ϕ and ψ are trees over $<V,r'>$.

(3) $S \subseteq T_V$ is a finite set of start symbols.

Definition 3.11. Derivation $\alpha \xrightarrow{a} \beta$ is in G_t iff $\phi \rightarrow \psi$ in P such that $\alpha/a = \phi$ and $\beta = \alpha(a \leftarrow \psi)$. $\alpha \overset{*}{\Rightarrow} \beta$ is a derivation iff there exist $\alpha_0\alpha_1 \ldots \alpha_m$, $m \geq 0$ such that $\alpha = \alpha_0 \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_m = \beta$ in G_t.

Definition 3.12. The language generated by $G_t = (V,r',P,S)$ over $<\Sigma,r>$ is defined as, $L(G_t) = \{\alpha \varepsilon T_\Sigma \mid$ there exist $x \varepsilon S$ such that $x \overset{*}{\underset{G_t}{\Rightarrow}} \alpha\}$.

Definition 3.13. A tree grammar $G_t = (V,r',P,S)$ over $<\Sigma,r>$ is expansive iff each production in P is of the form

$$X_0 \rightarrow \begin{array}{ccccc} & & x & & \\ & \diagup & & \diagdown & \\ X_1 & \cdots & & & X_{r(x)} \end{array} \quad \text{or } X_0 \rightarrow x \text{ where } x \varepsilon \Sigma.$$

and X_0, $X_1 \ldots X_{r(x)}$ are nonterminal symbols.

Following the definition given by Brainerd [66], a tree automaton is a finite automaton with many-to-one state transition functions.

Definition 3.14. Let $<\Sigma,r>$ be a rank and $\Sigma = \{\sigma_1,\sigma_2 \ldots \sigma_k\}$. A finite Σ-automaton or tree automaton over Σ is a system $M_t = (Q,f_1 \ldots f_k,R)$ where

(1) Q is a finite set of states,

(2) for each i, $1 \leq i \leq k$, f_i is a relation on $Q^{r(\sigma_i)} \times Q$,

(3) $R \subseteq Q$ is a set of final states.

if each f_i, $1 \leq i \leq k$, is a function f_i: $Q^{r(\sigma_i)} \rightarrow Q$ then M_t is deterministic; otherwise M_t is nondeterministic.

The construction procedure of tree automaton for a regular tree grammar can be summarized as follows [50].

Algorithm 3.1. Construction of tree automaton.

Step 1. To obtain an expansive tree grammar (V',r,P',S) for the given tree grammar (V,r,P,S) over alphabet Σ.

Step 2. The equivalent nondeterministic tree automaton is $M_t = (V'-\Sigma, f_1 \ldots f_k, \{S\})$ where $f_i(X_1 \ldots X_n) \rightarrow X_0$ if $X_0 \rightarrow X_1 \ldots X_n x_i$ is in P'.

The acceptance of a tree by tree automaton is a backward procedure. It reads paralliel branches simultaneously then transfers to the states of their immediate predecessors.

3.3 Structure-Preserved Error-Correcting Tree Automaton (SPECTA)

Let D be a tree domain, $D \subset U$, Σ be a set of terminal symbols, we define $T_\Sigma^D = \{\alpha \mid \alpha \in T_\Sigma, D_\alpha = D\}$ be the set of trees in the tree domain D. In this section, substitution error is described in terms of the transformation S: $T_\Sigma^D \rightarrow T_\Sigma^D$. For $a \in D$, $x \in \Sigma$, and $\alpha, \alpha' \in T_\Sigma^D$, we write $\alpha \vdash^{S_{a/x}} \alpha'$ if α' is the result of replacing the label on node a of tree α by terminal symbol x. Furthermore, S^k denotes the composition of S with itself k times.

The distance on trees in T_Σ^D, $d(\alpha,\alpha')$, is defined as the smallest integer k for which $\alpha \vdash^{S^k} \alpha'$ if α and α' are two trees in T_Σ^D for some $D \subset U$. The function of d is symmetric and satisfies triangle inequality. Let L be a tree language, and tree $\alpha' \notin L$. The essence of SPECTA is to search for a tree α, $\alpha \in L$ such that

$$d(\alpha,\alpha') = \min_{\beta} \{d(\beta,\alpha') \mid \beta \in L,\ D_\beta = D_{\alpha'}\} \qquad (3.1)$$

and reconstruct α' as α. $d(\alpha,\alpha')$ in the above equation is also defined as the distance of α' from L, denoting $d(L,\alpha')$. α is called the minimum distance correction of α' in L.

3.3.1 Minimum-Distance SPECTA

By adding terminal error production rules corresponding to substitution error transformations, the covering grammar $G_{t'} = (V',r',P',S)$ of a given tree grammar $G_t = (V,r,P,S)$ is constructed as follows:

Step 1. $V' = (V-\Sigma) \cup \Sigma'$, where $\Sigma' \supseteq \Sigma$ is a new set of terminal symbols.

Step 2. For each $y \in \Sigma'$ add to P'

$$X_0 \rightarrow \begin{array}{c} y \\ / \quad \backslash \\ X_1 \cdots X_{r(x)} \end{array}, \text{ if } X_0 \rightarrow \begin{array}{c} x \\ / \quad \backslash \\ X_1 \cdots X_{r(x)} \end{array} \in P$$

or $X_0 \rightarrow y$ if $X_0 \rightarrow x$ is in P.

The language generated from G_t' consists of the language $L(G_t)$ and its corresponding erroneous trees. Hence, $L(G_t')$ can be written as

$$L(G_t') = \{\alpha' \mid \alpha' \in T_{\Sigma'}, \text{ and } \exists\ \alpha \in L(G_t) \text{ such that } D_{\alpha'} = D_\alpha\}$$

For a given tree grammar, G_t, the SPECTA is formulated to accept trees in $L(G_t')$ and to generate a parse that consists of the minimum number of error productions. Assume that α' is an input tree, the operation of a SPECTA is a backward procedure of construction a tree-like transition table from the frontiers to the root of α'. For each node $a \in D_{\alpha'}$, there

is a corresponding set of triplets, denoting t_a in the transition table. Each triplet (X,ℓ,k) is added to t_a if X is a candidate state of node a, ℓ is the minimum number of errors in subtree α'/a when node a is represented by state X, and k specifies the production rule used. The algorithm is given as follows.

Algorithm 3.2. Minimum-distance SPECTA

Input: $G_t = (V,r,P,S)$ and tree α'.

Output: Transition table of α' and $d(L(G_t),\alpha')$.

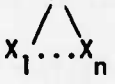
Method:

Step 1. If $r[\alpha'(a)] = 0$, $\alpha'(a) = x$, then add to t_a

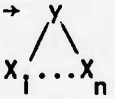
(a) $(X_0,0,k)$ if $X_0 \to x$ is the kth rule in P.

(b) $(X_0,1,k)$ if $X_0 \to y$ is the kth rule in P and $y \neq x$.

Step 2. If $r[\alpha'(a)] = n > 0$, $\alpha'(a) = x$, then add to t_a

(a) (X_0,ℓ,k), if $X_0 \to$  is the kth rule in P and

$(X_1,\ell_1,k_1) \in t_{a\cdot 1},\ldots,(X_n,\ell_n,k_n) \in t_{a\cdot n}$ then $\ell = \ell_1+\ldots+\ell_n$

(b) (X_0,ℓ,k) , if $X_0 \to$  is the kth rule in P, $y \neq x$, and

$(X_1,\ell_1,k_1) \in t_{a\cdot 1},\ldots,(X_n,\ell_n,k_n) \in t_{a\cdot n}$ then $\ell = \ell_1+\ldots+\ell_n+1$.

Step 3. Whenever more than one item in t_a has the same state, delete the item with larger number of errors.

Step 4. If $(S,\ell,k) \in t_0$, then $d(L(G_t),\alpha') = \ell$. If no item is in t_0 of the form (S,ℓ,k), then no tree in $L(G_t)$ is in tree domain $D_{\alpha'}$, the input tree is rejected.

The minimum-distance correction of α' can easily be traced out from the transition table.

The tree grammar in the following example is a part of the highway grammar used in Section 3.3.3. In the meantime, we use it as an example here to illustrate the operation of SPECTA and to demonstrate the highway patterns recognition procedure that will be discussed in Section 3.3.3.

<u>Example 3.1</u>. Consider a set of verticle line patterns as given in Figure 3.1. Assume that elements in the 4 x 4 array are connected as a tree shown in Figure 3.2. Thus, each pattern has its corresponding tree representations. For example, pattern (b) in Figure 3.1 can be represented by the tree shown in Figure 3.3, where nodes labeled by symbol "b" represent blank or nonhighway elements "□ ", and nodes labeled by "h" represent highway elements " ▧ ". The tree grammar that generates these tree representations can be written as:

$G_H = (V,r,P,S)$ over $<\Sigma,r>$ where

$V = \{S,A_0,A_1,A_2,A_3,X_0,I_1,I_2,I_3,\$,b,h\}$

$\Sigma = \{\underset{\$}{\cdot}, \underset{b}{\square}, \underset{h}{▧}\}$

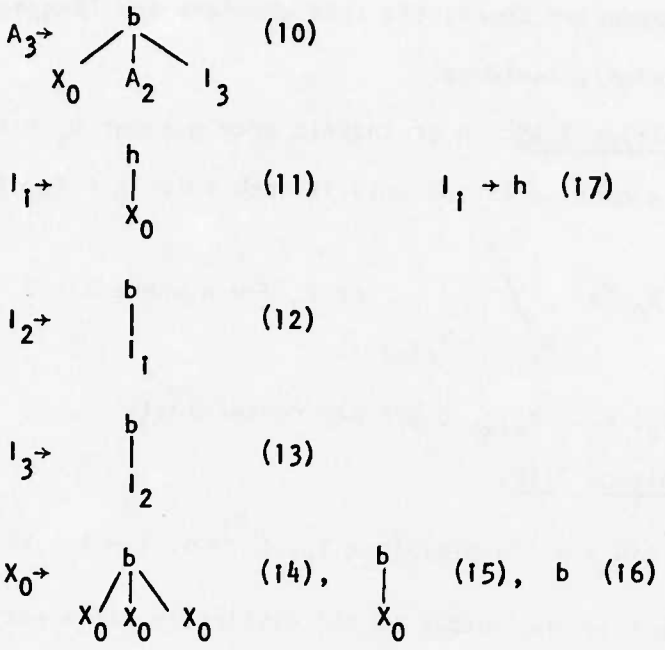
$r(\$) = 1,\ r(b) = \{0,1,3\},\ r(h) = \{0,1,3\}$

$$P:\ S \rightarrow \begin{array}{c}\$\\ |\\ A_0\end{array}(1),\quad \begin{array}{c}\$\\ |\\ A_1\end{array}(2),\quad \begin{array}{c}\$\\ |\\ A_3\end{array}(3),\quad \begin{array}{c}\$\\ |\\ A_4\end{array}(4)$$

$$A_0 \rightarrow \begin{array}{ccc} & h & \\ / & | & \backslash \\ A_0 & X_0 & X_0\end{array}(5),\quad \begin{array}{c}h\\ |\\ A_0\end{array}(6),\quad h\ (7)$$

$$A_1 \rightarrow \begin{array}{ccc} & b & \\ / & | & \backslash \\ X_0 & A_0 & I_1\end{array}(8)$$

$$A_2 \rightarrow \begin{array}{ccc} & b & \\ / & | & \backslash \\ X_0 & A_1 & I_2\end{array}(9)$$



Given a noisy pattern shown in Figure 3.4(a), let α' be its tree representation. α' is given in Figure 3.4(b). The transition table of α' resulted from using the minimum-distance SPECTA with respect to grammar G_H is shown in Figure 3.5. Since (S,3,2) is in t_0, α' is accepted by the SPECTA and the number of errors in α' is 3. Let the minimum-distance correction of α' be called α. The generation of α from the transition table is illustrated in Figure 3.6.

3.3.2 Maximum-likelihood SPECTA

When the probability distribution of patterns and the deformation probabilities on each terminal are availabe, error-correcting parsing based on maximum-likelihood criterion may provide a better recognition performance. Definitions of stochastic grammar, terminal deformation probabilities, and maximum-likelihood criterion have been introduced in Chapter 2. A similar approach to MLECP is used in formulating SPECTA based on maximum-likelihood criterion.

The expansive stochastic tree grammars and languages defined in [72] are briefly reviewed.

Definition 3.15. A stochastic tree grammar $G_s = (V,r,P,S)$ over Σ is expansive if and only if each rule in P is of the form

$$X_0 \xrightarrow{p} \overset{x}{\diagup\ \diagdown}_{X_1 \ \cdots \ X_{r(x)}} \quad \text{or} \quad X_0 \xrightarrow{p} x \text{ where } x \in \Sigma$$

and $X_0, X_1 \ldots X_{r(x)} \in V-\Sigma$ are nonterminals.

Definition 3.16.

$$L(G_s) = \{(\alpha, p(\alpha)) \mid \alpha \in T_\Sigma,\ S \overset{p_i}{\Longrightarrow} \alpha,\ i = 1 \ldots k,\ p(\alpha) = \sum_{i=1}^{k} p_i\}$$

where k is the number of all distinctly different derivation of α from S, and p_i is the probability associated with the ith distinct derivation of α from S.

Assume that the occurrence of substitution error on a terminal is independent from its neighboring terminals. Fung and Fu [31] define a substitution error made on strings to be a stochastic mapping $\sigma: \Sigma \to \Sigma$ such that $\sigma(a) = b$, if a and $b \in \Sigma$, with probability $q(b|a)$ and furthermore,

$$\sum_{b \in \Sigma} q(b|a) = 1 \tag{3.2}$$

The same definition can be applied to model a substitution error made on trees. Furthermore, assume that $t = t_1 \ldots t_n x$ is a term over $\langle \Sigma, r \rangle$,

$$\sigma(t_1 \ldots t_n x) = \sigma(t_1) \ \ldots \ \sigma(t_n)\sigma(x) \tag{3.3}$$

If two trees $\alpha = t_1 \ldots t_n\ x$, $\alpha' = t_1' \ldots t_n'\ x'$ are both in T_Σ^D, the probability of α' being the noisy deformed tree of α is

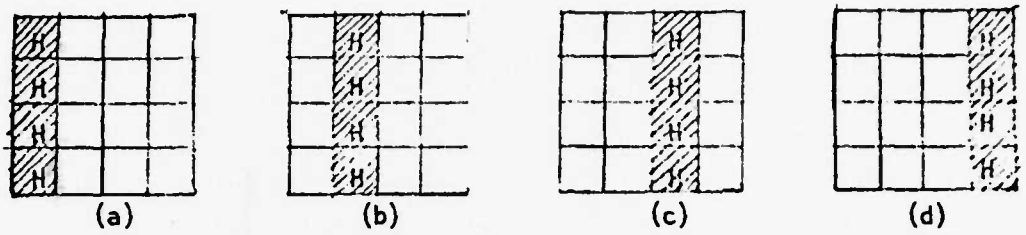


Figure 3.1 Vertical Line Patterns

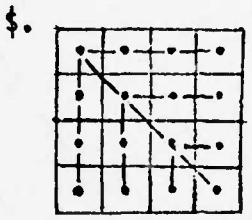


Figure 3.2 A Tree Structure

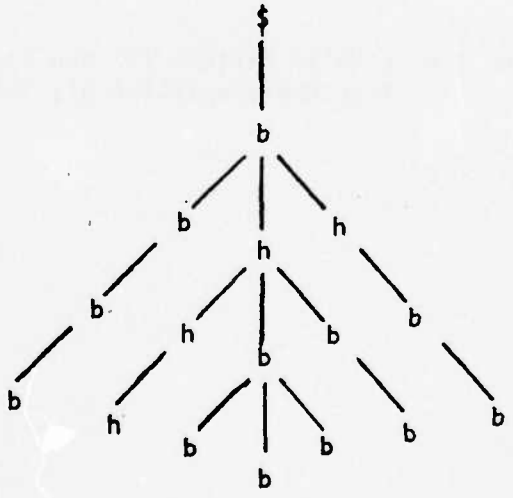


Figure 3.3 Tree Representation of Pattern (b)

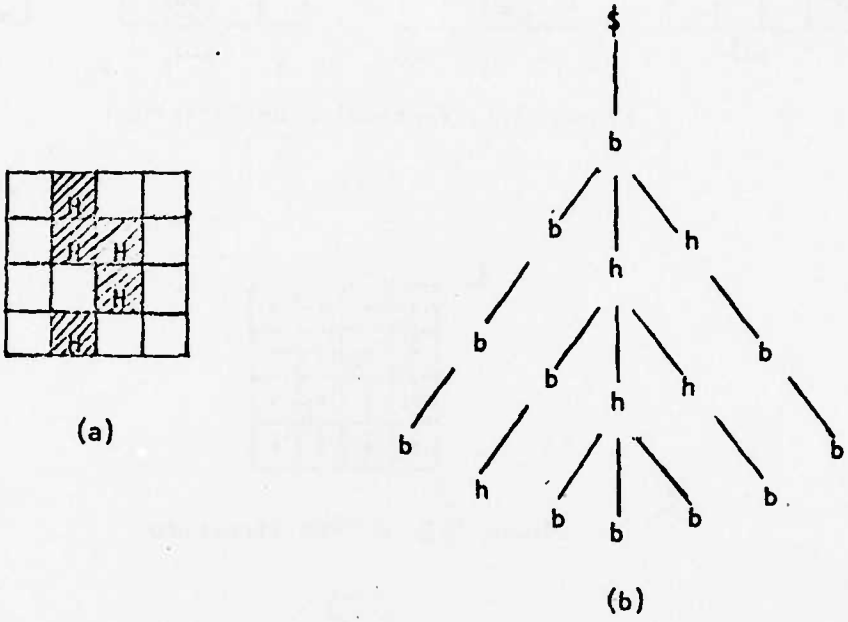


Figure 3.4 A Noisy Pattern (a) and its Tree Representation, α', (b).

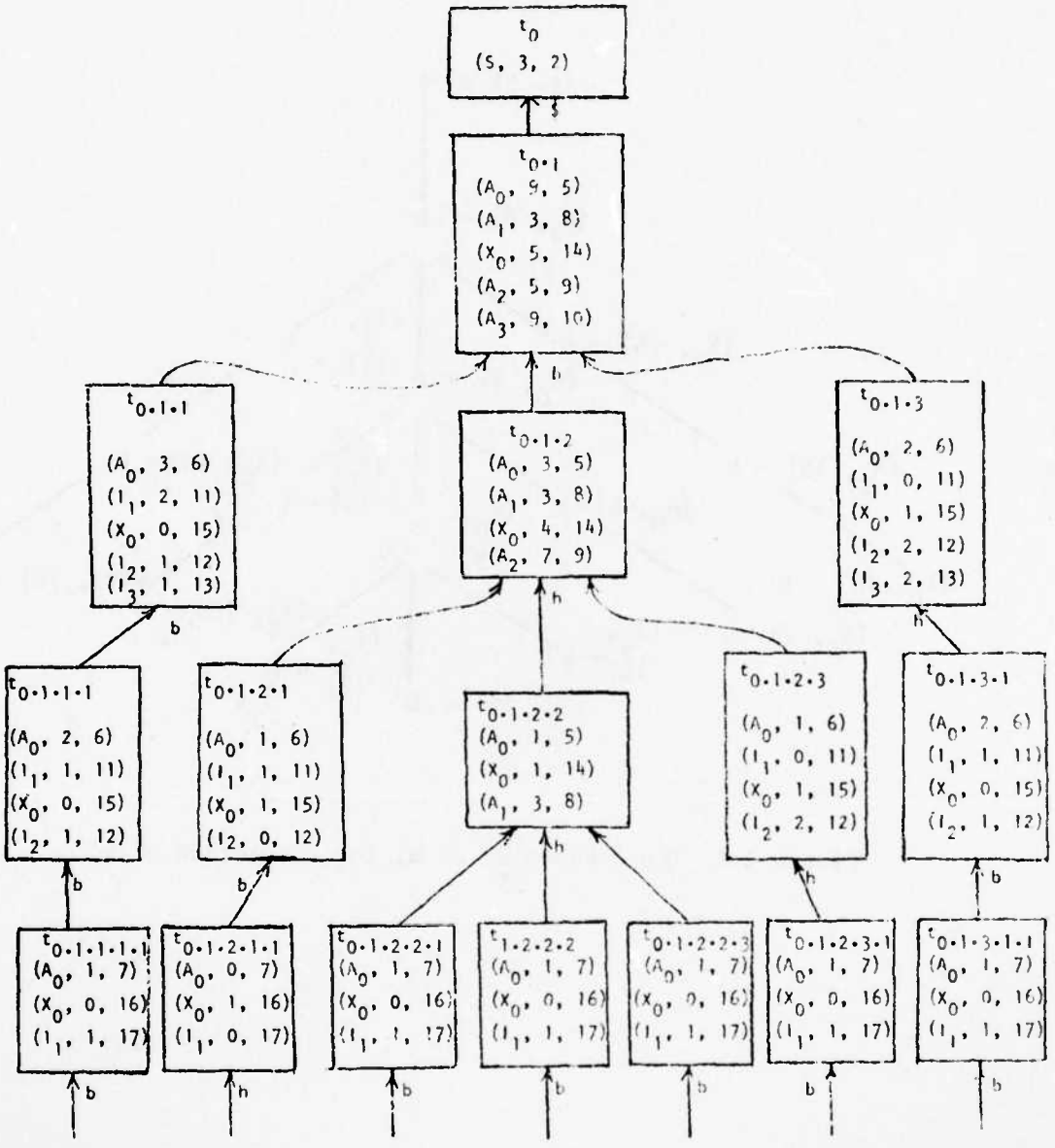


Figure 3.5 Transition Table of Tree α'

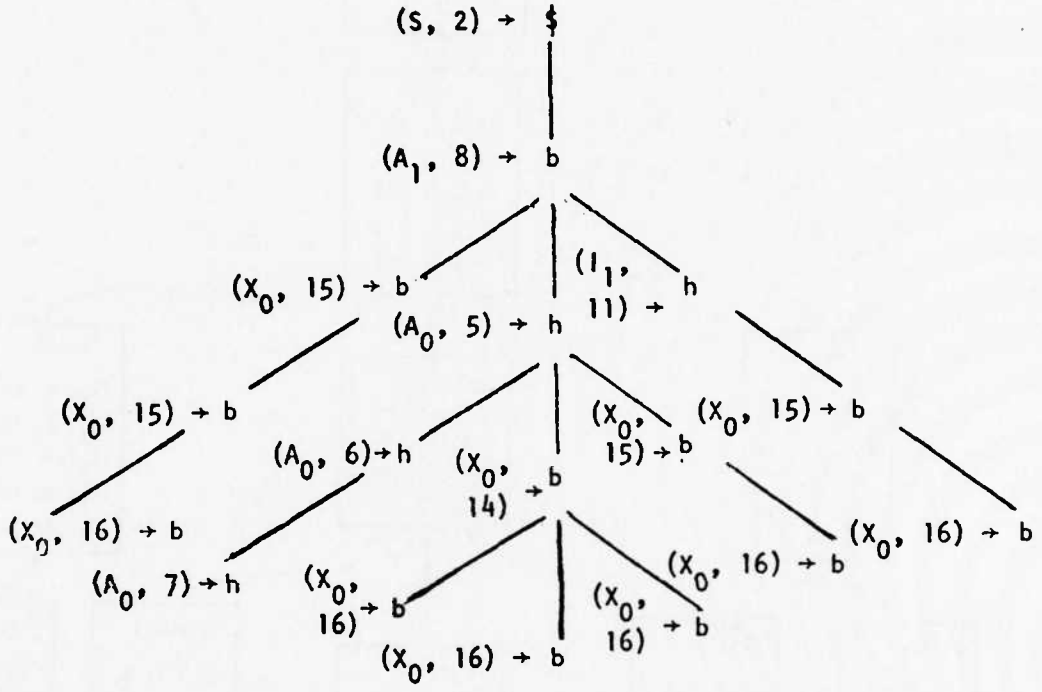
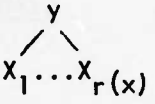
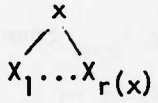


Figure 3.6 The generation of α, the correction of α'

$$q(\alpha'|\alpha) = q(t_1'|t_1) \ldots q(t_n'|t_n)\, q(x'|x) \tag{3.4}$$

We further have

$$\sum_{\alpha' \in T_\Sigma^D} q(\alpha'|\alpha) = 1 \quad \text{for any } \alpha \in T_\Sigma^D,\ D \subset U \tag{3.5}$$

For a given stochastic tree grammar $G_s = (V,r,P,S)$ over $<\Sigma,r>$, when the deformation probabilities, $q(y|x)$, are known for all $x \in \Sigma$, and $y \in \Sigma'$, the stochastic expanded grammar is $G_s' = (V',r',P',S)$ over $<\Sigma',r'>$ where $V' = (V-\Sigma)\ \Sigma'$ and $\Sigma' \supseteq \Sigma$ is the set of terminal symbols, and for all $y \in \Sigma'$, $X_0 \xrightarrow{p'} y$ is in P' if $X_0 \xrightarrow{p} x$ is in P, or $X_0 \xrightarrow{p}$  is in P', if $X_0 \xrightarrow{p}$  is in P and $p' = p\, q(y|x)$.

The language that generated from G_s' can be written as:

$$L(G_s') = \{(\alpha', p'(\alpha')) \mid \alpha' \in T_{\Sigma'},\ p'(\alpha') = \sum_{\substack{\alpha \in L(G_s) \\ D_{\alpha'} = D_\alpha}} q(\alpha'|\alpha)p(\alpha)\} \tag{3.6}$$

Suppose that the given noisy input tree α' is in tree domain D, the maximum-likelihood decision rule in this case is to choose a tree α in $L(G_s)$ of domain D, i.e., $\alpha \in L(G_s)$ and $\alpha \in T_\Sigma^D$, such that

$$q(\alpha'|\alpha)p(\alpha) = \max_\beta \{q(\alpha'|\beta)p(\beta) \mid \beta \in L(G_s)\ \ T_\Sigma^D\} \tag{3.7}$$

We call this value, $q(\alpha'|\alpha)p(\alpha)$, the probability of α' being a noise deformed tree of $L(G_s)$ and denote it as $q(\alpha'|G_s')$.

The structure-preserved maximum-likelihood SPECTA is given as follows:

Algorithm 3.3. Maximum-likelihood SPECTA

Input: (1) Stochastic tree grammar $G_s = (V,r,P,S)$ over $<\Sigma,r>$.

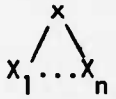
(2) Deformation probabilities, $q(y|x)$ for all $x \in \Sigma$, $y \in \Sigma'$.

(3) Input tree α'.

Output: Transition table of α' and $q(\alpha'|G_s')$.

Method:

Step 1. If $r[\alpha(a)] = 0$, $\alpha(a) = y$ then add to t_a, (X_0,p',k), if $X_0 \xrightarrow{p} x$ is the kth rule in P and $p' = p\,q(y|x)$.

Step 2. If $r[\alpha(a)] = n$, $n > 0$, $\alpha(a) = y$ then add to t_a, (X_0,p',k), if $X_0 \xrightarrow{p}$  is the kth rule in P and (X_1,p_1',k_1) $\in t_{a\cdot 1},\ldots,(X_n,p_n',k_n) \in t_{a\cdot n}$ then $p' = p_1'*\ldots*p_n'*p*q(y|x)$.

Step 3. Whenever more than one item in t_a have the same state, delete the item associated with smaller probability.

Step 4. If $(S,p',k) \in t_0$, then $q(\alpha|G_s') = p'$. If no item in t_0 is associated with the start state S, then no tree in $L(G_s)$ is in tree domain D_α. Input tree is rejected.

3.3.3 Application of SPECTA to LANDSAT Data Interpretation

Recently, syntactic methods have been used to analyze and interpret data obtained from the earth resource technology satellite (LANDSAT) [39,53]. The input data used in [39,53] are the results of pointwise classification [73]. Each pixel collected by LANDSAT represents a ground area of approximately $60 \times 70\ m^2$. According to spectral and/or temporal measurements of the object, a pixel is then classified into classes of water, cloud, downtown, concrete, or grass, etc. Due to the resolution size, spectral signals of smaller objects are usually composed of

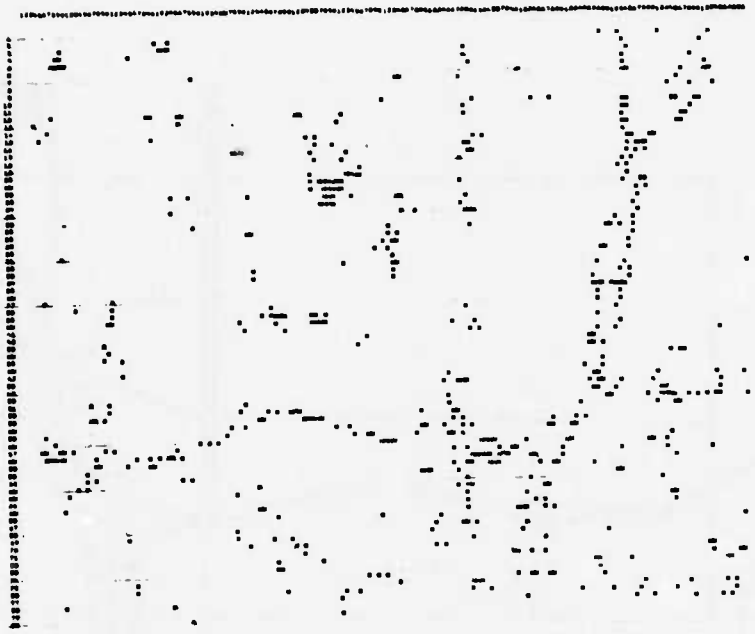


Figure 3.7 Pointwise classified highway data obtained from Grand Rapids, Michigan.

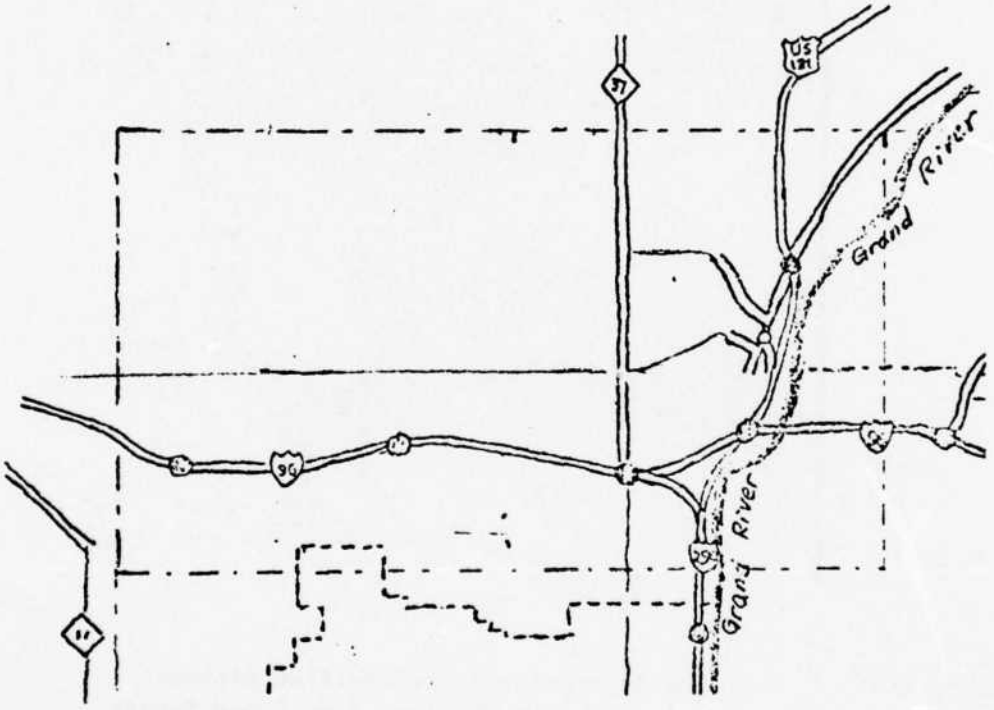


Figure 3.8 Divided highway map of northern part of Grand Rapids.

reflectance of several different kinds of ground cover. For instance, the spectral signal of a segment of a highway actually results from a combined reflectance of concrete surface, grass, and transportation vehicles. Consequently, the variation of size of smaller objects and their surroundings changes their reflectances, and thus, their spectral properties from point to point. This uncertainty causes some difficulty in setting threshold for classification based on spectral information of individual points only. One example of the results of pointwise classified highway patterns is given in Figure 3.7 which covers the area of the northern part of Grand Rapids,Michigan. Each symbol "H" represents a pixel that is classified as a segment of highway. Figure 3.8, which is obtained from the official highway map, indicates the major divided highways of the same area.

As illustrated in Figure 3.7, the inadequate resolution of highways and the mass of scattered concrete and grass-mixed objects other than highways result in discontinuity of highways and spurious points from pointwise classification. Syntactic methods have the advantage of using contextual and structural information contained in patterns for recognition purpose. In order to discriminate highways or rivers from other objects having similar spectral properties a tree system approach is proposed by Li and Fu [39]. The method demonstrates fairly good results in recognizing rivers among watery areas and some modern buildings with glass walls having similar reflecting surfaces as water, but poor in analyzing highway patterns. Evidently, highway patterns in some cases are too noisy to be effectively analyzed by a conventional training and parsing methods.

In this experiment, an input picture is processed window by window, where window size is 8 x 8 array of pixels. The labels on single pixels

are used as primitives. Thus, we have two kinds of primitives: "h" represents a highway pixel and "b" represents a blank or nonhighway pixel. Initially, sample patterns of vertical, horizontal, or diagonal line are used as our positive samples (or desired patterns). Assume that only a single segment of highway or at most two intersected highways that appeared within a window is considered. Hence, some patterns consisting of two such sample patterns are also added to the set of positive samples.

Similar to the procedure illustrated in Example 3.1, an array of primitives in a window are represented as a tree. Each primitive becomes a labeled node in the tree representation. We fix the tree domain to be D_H, and allow node label to be either "h" or "b". Totally there are 2^{64} tree representations in $T_\Sigma^{D_H}$, where $\Sigma = \{b,h,\$\}$, and \$ is the start terminal. Hence, the set of all possible patterns in an 8 x 8 window and the set of all labeled trees in $T_\Sigma^{D_H}$ are one-to-one correspondence. Figure 3.9 illustrates the correspondence between points in the 8 x 8 array and nodes in the tree domain D_H. The set of positive sample patterns can now be transformed into a set of positive sample trees of domain D_H. The construction of a pattern grammar, when error-correcting parser is used as recognizer, is a little different from the regular procedure [73]. With some presumed patterns as positive samples, the language generated from the constructed grammar must be as close to the set of positive samples as possible. Otherwise, there is always the possibility that the noisy pattern finds its best match in the set of "unwanted sentences" (sentences in L(G) but not in the set of positive samples).

The tree grammar, G_H, constructed to generate positive sample trees is given in Appendix C. Figure C.1 in Appendix C illustrates some

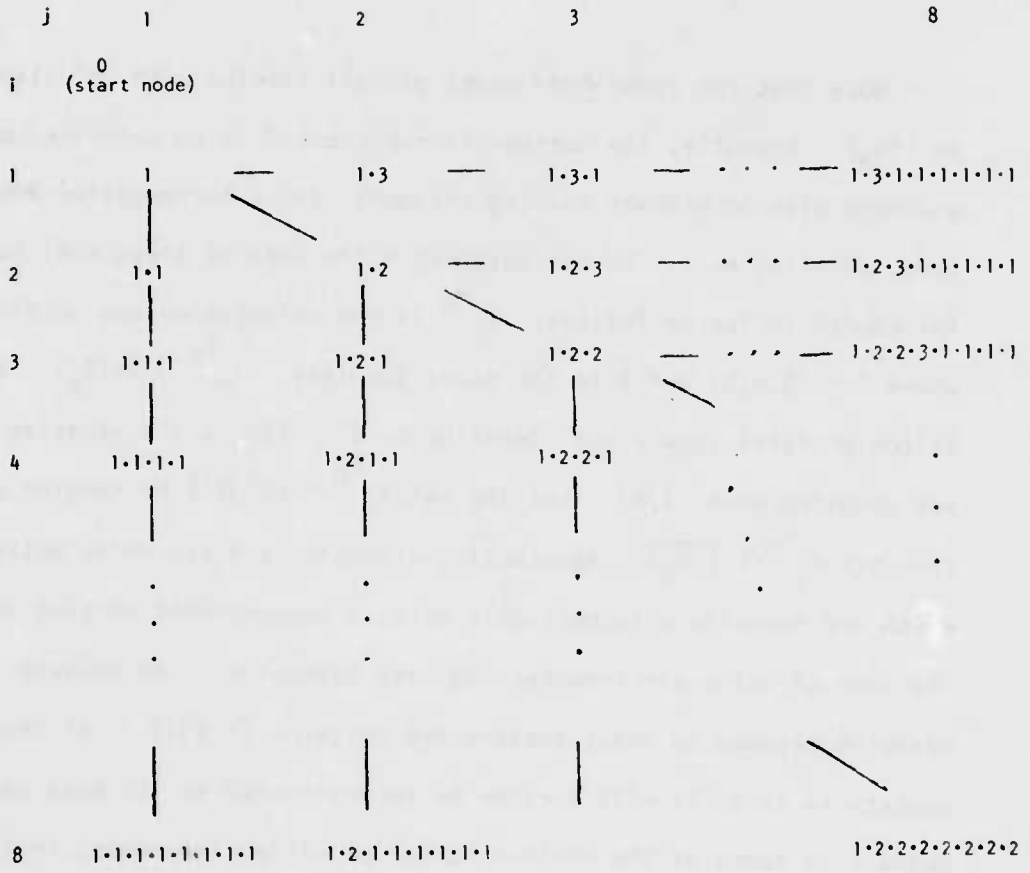


Figure 3.9 Nodes in tree domain D_H and their corresponding positions in the 8 x 8 array.

typical patterns whose tree representations are in $L(G_H)$. In Figure C.1, "Group A" denotes the group of patterns that are generated from rules of the form

$$S \rightarrow \begin{array}{c}\$\\ |\\ A_1\end{array} \quad , \quad 1 = 1...8, \text{ "Group B" are from } S \rightarrow \begin{array}{c}\$\\ |\\ B_1\end{array} \quad \text{etc.}$$

Note that the tree whose nodes are all labeled with "b" also belongs to $L(G_H)$. Actually, the pattern it represented is an undesired one, i.e., a window with no highway passing through. It is our negative sample tree, denoting as λ. We may categorize the sets of trees that have been introduced so far as follows: $T_\Sigma^{D_H}$ is the universe we are working on, where $\Sigma = \{\$,h,b\}$ and $\$$ is the start terminal. $T_\Sigma^{D_H} \cap (L(G_H) - \{\lambda\})$ is the positive sample set, denoting as S^+. $\{\lambda\}$ is the negative sample set denoting as S^- [74]. Let the set $T_\Sigma^{D_H} - (S^+ \cup S^-)$ be denoted as N, then $N = T_\Sigma^{D_H} \cap \overline{L(G_H)}$. Apparently, elements in N are noisy patterns, which are normally unrecognizable using a conventional parsing scheme. The idea of using error-correcting tree automaton is to measure the distance between an input pattern and patterns in $S^+ \cup S^-$. If the input pattern is in N, it will further be reconstructed to its best matching pattern in terms of the minimum number of mislabeled nodes, in S^+, or erased if its best matching pattern is in S^-. Since S^- is also in $L(G_H)$, we may measure the distance of the input tree with S^+ and S^- at the same time.

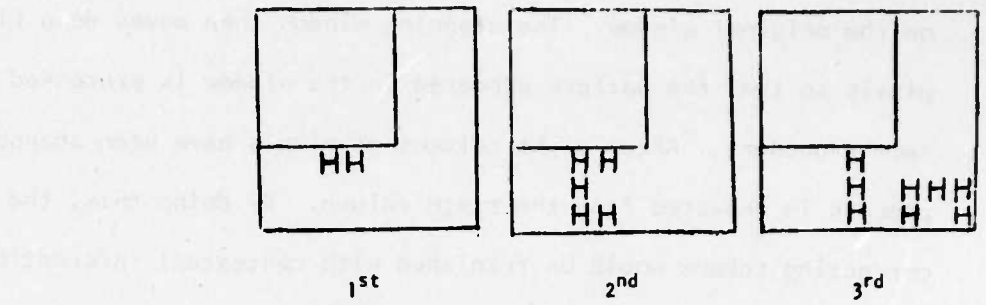
Assume that an input picture is scanned column by column from left to right. After an 8 x 8 array of primitives has been processed by SPECTA, we erase the top left 5 x 5 array of pixels, i.e., replace "H" points by

blanks of the original pattern and superimpose the reconstructed pattern on the original window. The scanning window then moves down five rows of pixels so that the pattern appeared in the window is processed by the same procedure. After eight columns of pixels have been scanned, the process is repeated from the sixth column. By doing this, the error-correcting scheme would be furnished with contextual information, not only within the window, but between the window and its surroundings. Using the pattern shown in column 141-148, row 1-18 of Figure 3.7, the window-by-window correcting process is illustrated in Figure 3.10. The flowchart of the entire recognition procedure is given in Figure 3.11.

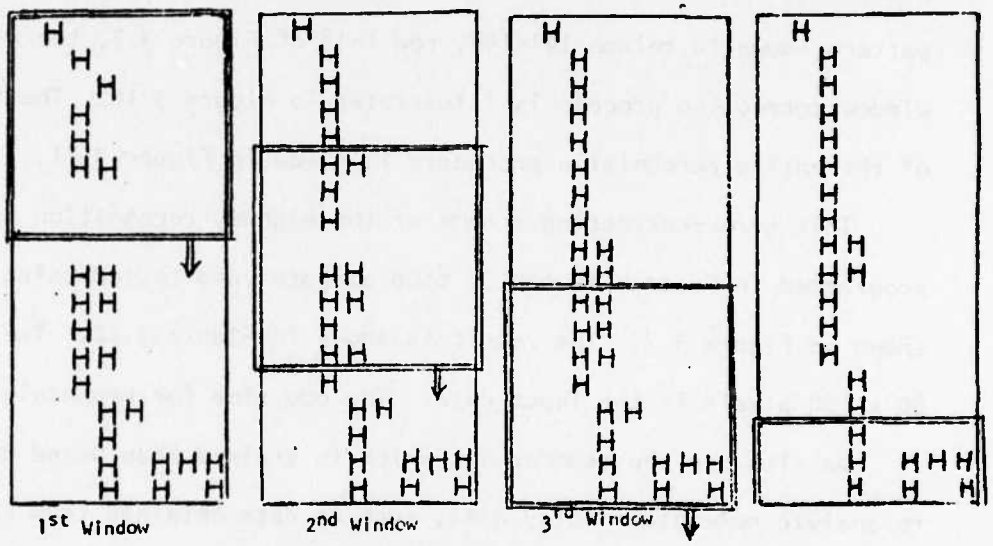
This error-correcting scheme of the highway recognition problem is programmed in Fortran IV on CDC 6500 computer and tested using the data shown in Figure 3.7. The result is shown in Figure 3.12. There are 80 x 160 pixels in the input data. The cpu time for processing is 150 sec.

We also use the grammar G_H, which is trained from Grand Rapids data to analyze some other noisy data, such as data obtained from Lafayette, Indiana. The pointwise classified data of Lafayette is shown in Figure 3.13 which contains 125 x 125 pixels. The result of the error-correcting analysis is shown in Figure 3.14. The cpu time used is 101 sec. For comparison, we use the highway map shown in Figure 3.15 as ground truth.

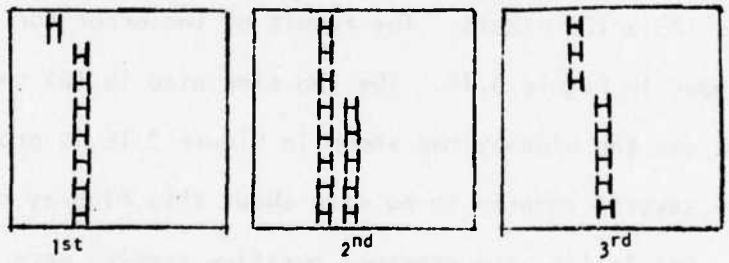
There are several remarks to be made about this highway recognition example. (1) Originally, the presumed positive samples were more than those generated from G_H in Appendix C. Many patterns of two intersected highways are considered. After the originally constructed grammar was tested by Grand Rapids data, we further removed production rules that were infrequently (or not) accessed and reduce the highway grammar to G_H in Appendix C.



(a) Erased Windows



(b) Before and After Window Scanning



(c) Corrected Windows

Figure 3.10 Window-by-window correcting process

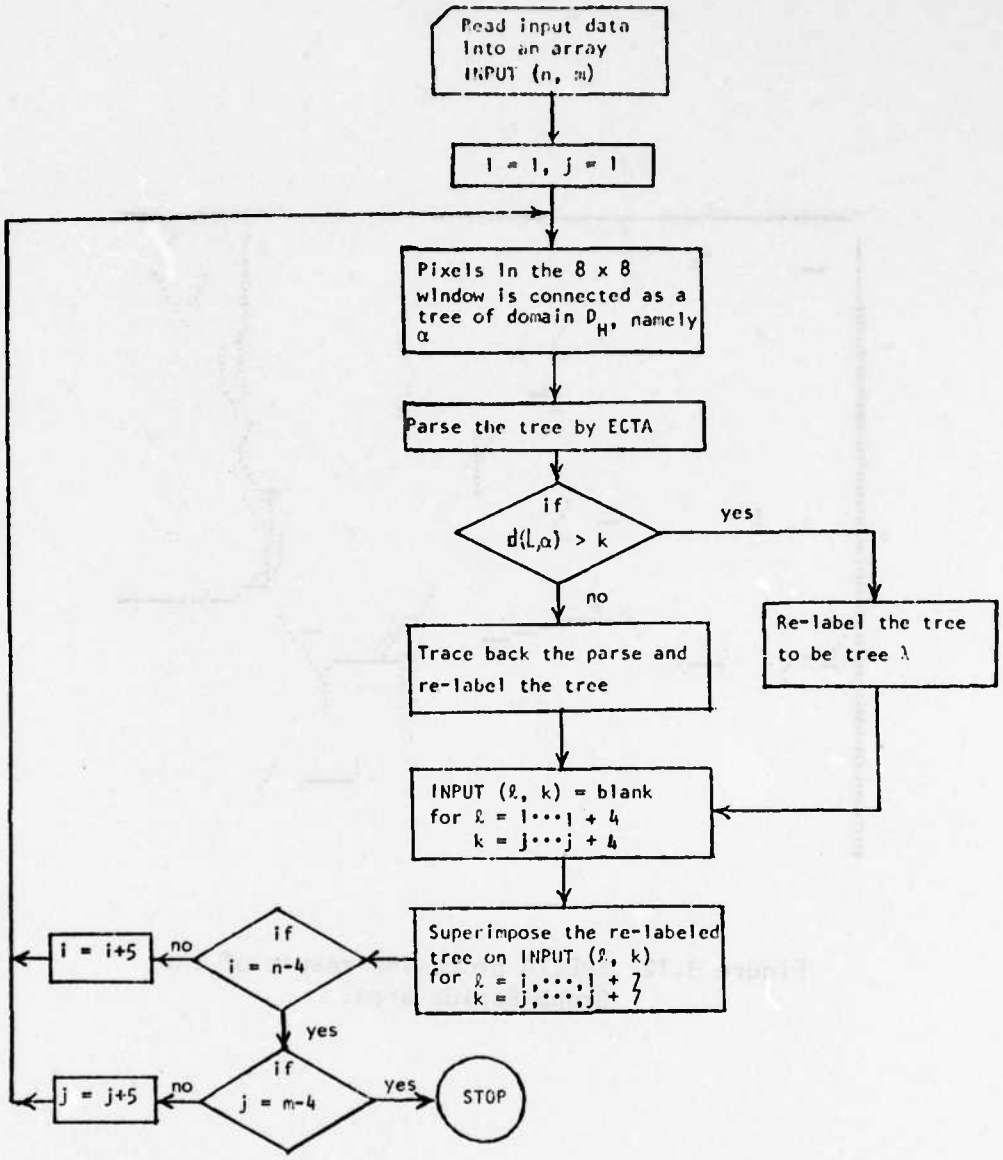


Figure 3.11 Flow chart of highway recognition using SPECTA

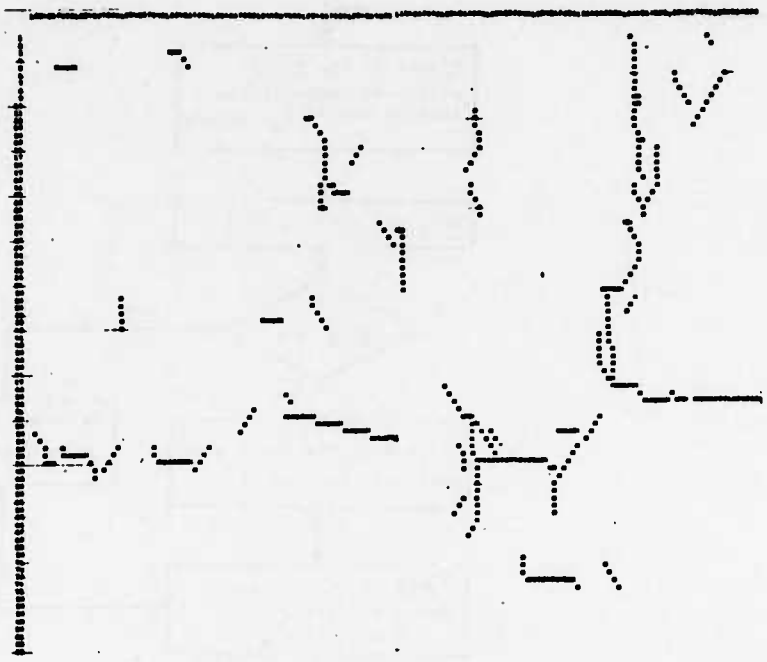


Figure 3.12 SPECTA processed result of the Grand Rapids area.

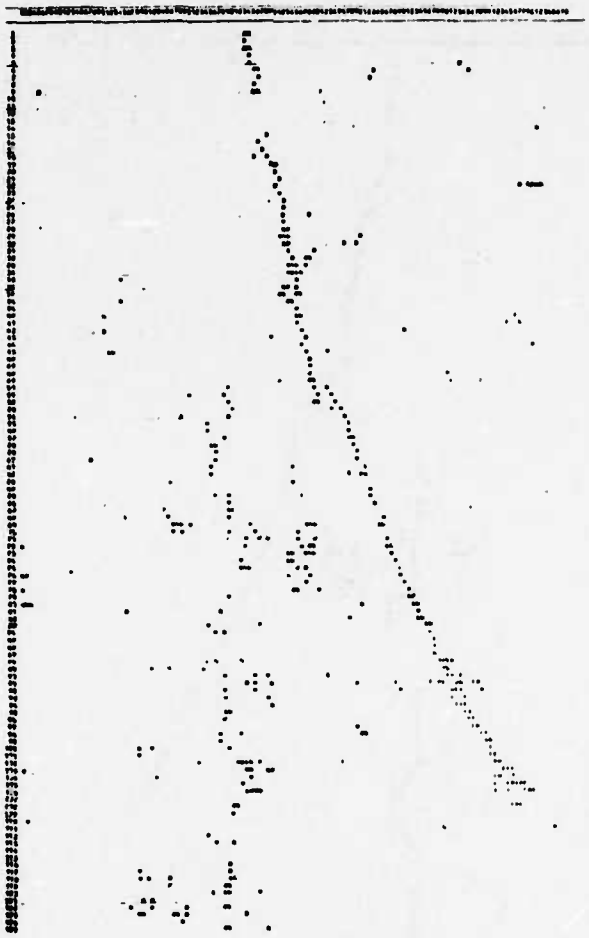


Figure 3.13 Pointwisely classified data of the Lafayette area.

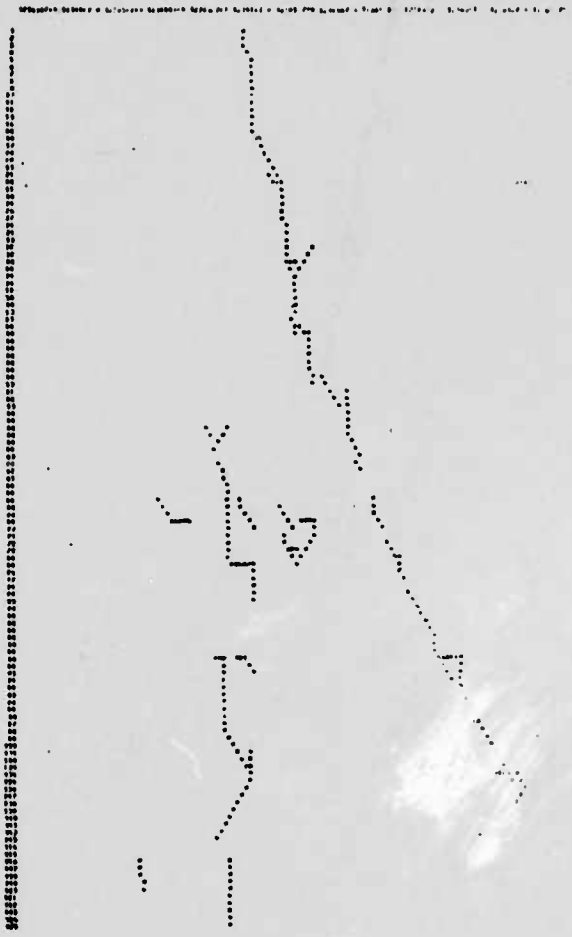


Figure 3.14 SPECTA processed result of the Lafayette area.

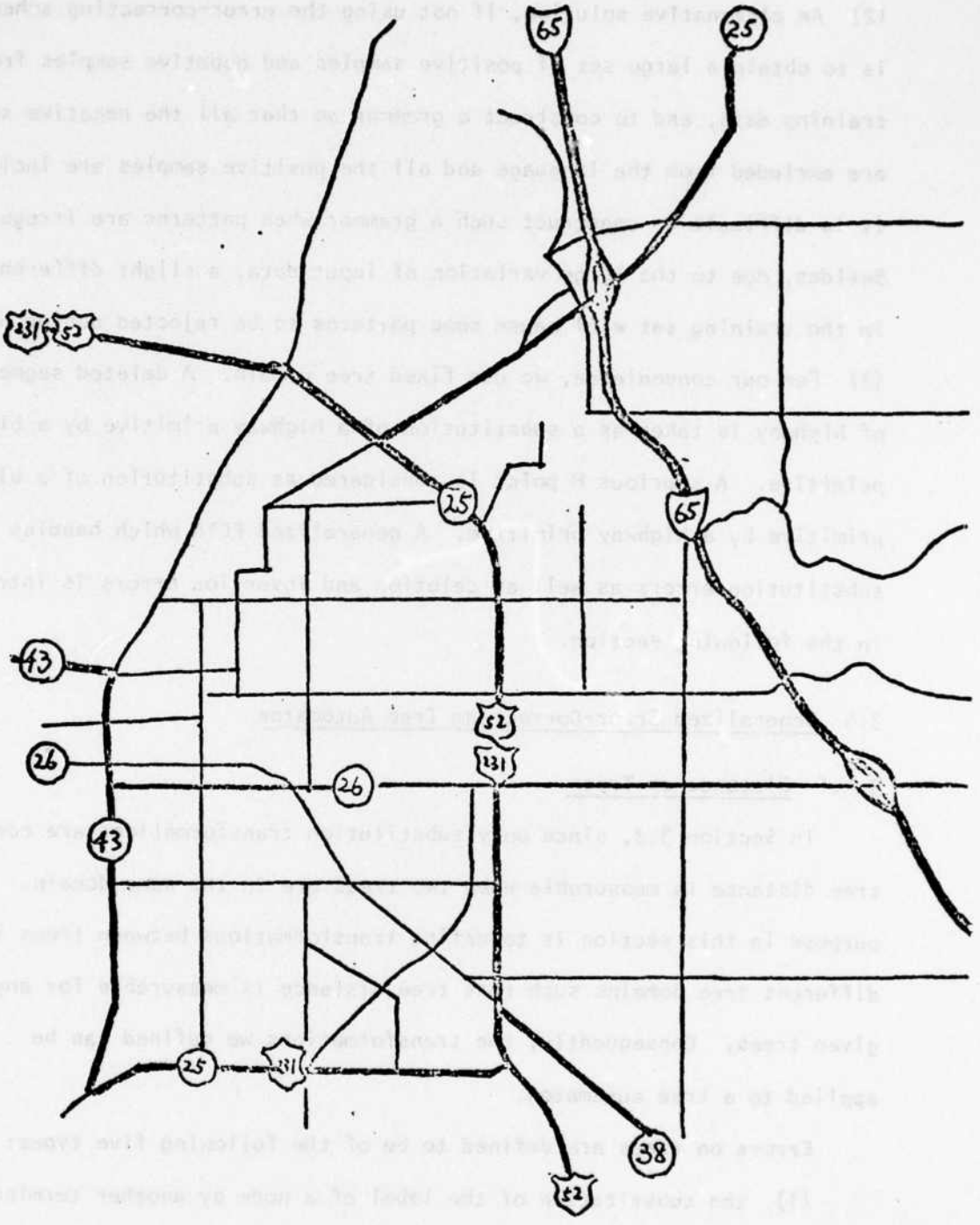


Figure 3.15 Highway map of Lafayette, Indiana.

(2) An alternative solution, if not using the error-correcting scheme, is to obtain a large set of positive samples and negative samples from training data, and to construct a grammar so that all the negative samples are excluded from the language and all the positive samples are included. It is difficult to construct such a grammar when patterns are irregular. Besides, due to the large variation of input data, a slight difference in the training set will cause some patterns to be rejected during parsing.
(3) For our convenience, we use fixed tree domain. A deleted segment of highway is taken as a substitution of a highway primitive by a blank primitive. A spurious H point is considered as substitution of a blank primitive by a highway primitive. A generalized ECTA which handles substitution errors as well as deletion and insertion errors is introduced in the following section.

3.4 Generalized Error-Correcting Tree Automaton

3.4.1 Distance on Trees

In Section 3.3, since only substitution transformations are considered, tree distance is measurable when two trees are in the same domain. Our purpose in this section is to define transformations between trees in different tree domains such that tree distance is measurable for any two given trees. Consequently, the transformations we defined can be applied to a tree automaton.

Errors on trees are defined to be of the following five types:

(1) the substitution of the label of a node by another terminal symbol,

(2) the insertion of an extraneous labeled node between a node and its immediate predecessor.

(3) the insertion of an extraneous labeled node to the left of all the immediate successors of a node,

(4) the insertion of an extraneous labeled node to the right of a node,

(5) the deletion of a node of rank 1 or 0.

The three operations of insertions in rule (2), (3) and (4) are named as stretch, branch, and split, respectively, according to the relative position of the inserted node to the original tree. Apparently, the inverse operation of any type of insertion is deletion, and the inverse of deletion operation is one of the three types of insertion.

We define five transformations S, T, P, B, D, from T_Σ to the subsets of T_Σ, to describe substitution, stretch, split, branch, and deletion errors respectively. Let α be a tree over $<\Sigma,r>$, $a\in U$, $y\in\Sigma$, and $c\cdot i = a$, $r[\alpha(c)] = n$, then error transformations are defined as follows:

(1) $S_{a/y}(\alpha) = \alpha(a\leftarrow\{(0,y)\}\cup\{i\cdot\alpha/a\cdot i \mid 1 \le i \le r[\alpha(a)]\})$,

(2) $T_{a/y}(\alpha) = \alpha(a\leftarrow\{(0,y)\}\cup\{1\cdot\alpha/a\})$,

(3) $B_{a/y}(\alpha) = \alpha(a\leftarrow\{(0,\alpha(a)),\ (1,y)\}\cup\{i+1\cdot\alpha/a\cdot i \mid 1 \le i \le r[\alpha(a)]\})$,

(4) $P_{a/y}(\alpha) = \alpha(c\cdot n+1 \leftarrow \alpha/c\cdot n)\ldots(c\cdot i+1 \leftarrow \alpha/c\cdot i)\cup(a \leftarrow (0,y))$,

(5) $D_{a/y}(\alpha) = \begin{cases} \alpha(c\cdot 1 \leftarrow \alpha/c\cdot 1+1)\ldots(c\cdot n-1 \leftarrow \alpha/c\cdot n)\cup(c\cdot n \leftarrow \lambda) & \text{if } r[\alpha(a)] = 0 \\ \alpha(c \leftarrow \alpha/a) & \text{if } r[\alpha(a)] = 1 \end{cases}$

S, T, B, P and D transformations are illustrated in Figure 3.16 (a), (b), (c), (d) and (e), respectively.

We write $\alpha \vdash^{\Delta} \beta$, if β is in $\Delta(\alpha)$, where $\Delta\in\{S,T,B,P,D\}$, and further denote that $\alpha \vdash^{\Delta^k} \beta$ for $k \ge 0$, if β is derived from α by applying

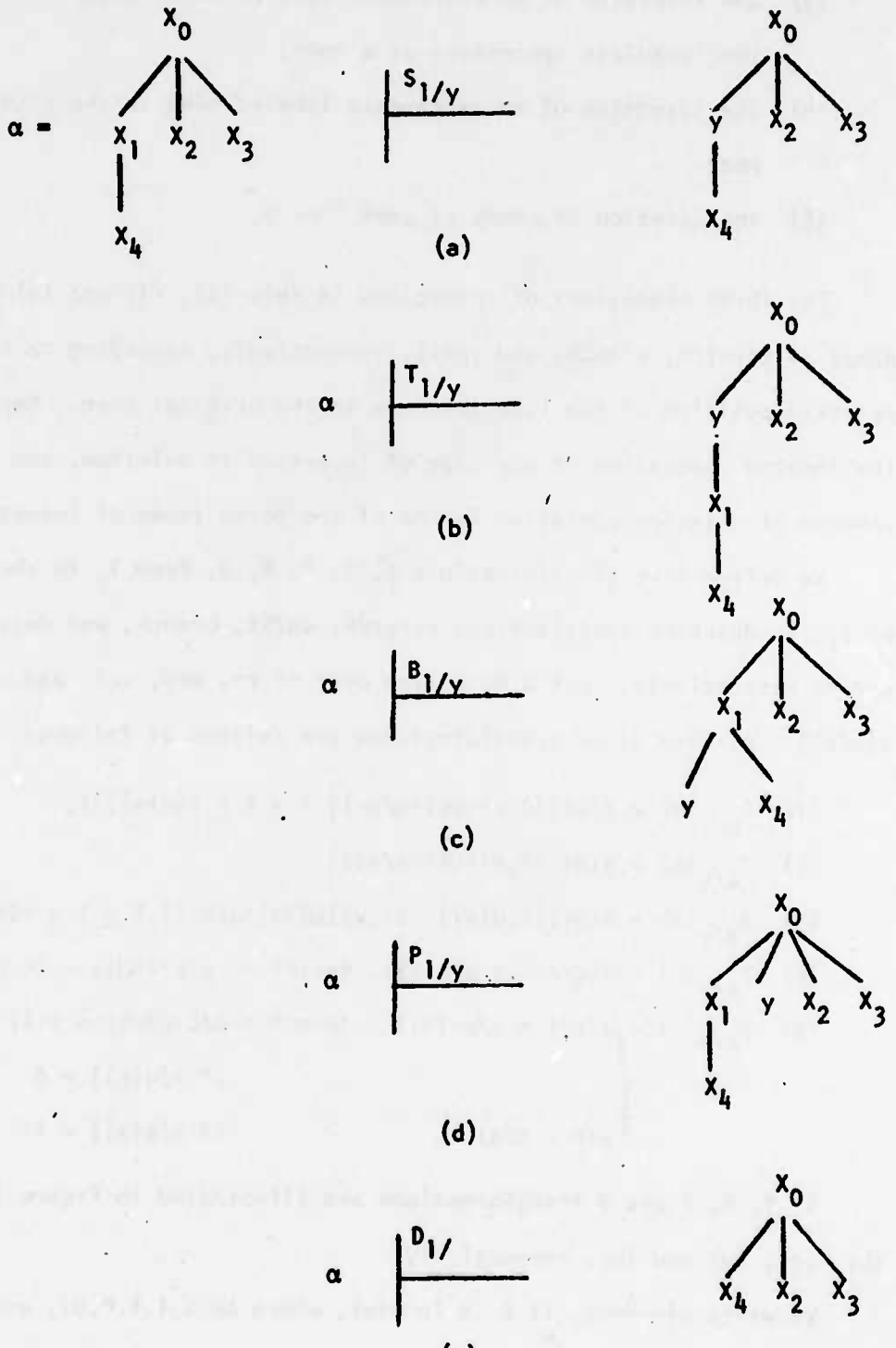


Figure 3.16 S, T, B, P, and D transformations on α.

k transformations, where Δ^k denotes the composition of Δ with itself k times. The distance on trees over Σ, $d(\alpha,\beta)$, is defined as the smallest integer k for which $\alpha \vdash^{\Delta^k} \beta$, if α and β are two trees in T_Σ.

For example, given two trees α and β, where $\alpha = \{(0,\$), (1,-), (1\cdot1,P), (2,\sim), (2\cdot1,q)\}$, $\beta = \{(0,\$), (1,V), (1\cdot1,P), (1\cdot2,q), (2,q)\}$, then $d(\alpha,\beta) = 3$, since $\beta = D_{2/}\ (P_{1\cdot1/q}\ (S_{1/v}(\alpha)))$ and no other derivation of β from α costs transformations less than three. Trees α and β are shown in Figure 3.17.

Let L be a tree language, a tree β not in L can be derived from some tree in L by a sequence of error transformations. The distance between β and L is defined as,

$$d(L,\beta) = \min_{\alpha} \{d(\alpha,\beta) \mid \alpha \in L\} \qquad (3.8)$$

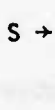
Example 3.2. Assume a directed graph of labeled verticesand unlabeled branches as shown in Figure 3.18, the tree grammar constructed to generate such patterns is given as follows:

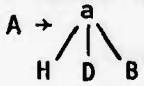
$G_t = (V,r,P,S)$

$V = \{S,A,B,C,D,E,H,\$,a,b,c,d,e,h\}$

$r(\$) = \{1\}$, $r(a) = \{0,3\}$, $r(b) = \{0,3,4\}$, $r(c) = \{0,2\}$

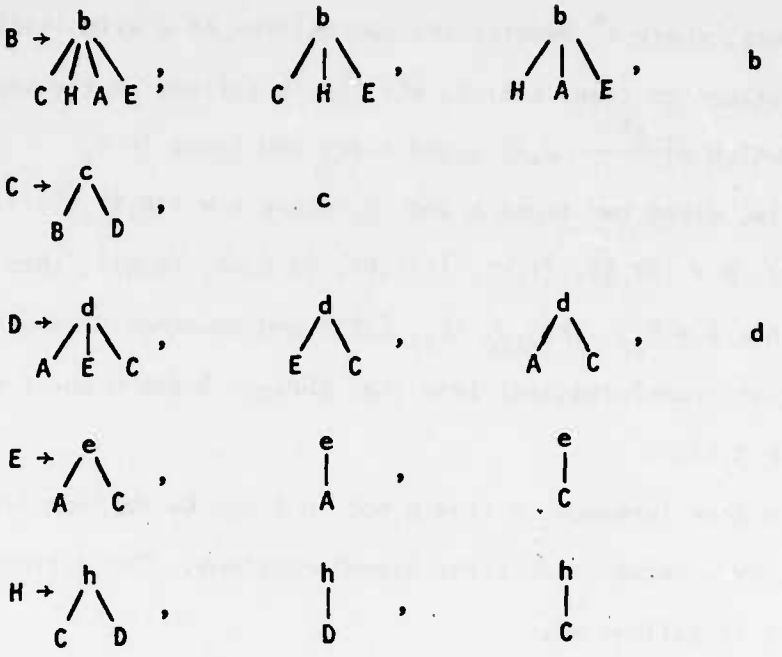
$r(d) = \{0,2,3\}$, $r(e) = \{1,2\}$, $r(h) = \{1,2\}$

$S \rightarrow$ 

$A \rightarrow$ 

, 

, a



Suppose that α' is the given distorted pattern. The successive graphs after each tree error transformation is applied are illustrated in Figure 3.19.

3.4.2 The Formulation of GECTA

For any given α' not in language L, the generalized error-correcting tree automaton (GECTA) is formulated to search for α in L such that the distance of α from α' is the smallest among all the sentences in L. That is,

$$d(\alpha,\alpha') = \min_{\beta \in L} d(\beta,\alpha') \qquad (3.9)$$

Note that the condition $D_\beta = D_{\alpha'}$ in equation (3.1) is removed here.

Before introducing the algorithms for GECTA, a normal form of trees and tree grammars, called the binary form, is defined.

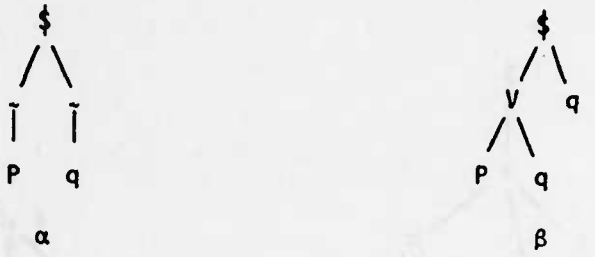


Figure 3.17 $\beta = D_{2/}\,(P_{1.1/q}(S_{1/v}(\alpha)))$.

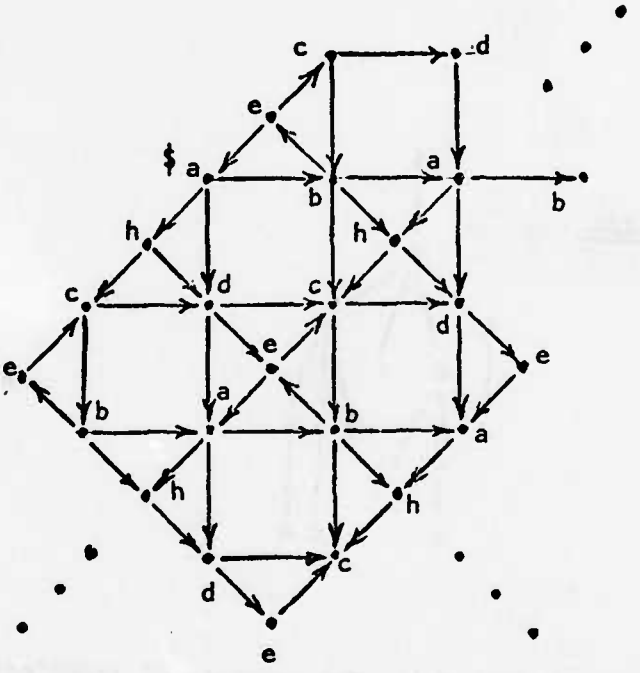


Figure 3.18 A directed graph

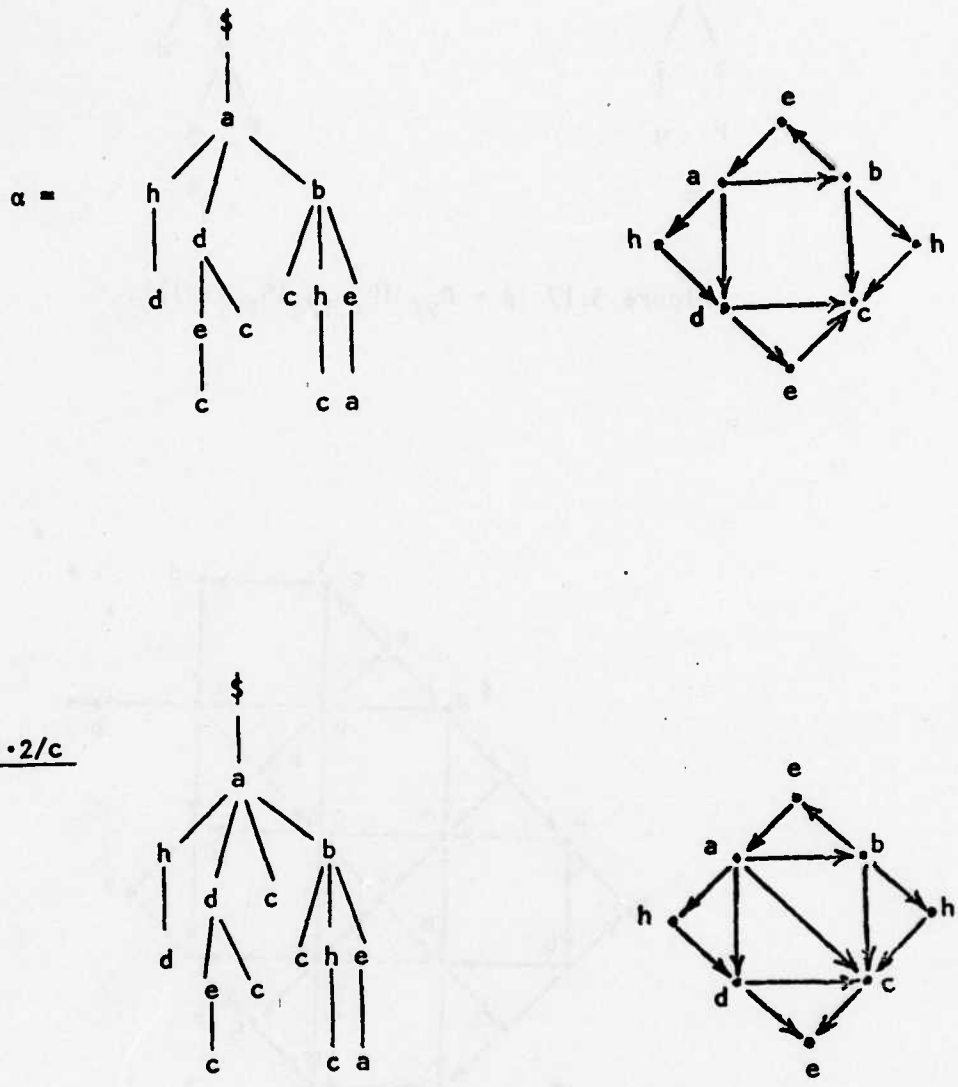


Figure 3.19 The sequence of transformation of α' from α

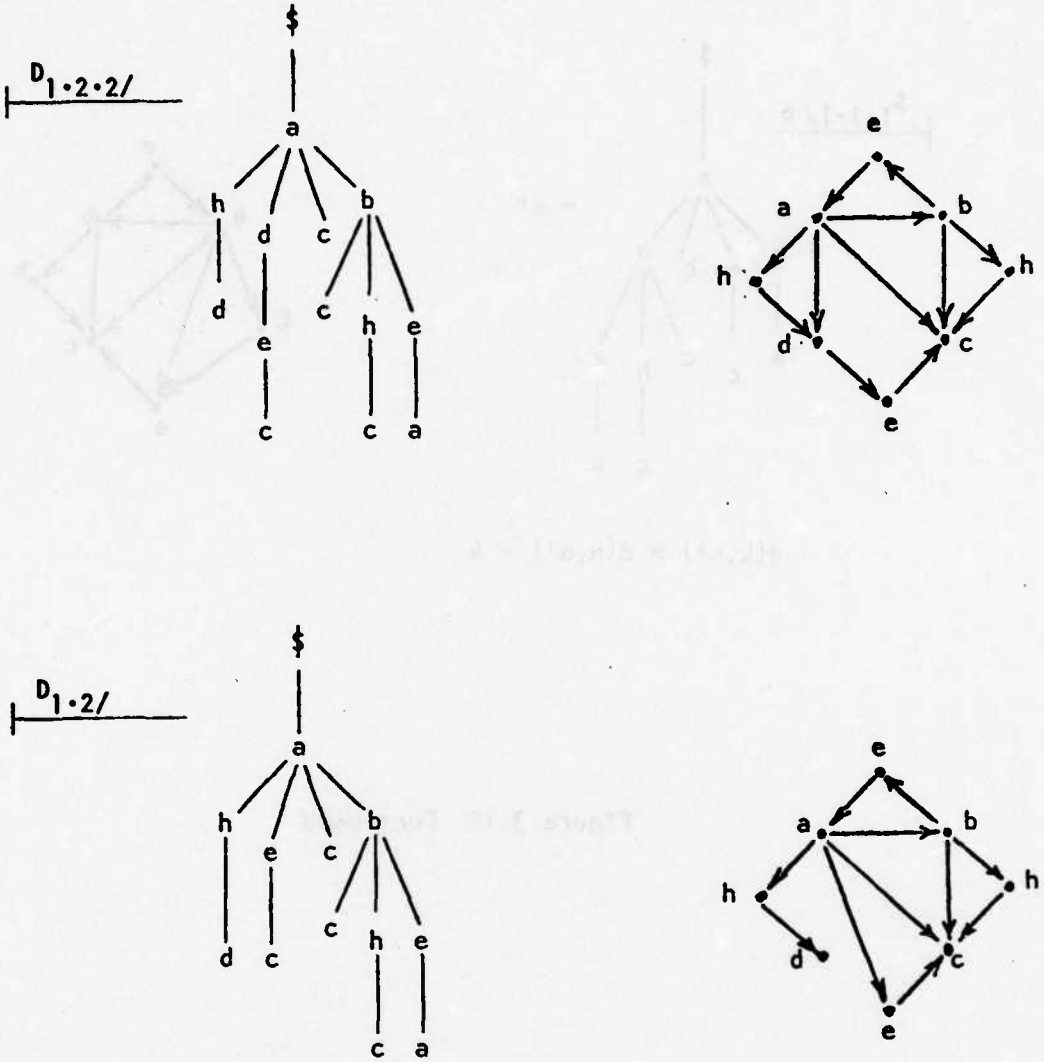
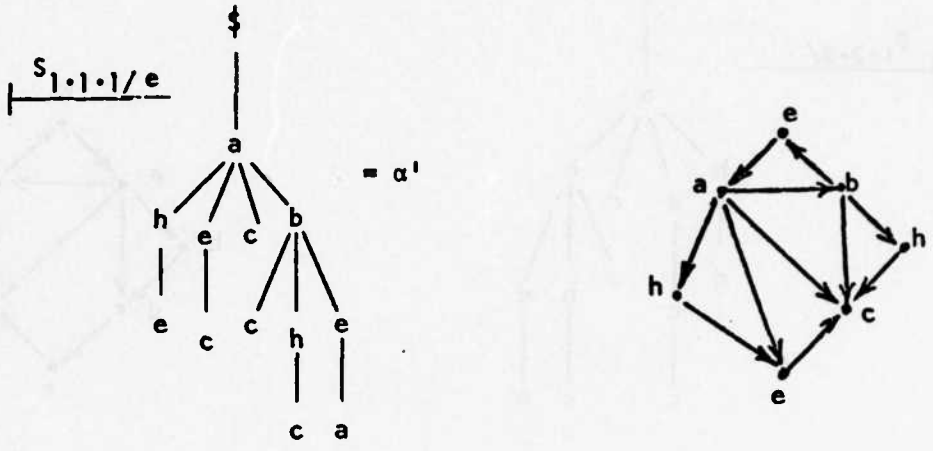


Figure 3.19 Continued



$d(L,\alpha') = d(\alpha,\alpha') = 4$

Figure 3.19 Continued

(A) Tree Binary Form

A binary tree grammar is defined as follows:

Definition 3.17. A tree grammar $G_b = (V_b, r_b, P_b, S)$ over $\langle \Sigma_b, r_b \rangle$ is said to be in binary form if,

(1) A pseudo symbol * is in Σ_b.

(2) $r_b = \{0,1,2\}$

(3) Each production in P_b is in one of the following forms:

(a) $U_1 \rightarrow X_1\ *.$

(b) $U_2 \rightarrow U_1 X_1\ *$

(c) $X_0 \rightarrow U_1 X_1\ x$

(d) $X_0 \rightarrow X_1\ x$

(e) $X_0 \rightarrow x$

where U_1, U_2, X_0, X_1 are in $V_b - \Sigma_b$, $x \epsilon \Sigma_b - \{*\}$.

Let α be a tree in T_Σ, and * be a pseudo symbol not in Σ, the conversion of tree α into its binary form α^* is given in the following Algorithm.

Algorithm 3.4. Binary Form Conversion.

Input: Tree α in T_Σ.

Output: α^*, binary form of α.

Method: Repeat CONVERT until α is in binary form.

<CONVERT>: (1) if $t_0 = x$ is a term in α, then t_0 is said to be in binary form.

(2) if $t_1 \ldots t_n$ are already in binary form, and $t_0 = (t_1 \ldots t_n)\ x$ is a term in α, $x \epsilon \Sigma_n$ then t_0 is said to be in binary form if $n = 1$, then $t_0 = (t_1)\ x$

If $n > 1$, then define $t_1^* \ldots t_{n-1}^*$ such that

$t_1^* = (t_1)^*$

$t_2^* = (t_1^* \; t_2)^*$

.

.

.

$t_{n-1}^* = (t_{n-2}^* \; t_{n-1})^*$

$t_0 = (t_{n-1}^* \; t_n)x$

The construction of the binary tree grammar $G_b = (V_b, P_b, r_b, S)$ from a given tree grammar $G_t = (V, r, P, S)$ in expansive form is as follows:

Let $\Sigma_b = \Sigma \cup \{*\}$, add V to V_b. P_b is constructed as follows:

Step 1. For each production in P of the form $X_0 \rightarrow x$, $x \in \Sigma_0$, add $X_0 \rightarrow x$ to P_b and $r_b(x) = 0$,

Step 2. For each production in P of the form $X_0 \rightarrow X_1 x$, $x \in \Sigma_1$, add $X_0 \rightarrow X_1 x$ to P_b and $r_b(x) = 1$,

Step 3. For each production in P of the form $X_0 \rightarrow X_1 \ldots X_n x$, $x \in \Sigma_n$, $n \geq 2$, add,

$U_{01} \rightarrow X_1^*$

$U_{02} \rightarrow U_{01} X_2^*$

.

.

.

$U_{0n-1} \rightarrow U_{0n-2} X_{n-1}^*$

$X_0 \rightarrow U_{0n-1} X_n x$

to P_b, and $r_b(*) = \{1,2\}$, $r_b(x) = 2$. Add $\{U_{01} \mid 1 \leq 1 \leq n-1\}$ to V_b.

Let the binary form of a tree language $L(G_t)$ be denoted as $L_b(G_t)$ and G_b be the binary form of grammar G_t, then $L_b(G_t) = L(G_b)$.

Example 3.3 Character E

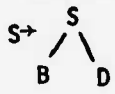
A character E is shown in Figure 3.21 (a). Based on the pattern primitives defined in Figure 3.20, the tree representation α of E is shown in Figure 3.21 (b). The tree grammar G_t which generates the character E of different sizes is as follows:

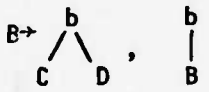
$G_t = (V,r,P,S)$ over $<\Sigma,r>$

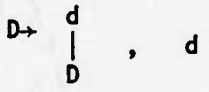
$V = \{S,B,C,D,\$,b,d\}, \Sigma = \{\$,b,d\}$

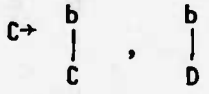
$r(\$) = 2,\ r(b) = \{1,2\},\ r(d) = \{0,1\}$

P:

$S \rightarrow$ 

$B \rightarrow$ 

$D \rightarrow$ 

$C \rightarrow$ 

The binary from of α, α^*, is given in Figure 3.22 and the binary form of G_t, G_b, is:

$G_b = (V_b,r_b,P_b,S)$ over $<\Sigma_b,r_b>$

$V_b = \{S,U_S,B,U_B,D,C,\$,b,d,*\},\ \Sigma_b = \Sigma \cup \{*\}$

$r_b(\$) = \{2\},\ r_b(b) = \{1,2\},\ r_b(d) = \{0,1\},\ r_b(*) = \{1\}.$

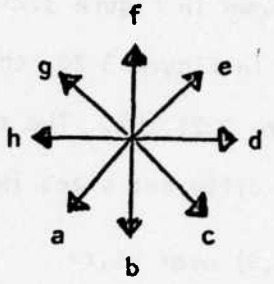


Figure 3.20 Pattern primitives

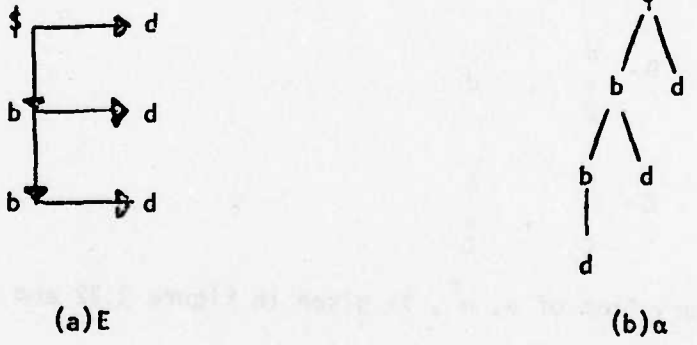


Figure 3.21 Character E (a) and its tree representation (b)

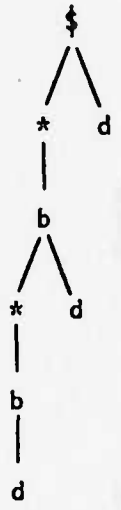
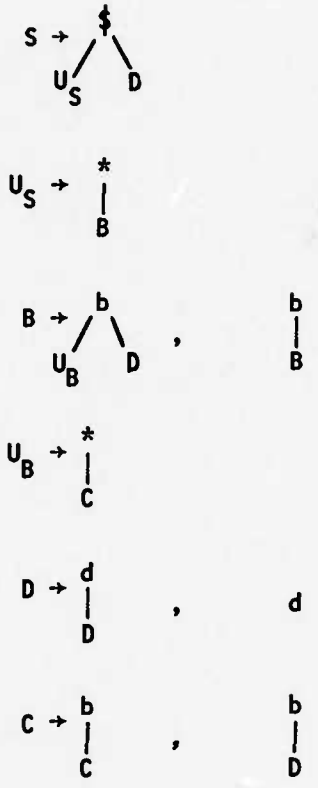


Figure 3.22 α^*, binary form of α

P_b:



(B) GECTA

Let $M_b = (V_b-\Sigma_b, F, \{S\})$ be a tree automaton that accepts a tree grammar in binary form $G_b = (V_b, r_b, P_b, S)$ over Σ_b, where F is the set of transition functions on $(V_b-\Sigma_b)^{r(X)}X(V_b-\Sigma_b)$, for all $x \in \Sigma_b$. By adding error transitions according to the transformations defined in Section 3.4.1, the expanded tree automaton that accept all the possible erroneous trees is constructed as follows:

Algorithm 3.5. Expanded Tree Automaton $\bar{M}_b$.

Step 1. $\bar{M}_b = (\bar{V}, \bar{F}, \{S\})$ where $\bar{V} = (V_b-\Sigma_b) \cup \{I\}$, $\bar{F} = F \cup F^S \cup F^D \cup F^I$, and F^S, F^D, and F^I are called substitution, deletion, and insertion error transitions, respectively.

Step 2. $F^S = \{f_y(\xi) \sim X_0, \sigma | F_x(\xi) \sim X_0 \in F, x \neq *, y \in \Sigma_b - \{x\}\}$

Step 3. $F^D = \{f(\xi) \sim X_0, \gamma | F_x(\xi) \sim X_0 \in F, x \neq *\} \cup$
$\{f(\xi) \sim X_0, 0 | F_*(\xi) \sim X_0 \in F\}$

Step 4.

(a) Add $f_y(X_0) \sim X_0, \delta$, to F^I for all $y \in \Sigma_b - \{*\}$ if $f_x(\xi) \sim X_0$ is in F and $x \neq *$.

(b) Add $f_x(1,X_1) \sim X_0, 0$, and $f_y(1,X_1) \sim X_0, \sigma$ or $f_x(1) \sim X_0, 0$, and $f_y(1) \sim X_0, \sigma$ to F^I if $f_x(X_1) \sim X_0$ or $f_x \sim X_0$ is in F, $X \in \Sigma_b$.

(c) Add $f_*(\xi) \sim X_0, 0$, $f_*(X_0,1) \sim X_0, 0$, $f_y(X_0,1) \sim X_0, \sigma$ and $f_x(X_0,1) \sim X_0, 0$ to F^I if $f_x(\xi) \sim X_0$ is in F for all $x \in \Sigma_b$.

(d) Add to F^I, $f_y(1) \sim 1, \delta$, and $f_y(1,1) \sim 1, \delta$, for all $y \in \Sigma_b$ and $f_y \sim 1, \delta$, for all $y \in \Sigma_b - \{*\}$

Where $\xi \in (V_b - \Sigma_b)^i$, $i = 1,2$.

The notations σ, γ, and δ associated with transitions represent costs of substitution, deletion, and insertion errors, respectively. Hence, weighted distance can be measured by using expanded tree automaton. (a), (b), and (c) in Step 4 introduce stretch, branch, and split operations, respectively.

The search algorithm for the least cost (minimum-distance) solution is to construct a tree-like transition table with all candidate states and their corresponding costs recorded. Each element in the transition table corresponds to a node in the tree domain of the input tree. Let it be denoted

as t_a if $a \in D_{\alpha'}$, and α' be the input tree, then a pair (X,η) is in t_a if X is a candidate state representing subtree α'/a, η is the minimum cost associated with X, the algorithm is given as follows:

Algorithm 3.6. Minimum-Distance GECTA

Input: Expanded Tree Automaton $\bar{M}_b = (\bar{V},\bar{F},\{S\})$ of a tree language $L(G_t)$, and input tree α'.

Output: $d(L(G_t),\alpha')$, and transition table of α'.

Method:

<SUBTREE(Start)>

(a) SUBTREE(Start) = $\{(X_0,\gamma) \mid f \sim X_0, \nu \in F^D\}$.

(b) Add (X_0,η) to SUBTREE(Start) if $f(X_1) \sim X_0$, $\gamma \in F^D$, $(X_1,\pi) \in$ SUBTREE(Start), then $\eta = \gamma + \pi$.

(c) Add (X_0,η) to SUBTREE(Start) if $f(X_1,X_2) \sim X_0$, $\gamma \in F^D$, (X_1,π_1) and $(X_2,\pi_2) \in$ SUBTREE(Start), then $\eta = \gamma + \pi_1 + \pi_2$.

(d) Whenever two or more items associated with the same state. Delete items with higher costs.

<SUBTREE(X,θ)>

(a) SUBTREE(X,θ) = $\{(X_0,\eta) \mid f(X) \sim X_0, \gamma \in F^D, \eta = \gamma + \theta\} \cup \{(X,\theta)\}$.

(b) Add (X_0,η) to SUBTREE(X,θ) if $f(Y) \sim X_0$, $\gamma \in F^D$, $(Y,\pi) \in$ SUBTREE(X,θ), $\eta = \gamma + \pi$.

(c) Add (X_0,η) to SUBTREE(X,θ) if $f(Y_1,Y_2) \sim X_0$, $\gamma \in F^D$, $(Y_1,\pi_1) \in$ SUBTREE(Start) and $(Y_2,\pi_2) \in$ SUBTREE(X,θ), or $(Y_1,\pi_1) \in$ SUBTREE(X,θ) and $(Y_2,\pi_2) \in$ SUBTREE(Start), then $\eta = \gamma + \pi_1 + \pi_2$.

(d) Delete redundant states.

Step 1. If $r[\alpha(a)] = 0$, $\alpha(a) = x$, then

(a) (X_0,η) is in t_a if $f_x(X) \sim X_0$, $\lambda \in F \cup F^S \cup F^I$, for all (X,π) in SUBTREE(Start), $\eta = \lambda + \pi$.

(b) (X_0,η) is added to t_a if $f_x(X_1,X_2) \sim X_0$, $\lambda \in F \cup F^S \cup F^I$, (X_1,π_1) and $(X_2,\pi_2) \in$ SUBTREE(Start), $\eta = \lambda + \pi_1 + \pi_2$.

(c) Delete redundant states.

Step 2. If $r[\alpha(a)] = 1$, $\alpha(a) = x$, then

(a) (X_0,η) is in t_a if $f_x(Y) \sim X_0$, $\lambda \in F \cup F^S \cup F^I$, for all $(Y,\pi) \in$ SUBTREE (X,θ) and $(X,\theta) \in t_{a\cdot 1}$, $\eta = \lambda + \pi$.

(b) (X_0,η) is added to t_a if $f_x(Y_1,Y_2) \sim X_0$, $\lambda \in F \cup F^S \cup F^I$, for all $(Y_1,\pi_1) \in$ SUBTREE(X,θ) and $(Y_2,\pi_2) \in$ SUBTREE(Start) or $(Y_1,\pi_1) \in$ SUBTREE(Start) and $(Y_2,\pi_2) \in$ SUBTREE(X,θ) where $(X,\theta) \in t_{a\cdot 1}$, then $\eta = \lambda + \pi_1 + \pi_2$.

(c) Delete redundant states.

Step 3. If $r[\alpha(a)] = 2$, $\alpha(a) = x$, then

(a) (X_0,η) is in t_a if $f_x(Y_1,Y_2) \sim X_0$, $\lambda \in F \cup F^S \cup F^I$ for all $(Y_1,\pi_1) \in$ SUBTREE(X_1,θ_1) and $(Y_2,\pi_2) \in$ SUBTREE(X_2,θ_2) where $(X_1,\theta_1) \in t_{a\cdot 1}$ and $(X_2,\theta_2) \in t_{a\cdot 2}$, then $\eta = \lambda + \pi_1 + \pi_2$.

(b) When $x \neq *$, add (X_0,η) to t_a if $f_*(Y_1,Y_2) \sim Z$, $0 \in F \cup F^I$ and $f_x(Z_1,Z_2) \sim X_0$, $\lambda \in F \cup F^S$, $(Z_1,\pi_1) \in$ SUBTREE(Z,θ), $(Z_2,\pi_2) \in$ SUBTREE(Start), then $\eta = \lambda + \pi_1 + \pi_2$.

(c) Delete redundant states.

Step 4. If (S,η) is in t_0, then $d(L(G_t),\alpha') = \eta$. exit

For all states in M_b, a table of minimal deleted trees is first computed by the two subroutine SUBTREE(Start) and SUBTREE(X,θ). (X_0,η) is generated from subroutine SUBTREE(Start), if $X_0 \xrightarrow[G_b]{*} \psi$, and ψ is the smallest subtree among all the possible derivations starting from non-terminal X_0. Similarly, (X_0,η) is generated from subroutine SUBTREE(X,θ), if $X_0 \xrightarrow[G_b]{*} \psi$, where X is the state of a frontier in ψ, and all the other nodes in ψ are labeled by terminal symbols. Furthermore, ψ has the least number of nodes among all the possible derivations. Steps (1), (2), and (3) are formulated under the assumption that: (i) the trees represented by the states of the immediate successor of the current node may be the subtree of the acutal derivation, (ii) there are one or more subtrees that may be deleted between the current node and the node right adjacent to it. For example, if rule $X_0 \rightarrow \begin{matrix} X \\ | \\ Y \end{matrix}$ is applied at t_a, but $(X,\theta) \in t_{a\cdot 1}$, then (Y,π) must be in SUBTREE(X,θ). The other words, let β be the tree represented by state X and αbe the tree represented by state Y, then β is a subtree of α, where nodes in α but not in β, are assumed to be deleted, the cost of deletion is $\pi-\theta$ which is the minimum among all the possible deletions. All the insertions are handled by transitions in F' automatically.

Example 3.4. Recognition of Hand-Written Character E

Suppose a casually-written character E, as shown in Figure 3.23(a), whose binary tree representation α'^* is shown in Figure 3.23(b), is to be recognized. The expanded tree automaton constructed from the tree grammar G_E in Appendix D-1 is as follows:

$\bar{M}_E = (\bar{V}, \bar{F}, \{S\})$

$\bar{V} = \{S, U_S, B, U_B, D, C, I\}$, $\bar{\Sigma} = \{a, b, d, h, \$, *\}$,

$\bar{F}$:

F: $f_{\$}(U_S, D) \sim S, 0$

$f_*(B) \sim U_S, 0$

$f_b(U_B, D) \sim B, 0$

$f_*(C) \sim U_B, 0$

$f_d \sim D, 0$

$f_b(D) \sim C, 0$

F^S: $f_y(U_B, D) \sim B, \sigma$ $\quad y \in \{a, d, h\}$

$f_y \sim D, \sigma$ $\quad y \in \{a, b, h\}$

$f_y(D) \sim C, \sigma$ $\quad y \in \{a, d, h\}$

F^D: $f(B) \sim U_S, 0$

$f(U_B, D) \sim B, \gamma$

$f(C) \sim U_B, 0$

$f \sim D, \gamma$

$f(D) \sim C, \gamma$

F^I: $f_y(B) \sim B, \delta$

$f_y(D) \sim D, \delta$

$f_y(C) \sim C, \delta$

$y \in \{a, b, d, h\}$

$f_*(I, B) \sim U_S, 0$

$f_*(I, C) \sim U_B, 0$

$f_b(I) \sim D, 0$

$f_y(I) \sim D, \sigma$ $\quad y \in \{a, d, h\}$

f_b (I,D) ∿ C, 0

f_y (I,D) ∿ C, σ y ε {a,d,h}

f_* (U_S,D) ∿ S, 0

f_* (S,I) ∿ S, 0

$f_\$$ (S,I) ∿ S, 0

f_* (U_S,I) ∿ U_S, 0

f_* (U_B,D) ∿ B, 0

f_* (B,I) ∿ B, 0

f_b (B,I) ∿ B, 0

f_y (B,I) ∿ B, σ y ε {a,d,h}

f_* (U_B,I) ∿ U_B, 0

f_* (D) ∿ C, 0

f_* (C,I) ∿ C, 0

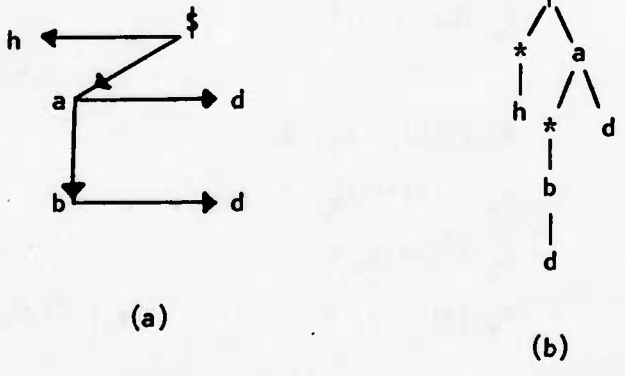
f_b (C,I) ∿ C, 0

f_y (C,I) ∿ C, σ y ε {a,d,h}

f_y (I) ∿ I, δ y ε {a,b,d,h,*}

f_y(I,I) ∿ I, δ y ε {a,b,d,h,*}

f_y ∿ I, δ y ε {a,b,d,h}



(a) (b)

Figure 3.23 Distorted Character E (a) and its Binary Tree Representation α'* (b).

Using the GECTA, the transition table of α'^* is illustrated in Figure 3.24 (a) and (b). From Figure 3.24 (b), since $(S, \sigma + \delta + \gamma)$ is in t_0, we have $d(L(G_E),\alpha') = \sigma + \delta + \gamma$.

3.4.3 An Illustrative Example on Character Recognition

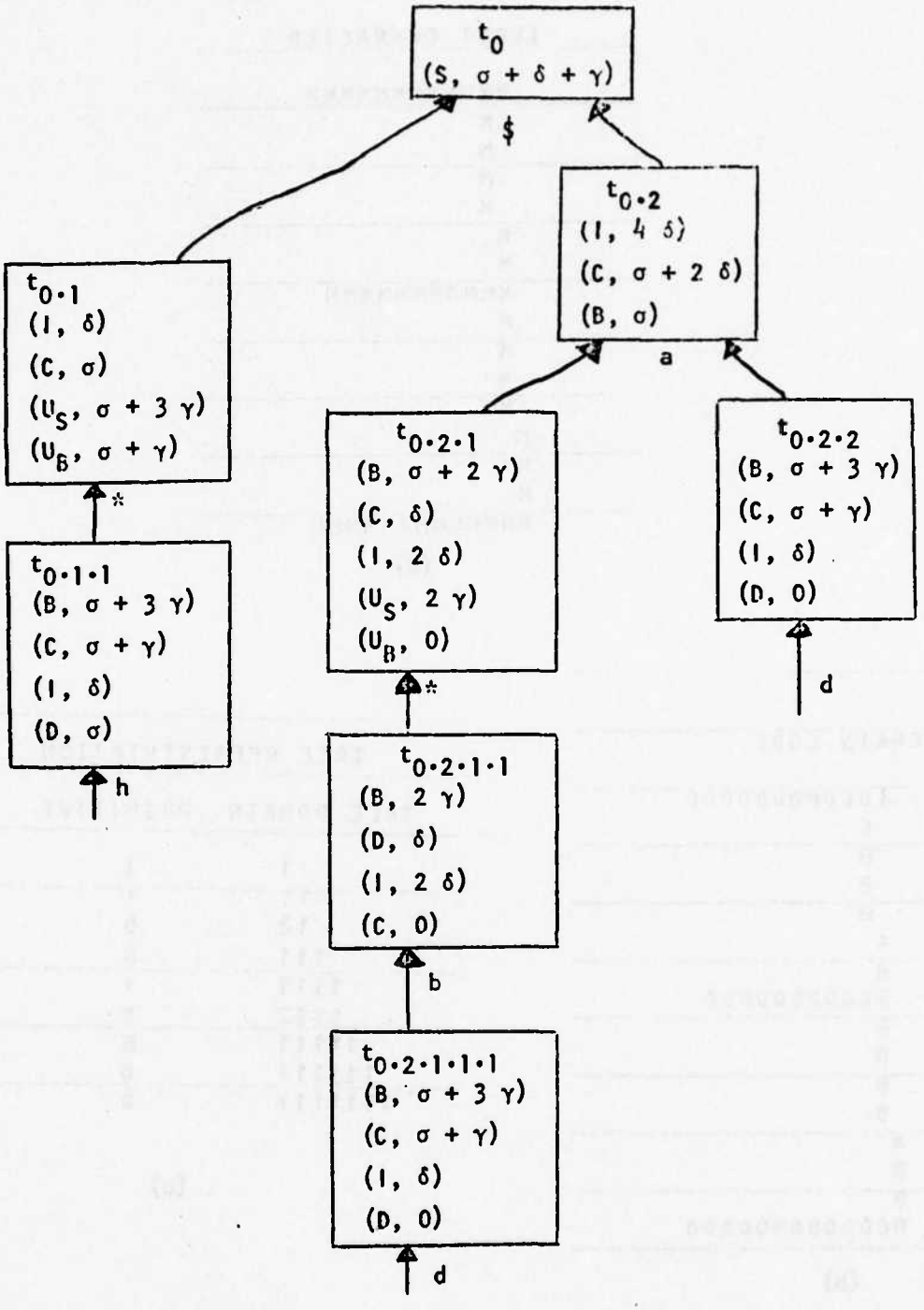
A classic example, the hand-printed chracter recognition problem, is given in this section to illustrate the operation of GECTA. Input characters are assumed to be digitized patterns in a 16 x 16 format. After chain coding [77] and primitive extracting, input patterns are transformed into their corresponding tree representations. Grammars for sample characters are given; hence, GECTA's are constructed. Input patterns are then analyzed by the GECTA's and classified based on the minimum-distance criterion.

An example of input pattern is shown in Figure 3.25 (a). Assume the leftmost of the top row to be the starting point. From the starting point, the input pattern is chain coded point by point. The successor point is coded as A, B,..., or H according to its relative position to the current node. The resulting chain code is shown in Figure 3.25 (b). In Figure 3.25 (b), the point labeled as "1" is the starting point. The majority code in a continuous line segment consisting of eight coded points is selected as the primitive of that line segment. Primitives selected by this method approximate the primitives defined in Figure 3.20. When a line is terminated or branched before it counts to eight points, the short line, if longer than two points, will be considered as a normal line segment; thus, a primitive. Otherwise, the line is neglected. The binary tree representation of pattern E is given in Figure 3.25 (c). The leftmost 1 of each strings in the column entitled

Y / π - θ / X	U_S	B	U_B	C	D
Start	4γ	4γ	2γ	2γ	γ
U_S	0	-	-	-	-
B	0	0	-	-	-
U_B	2γ	0	0	-	-
C	2γ	2γ	0	0	-
D	3γ	3γ	γ	γ	0

$(Y, \pi) \varepsilon$ SUBTREE(X, θ)

(a) Table of Minimal Deleted Trees



(b) Transition table of α'*

Figure 3.24 The acceptance of α'* by GECTA

INPUT CHARACTER

```
MMMMMMMMMMM
M
M
M
M
M
M
MMMMMMMMMMM
M
M
M
M
M
M
M
MMMMMMMMMMM
```

(a)

CHAIN CODE

```
IDEDDDDDDDDD
C
B
B
B
A
B
BCDDDDDDDDD
B
B
B
B
A
B
B
BCDDDDDDDDDD
```

(b)

TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	D	0
111	B	2
1111	*	1
1112	D	0
11111	B	1
111111	D	1
1111111	D	0

(c)

Figure 3.25 Input Format (a), Chain Code Result (b), and Binary Tree Representation (c).

as "tree domain" corresponds to the identify of U in Definition 3.1. The rest of the symbols in the string are added according to Definition 3.2. Therefore, 1, 11, 12, 111,---,etc. in Figure 3.25 (c) corresponds to tree domain 0, 1, 2, 1.1,---,etc. by Definition 3.2.

Five characters, A, C, D, E, H, are used in this experiment. The assumed sample patterns for each of the five characters and their tree representations are shown in Figure 3.26. Their respective tree grammars are given in Appendix D.1.

A total of 26 patterns, casually or meticulously printed characters, are tested. Assume that $\sigma = \gamma = \delta = 1$. The input patterns, their tree representations, as well as classification results and their distance from the assigned sample pattern are given in Appendix D.2. The entire recognition scheme is programmed in Fortran IV on a CDC 6500 computer. The cpu time used for chain coding and generating tree representation of each input is given under the title "time used for linking a tree." The actual recognition time is designated as "Time used for parsing." The average recognition time is 4.1 sec. per character.

These 26 test patterns are also processed by the (non-error-correcting) tree automata constructed respectively for grammars G_A, G_C, G_D, G_E, G_H given in Appendix D.1. Evidently, a pattern is acceptable only when its distance from a sample pattern is zero (see Appendix D.2). Therefore, only six out of the 26 test patterns are recognizable. The processing time is .012 sec. per character (average over the six recognizable patterns). Compared with their corresponding non-error-correcting schemes, error-correcting tree automata have the potential of recognizing highly noisy and distorted patterns. The trade-off is the processing efficiency.

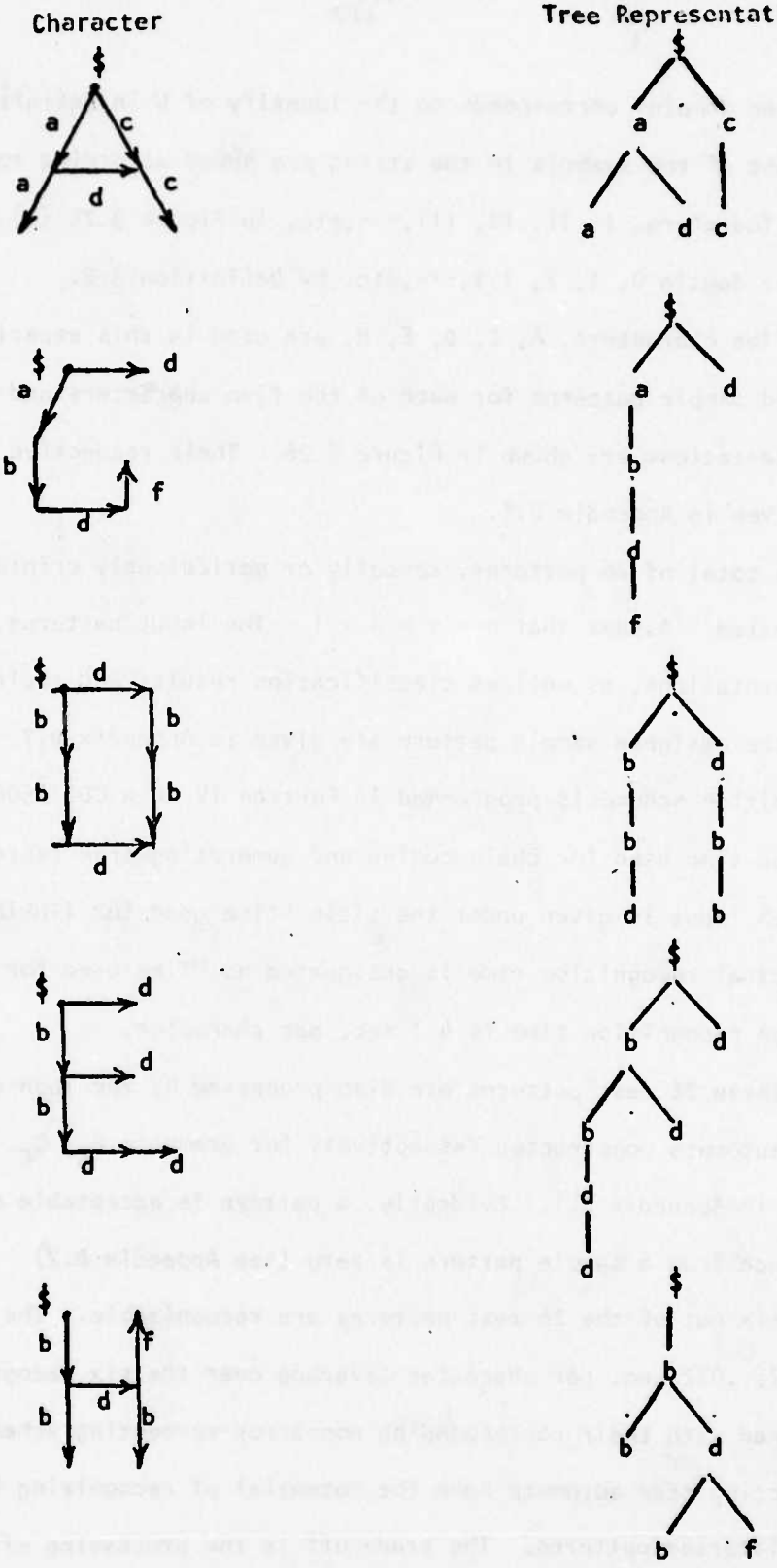


Figure 3.26 Sample patterns of character A, C, D, E, H

Using the techniques suggested in Section 2.4.3 can certainly reduce the processing time significantly.

When a large set of training samples, including noisy and distorted patterns, is available, a pattern grammar inferred from these samples will reliably represent its corresponding pattern structure. A conventional (non-error-correcting) syntax analyzer can thus perform recognition tasks satisfactorily. On the other hand, when the number of training samples is small, a pattern grammar inferred is usually very poor in describing the actual pattern structure. A conventional syntax analyzer designed according to the inferred grammar may often reject (noisy) patterns that should be accepted. In such a case, an error-correcting syntax analyzer, such as the proposed ECTA, could certainly recognize (or accept) these (noisy) patterns by using the minimum-distance criterion.

CHAPTER 4

CLUSTERING ANALYSIS FOR SYNTACTIC PATTERNS

4.1 Introduction

In statistical pattern recognition, a pattern is represented by a vector, called a feature vector. The similarity between two patterns can often be expressed by a distance, or more generally speaking, a metric in the feature space. Cluster analysis can be performed on a set of patterns on the basis of a selected similarity measure [3]. In syntactic or linguistic pattern recognition [5], a pattern is represented by a sentence in a language. The sentence could be a string, a tree, or a graph of pattern primitives and relations. The emphasis of such a representation is on the structure of patterns which is described by the syntax of a language. A similarity measure between two syntactic patterns must include the similarity of both their structures and primitives. In Chapter 2 and Chapter 3, we have proposed distance measures for strings and trees, which leads to the study of clustering analysis for syntactic patterns.

The conventional clustering methods, such as, the minimum spanning tree, the nearest (or K-nearest) neighbor classification rule and the method of clustering centers can be extended to syntactic patterns. We shall briefly describe the extension in Section 4.2. An illustrative example using a set of character patterns are presented in Section 4.3.

The studies described in Section 4.2 and Section 4.3 are mainly on the pattern-to-pattern basis. An input sentence (a pattern) is

compared with sentences in a formed cluster, one by one, or with the representation (cluster center) of the cluster. In Section 4.5 we shall use the distance measure between a sentence and a language proposed in Chapter 2. The proposed clustering procedure is combined with a grammatical inference procedure and an error-correcting parsing technique. The idea is to model the formed cluster by inferring a grammar, which implicitly characterize the structural identity of the cluster. The language generated by the grammar may be larger than the set consisting of the members of the cluster, and includes some possible similar patterns due to the recursive nature of grammar. Then the distance between an input sentence (a syntactic pattern) and a language (a group of syntactic pattern) is computed by using a ECP (error-correcting parser). The recognition is based on the nearest neighbor rule.

4.2. Sentence-to-Sentence Clustering Algorithms

4.2.1. A Nearest Neighbor Classification Rule

Suppose that C_1 and C_2 are two pattern sets, represented by sentences $X_1 = \{x_1^1, x_2^1, \ldots x_{n_1}^1\}$ and $X_2 = \{x_1^2, x_2^2, \ldots, x_{n_2}^2\}$ respectively. For an unknown syntactic pattern y, decide that y is in the same class as C_1 if

$$\min_j d(x_j^1, y) < \min_\ell d(x_\ell^2, y);$$

and y is in class C_2 if

$$\min_j d(x_j^1, y) > \min_\ell d(x_\ell^2, y). \qquad (4.1)$$

In order to determine $\min_j d(x_j^1, y)$, for some i, the distance between y and every element in the set X_1 have to be computed individually. The

string-to-string correction algorithm proposed in [26] yields exactly the distance between two strings defined in Definition 2.4. We shall briefly describe the algorithm in Appendix G.1.

The nearest neighbor classification rule can be easily extended to the K-nearest neighbor rule. Let $\tilde{X}_i = \{\tilde{x}_1^i, \tilde{x}_2^i, \ldots \tilde{x}_{n_i}^i\}$ be a reordered set of X_i such that $d(\tilde{x}_j^i, y) \leq d(\tilde{x}_\ell^i, y)$ iff $j < \ell$, for all $i \leq j$, $\ell \leq n_i$, then

$$\text{decide } y \in \begin{matrix} C_1 \\ C_2 \end{matrix} \text{ if } \sum_{j=1}^{K} \frac{1}{K} d(\tilde{x}_j^1, y) \lesseqgtr \sum_{j=1}^{K} \frac{1}{K} d(\tilde{x}_j^2, y) \tag{4.2}$$

We shall describe a clustering procedure in which, the classification of an input pattern is based on the nearest (or K-nearest) neighbor rule.

Algorithm 4.1

Input: A set of samples $X = \{x_1, x_2, \ldots, x_n\}$ and a design parameter, or threshold, t.

output: The partition of X into m clusters, $C_1, C_2, \ldots, C_m$.

Method:

Step 1. Assign x_1 to C_1, j=1, m=1.

Step 2. Increase j by one. if $D = \min_\ell d(x_\ell^i, x_j)$ is the minimum, $1 \leq i \leq m$, and

(i) $D \leq t$, then assign x_j to C_i

(ii) $D > t$, then initiate a new cluster for x_j, and increase m by one.

Step 3. Repeat Step 2 until all the elements of X have been put in a cluster.

Note that, in Algorithm 4.1, a design parameter is required. A commonly used clustering procedure is to construct a minimum spanning tree. Each node on the minimum spanning tree represents an element in the sample set X. Then partition the tree. Actually, when the distances between all of the pairs, $d(x_i,x_j)$, x_i, $x_j \in X$, are available, the algorithm for constructing minimum spanning tree is the same as that where X is a set of feature vectors in the statistical pattern recognition [94]. The algorithm is given in Appendix G.2.

4.2.2. The Cluster Center Techniques

Let us define a β-metric for a sentence x_j^i in cluster C_i as follow,

$$\beta_j^i = \frac{1}{n_i} \sum_{\ell=1}^{n_i} d(x_j^i, x_\ell^i) \qquad (4.3)$$

Then, x_j^i is the cluster center of C_i, if $\beta_j^i = \min_{\ell} \{\beta_\ell^i | 1 < \ell < n_i\}$, x_j^i is also called the representation of C_i, denoted A_i.

The following clustering algorithm is given in [94].

Algorithm 4.2

Input: A sample set $X = \{x_1, x_2, \ldots x_n\}$.

Output: The partition of X into m cluster

Method:

Step 1. Let m elementsof X, chosen at random, be the "representation" of the m clusters. Let them be called $A_1, A_2, \ldots, A_m$.

Step 2. For all i, $x_i \in X$ is assigned to cluster j, iff $d(A_j, x_i)$ is minimum.

Step 3. For all j, a new mean A_j is computed. A_j is the new representation of cluster j.

Step 4. If no A_j has changed, stop. Otherwise, go to Step 2.

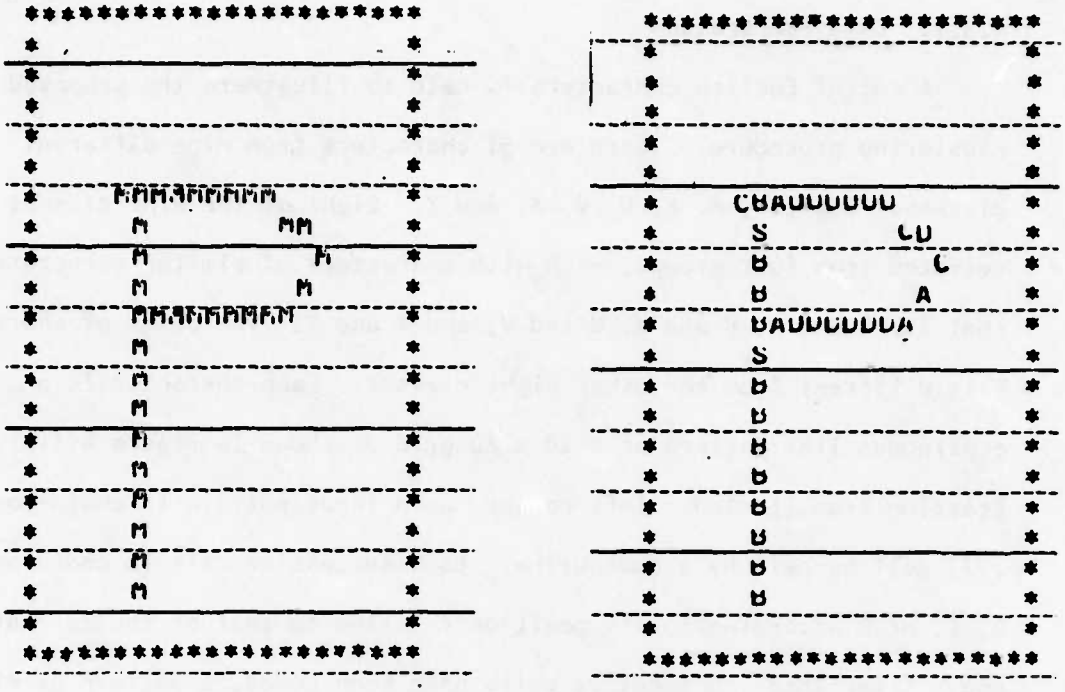
4.3. An Illustrative Example

4.3.1. Data Preparation

A set of English characters is used to illustrate the proposed clustering procedure. There are 51 characters from nine different classes: D, F, H, K. P, U. V, X, and Y. Eight of the nine classes are selected from four groups, each with characters of similar structure; that is, D and P, H and K, U and V, and X and Y. The class of character F is different from the other eight classes. Each character is a continuous line pattern on a 20 x 20 grid as shown in Figure 4.1(a). Starting from its lower left corner, each input pattern is chain-coded [77] cell by cell by a subroutine. Each successive cell is coded as A, B, C, or D according to its position relative to that of the current one. After three consecutive cells have been coded, a pattern primitive of this line segment is extracted.

(a) Primitive and Subpattern Extraction

Four pattern primitives which are line segments with four different orientations, $/$ a, $|$ b, $\backslash$ c, $-$ d, are selected. For example, ABA, CCA, or ADD are reduced to primitive a, b, or d, respectively. A primitive extraction subroutine PRIMITIVE is constructed and several typical results from the subroutine are shown in Figure 4.2. In the meantime, the chain-code subroutine searches for singular points of the input pattern for segmentation purposes. Coordinates of the singular points at two ends of a line segment (called a branch) are then recorded. For example, the chain-coded result for the character P shown in Figure 4.1(a) is given in Figure 4.1(b), in which the symbol "S" represents singular points. From Figure 4.1(b), the pattern has three branches, which,



(a) (b)

BRANCH1	START(	19	6)	TERM(	11	6)
BRANCH2	START(	10	6)	TERM(	7	6)
BRANCH3	START(	6	7)	TERM(	10	8)

(c)

Figure 4.1 The primitive extraction of a character P

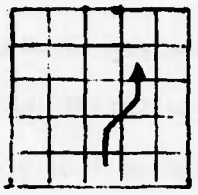
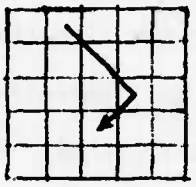
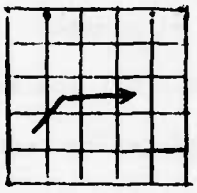
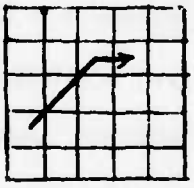
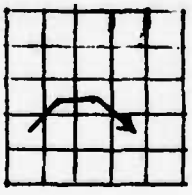
pattern	chain code	primitive
	BAB	b
	CCA	b
	ADD	d
	AAD	a
	ADC	d

Figure 4.2 The primitive selection of a line segment

together with the coordinates of their end points, are shown in Figure 4.1(c). Each branch is a subpattern, and the three branches consist of primitive strings bbb (Branch 1), b (Branch 2), and dddbdd (Branch 3), respectively.

(b) String Representation

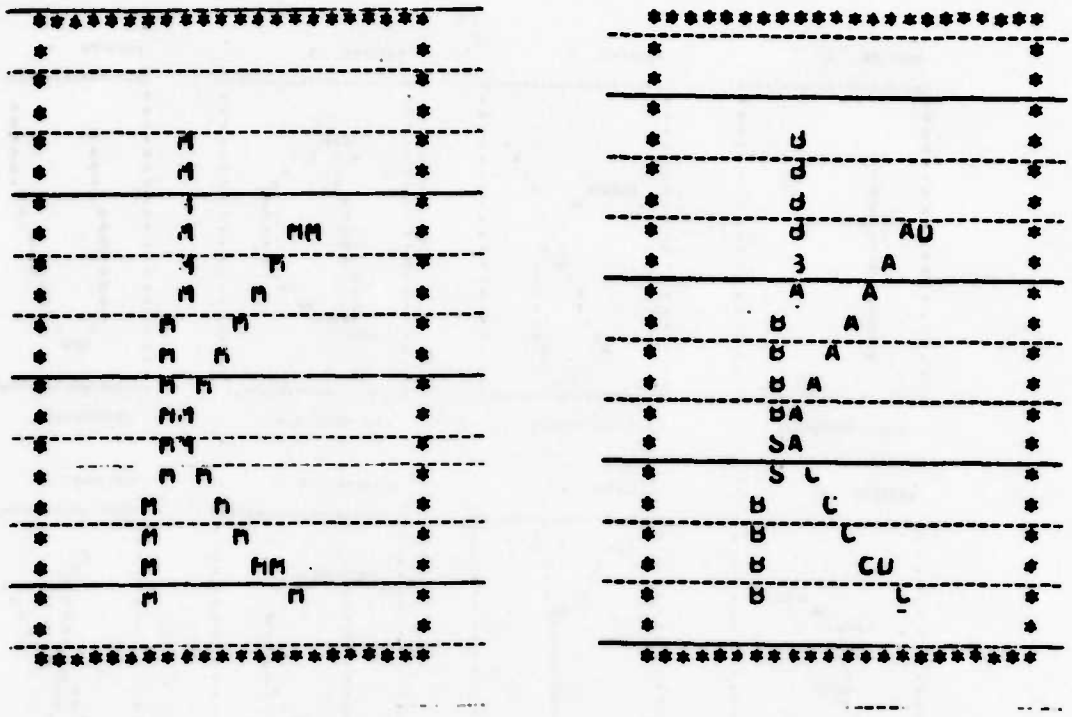
From the recorded coordinates of starting and terminating points (tail and head) of each branch, the subroutine CONCATE determines concatenation relations between branches. Following Shaw's PDL [95], three concatenation relations, +, x, *, and the parentheses (and) are used. However, * is used here primarily for the situation of a "self loop"; that is, a branch of which the head and the tail coincide. in such a case, the notation (B)* is used where B is the branch. The priority of the three concatenation relations follows the order of *, x, and then +. Consequently, redundant parentheses are eliminated. For the character P in Figure 4.1(a), the concatenations of branches are expressed as BRANCH 1 + (BRANCH 2 + BRANCH 3)*. The final string representation in terms of pattern primitives is bbb + (b + dddbdd)*. Similiarly, the character K in Figure 4.3 is represented by BRANCH 1 + BRANCH 2 x BRANCH 3 x BRANCH 4, and finally by a string of primitives b + bbbxaaxbc.

The 51 chain-coded patterns with their pattern numbers which represent input sequence are shown in Figure 4.4. The string representations extracted from subroutines, CHAIN-CODE, PRIMITIVE and CONCATE, are also listed in Table 4.1.

4.3.2. Experiments

(a) The Distance

The proposed distance measure is performed on the linguistic representations of pattern (sentences), rather than on the pattern



BRANCH1	START(	19	6)	TERM(	15	7)
BRANCH2	START(	13	7)	TERM(	4	8)
BRANCH3	START(	13	8)	TERM(	7	15)
BRANCH4	START(	14	8)	TERM(	19	14)

Figure 4.3 The primitive extraction of a character K

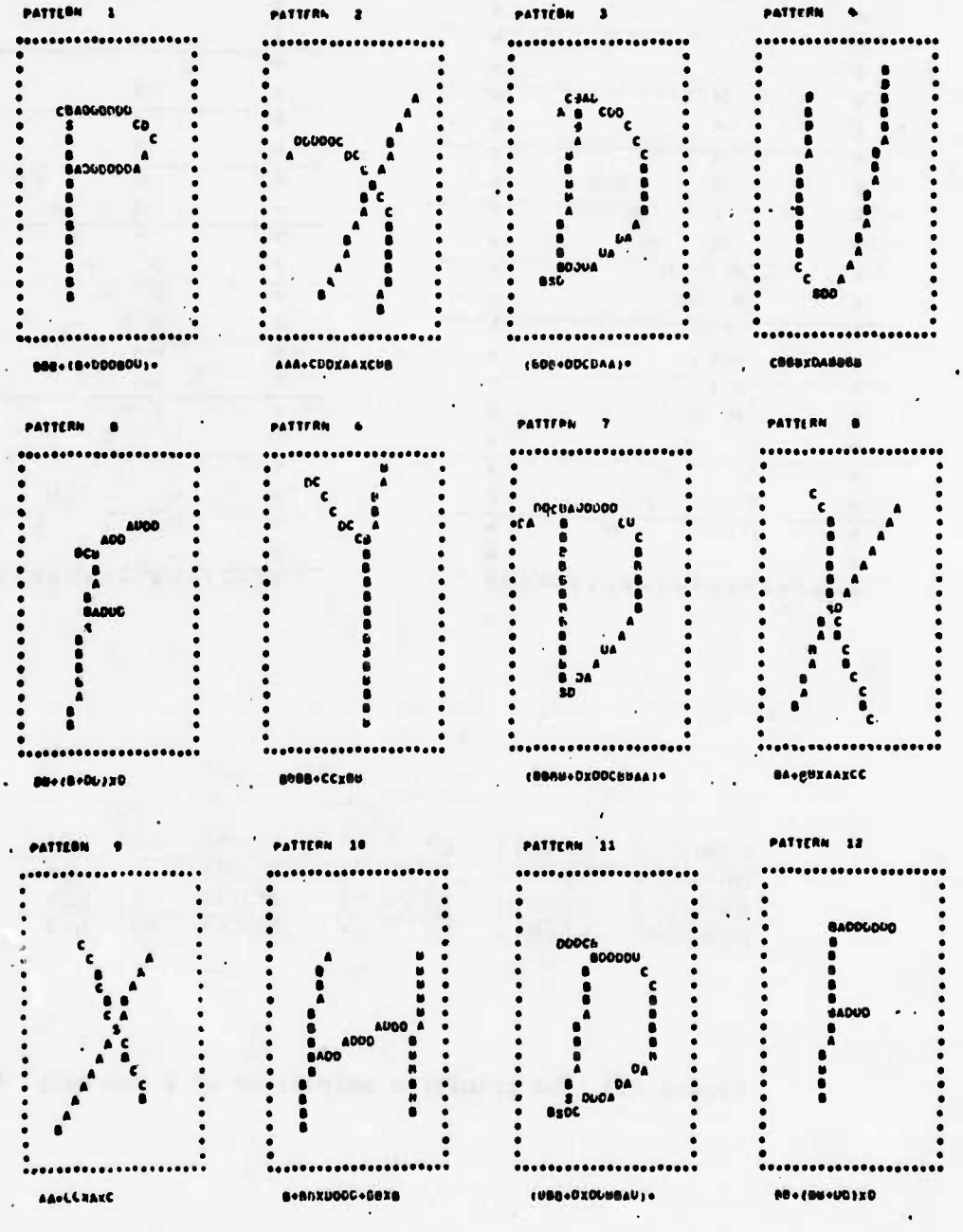


Figure 4.4 The 51 chain-coded character patterns

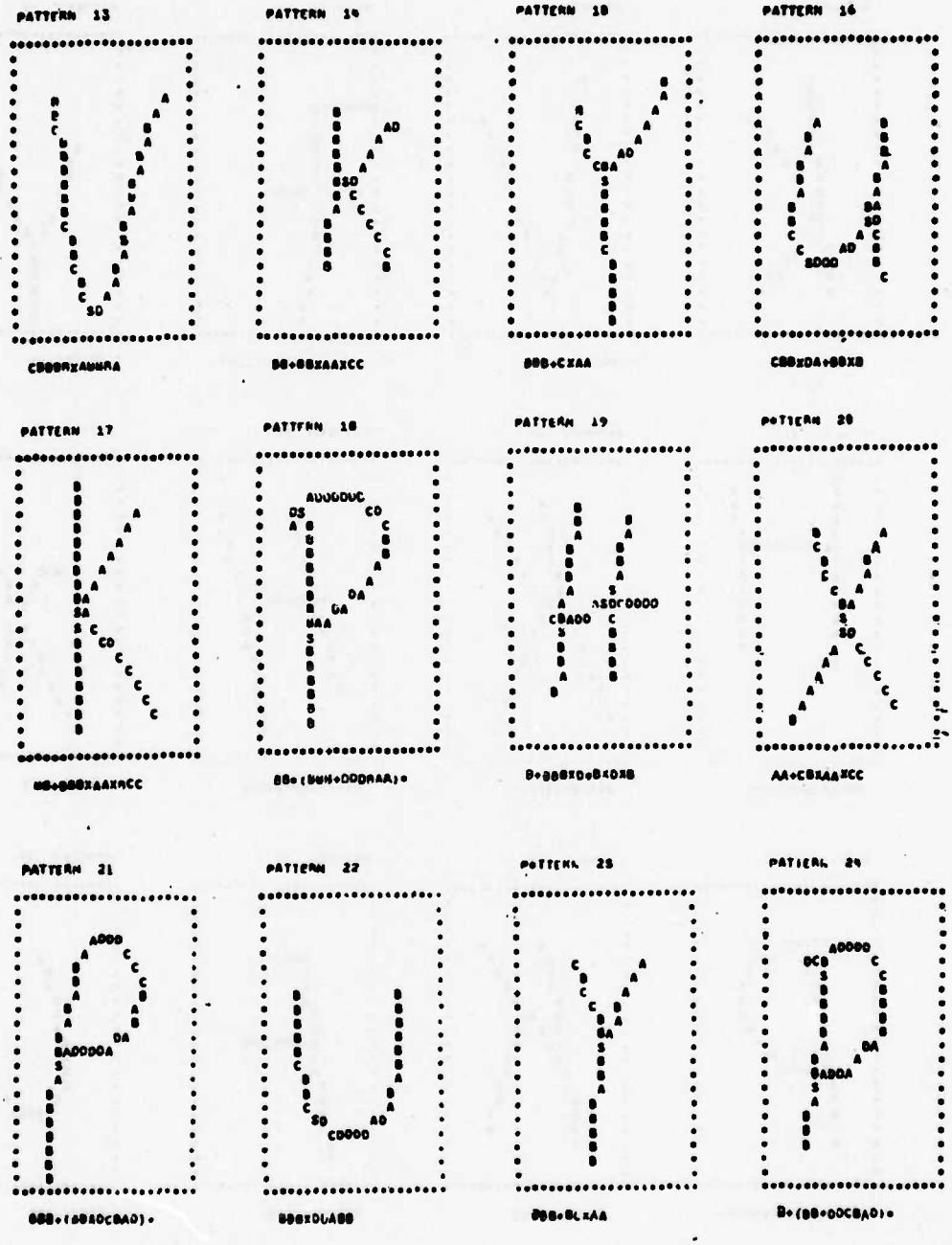


Figure 4.4 continued

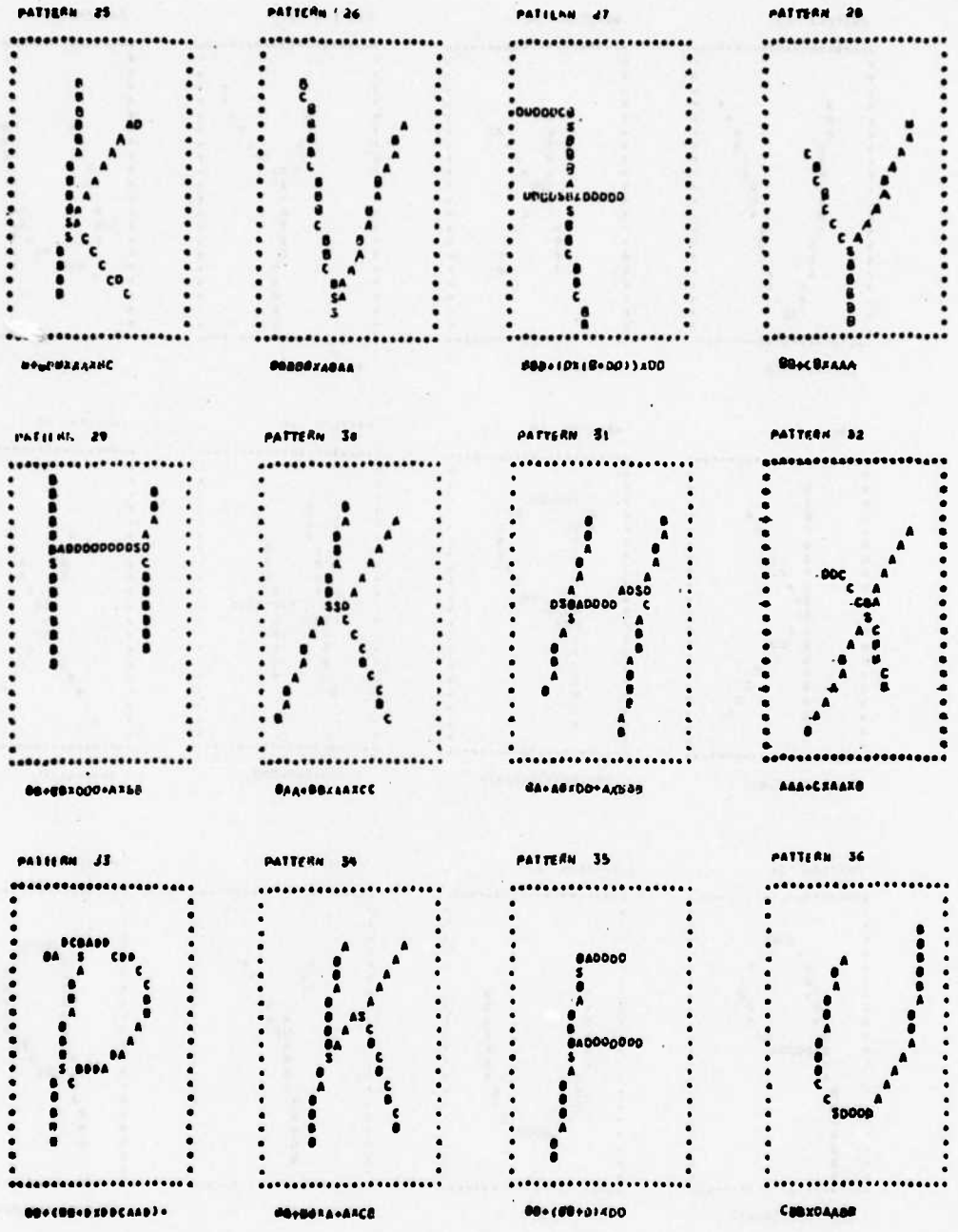


Figure 4.4 continued

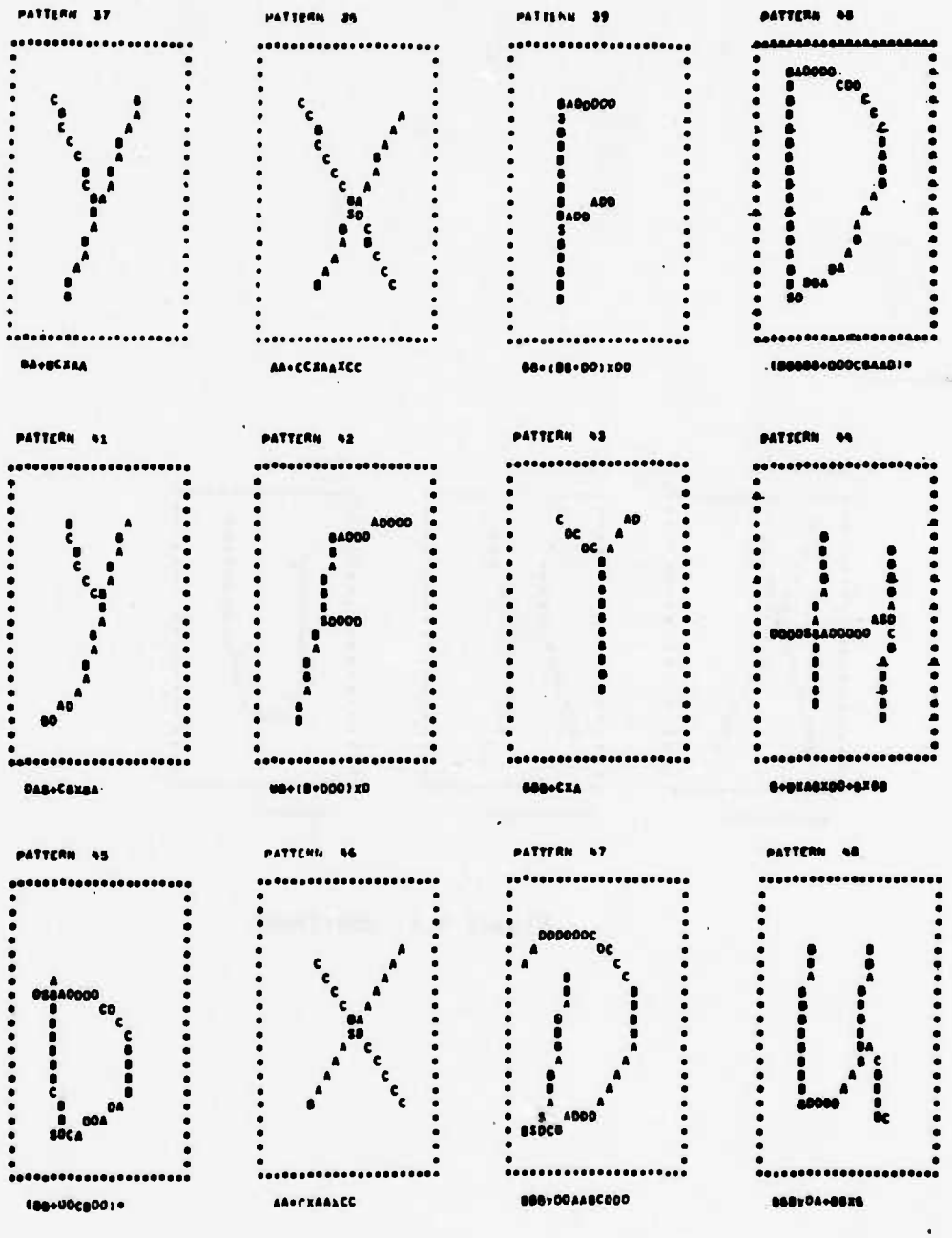


Figure 4.4 continued

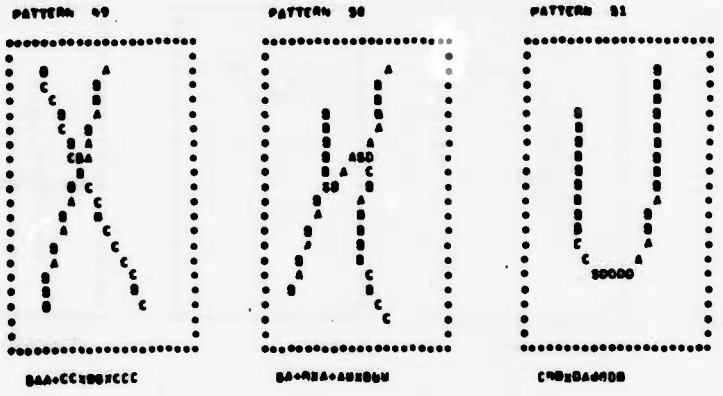


Figure 4.4 continued

Pattern No.	String Representation	Pattern No.	String Representation
1	bbb+(b+dddbdd)*	27	bbb+(dx(b+dd))xdd
2	aaa+cddxaaxcbb	28	bb+cbxaaa
3	(bbb+ddcbaa)*	29	bb+bbxddd+axbb
4	cbbbxdabbbb	30	baa+bbxaaxcc
5	bb+(b+dd)xd	31	ba+abxdd+axbbb
6	bbbb+ccxbb	32	aa+ccxaxc
7	(bbb+dxddcbbaa)*	33	bb+(bb+dxddcaad)*
8	ba+bbxaaxcc	34	bb+bbxa+axcb
9	aaa+cxaaxb	35	bb+(bb+d)xdd
10	b+bbxdddd+bbxb	36	cbbxdaabb
11	(bbb+dxddbbad)*	37	ba+bcxaa
12	bb+(bb+dd)xd	38	aa+ccxaaxcc
13	cbbbxabbba	39	bb+(bb+dd)xdd
14	bb+bbxaaxcc	40	(bbbbb+dddcbaad)*
15	bbb+cxaa	41	dab+cbxba
16	cbbxda+bbxb	42	bb+(b+ddd)xd
17	bb+bbbxaaxbcc	43	bbb+cxa
18	bb+(bbb+dddbaa)*	44	b+dxabxdd+bxbb
19	b+bbbxd+bxdxb	45	(bb+ddcbdd)*
20	aa+cbxaaxcc	46	aa+cxaaxcc
21	bbb+(bbadcbad)*	47	bbb+ddaabcddd
22	bbbxddabb	48	bbbxda+bbxb
23	bbb+bcxaa	49	baa+ccxbbxccc
24	b+(bb+ddcbad)*	50	ba+bxa+abxbbb
25	b+bbbxaaxbc	51	cbbxdabbbb
26	bbbbbxabaa		

Table 4.1 The 51 character patterns and their representations

themselves. Consequently, whether the model yields a good measurement is a matter of choosing representations. Figure 4.5 illustrates a couple of examples. In Figure 4.5(a) the distance between the two K's is smaller than a K and an X. Similarly, in Figure 4.5(b) the distance between a U and a distorted U, 'U', is still smaller than the distorted U and a H.*

(b) A Minimum Spanning Tree

Using algorithm in Appendix G.2 the minimum spanning tree for the 51 characters is constructed and shown in Figure 4.6. The true clusters are circled on the tree.

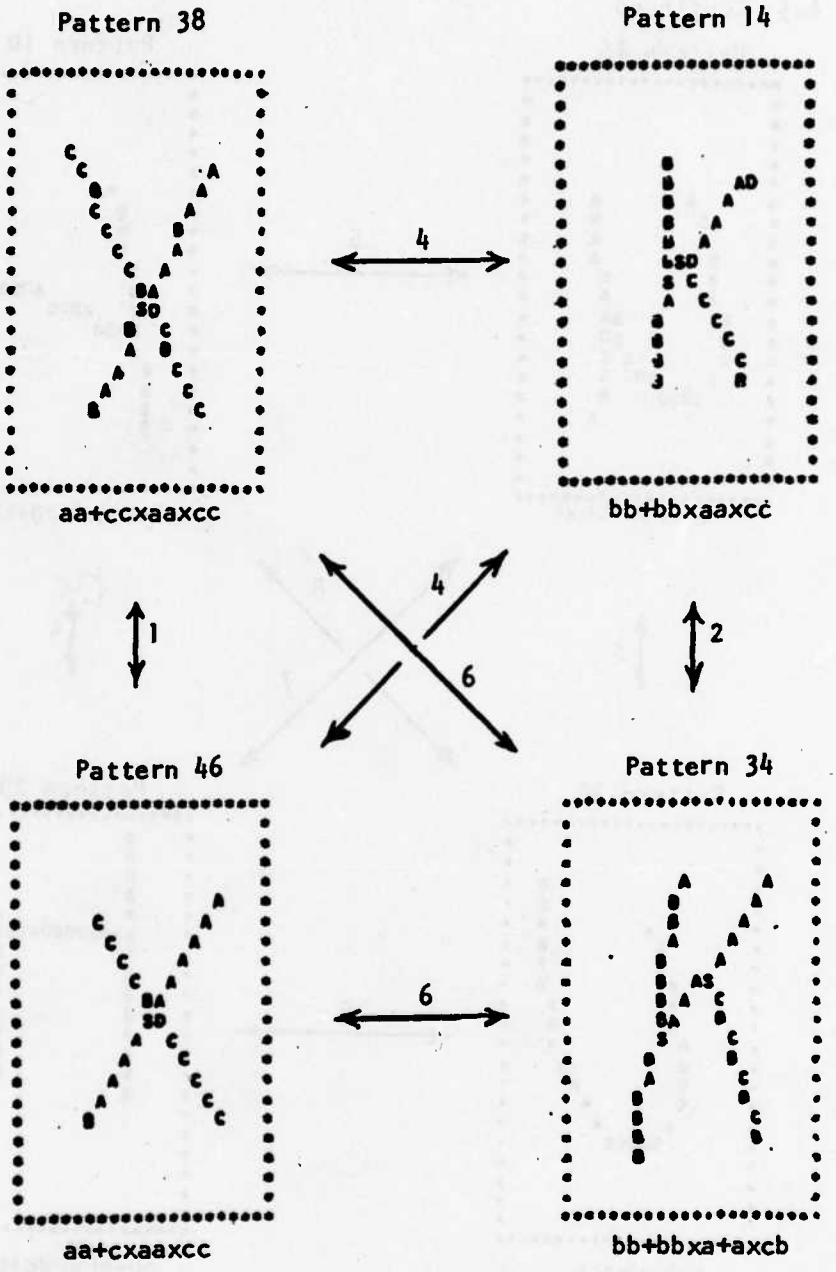
(c) The Clustered Results Using Algorithm 4.1

The results of clustering based on K-nearest neighbor classification rule are given in Figure 4.7, Figure 4.8 and Figure 4.9 when K=1 and t=6, K=3 and t=6, K=3 and t=6.5 respectively, where t is the preset threshold. The case that K=1 is the same as that using the nearest neighbor rule.

(d) The Clustering Results by Computing Cluster Centers.

The clustering procedure given in Algorithm 4.2 does not require a preset parameter. Let the algorithm be initiated by choosing the first nine input patterns; patterns 1 to 9 as the representations of the 9 classes. The procedure becomes stable after three iterations. The results of all the iterations are given in Table 4.2. The final clusters are shown in Figure 4.10. The representation of each formed cluster is marked with a square.

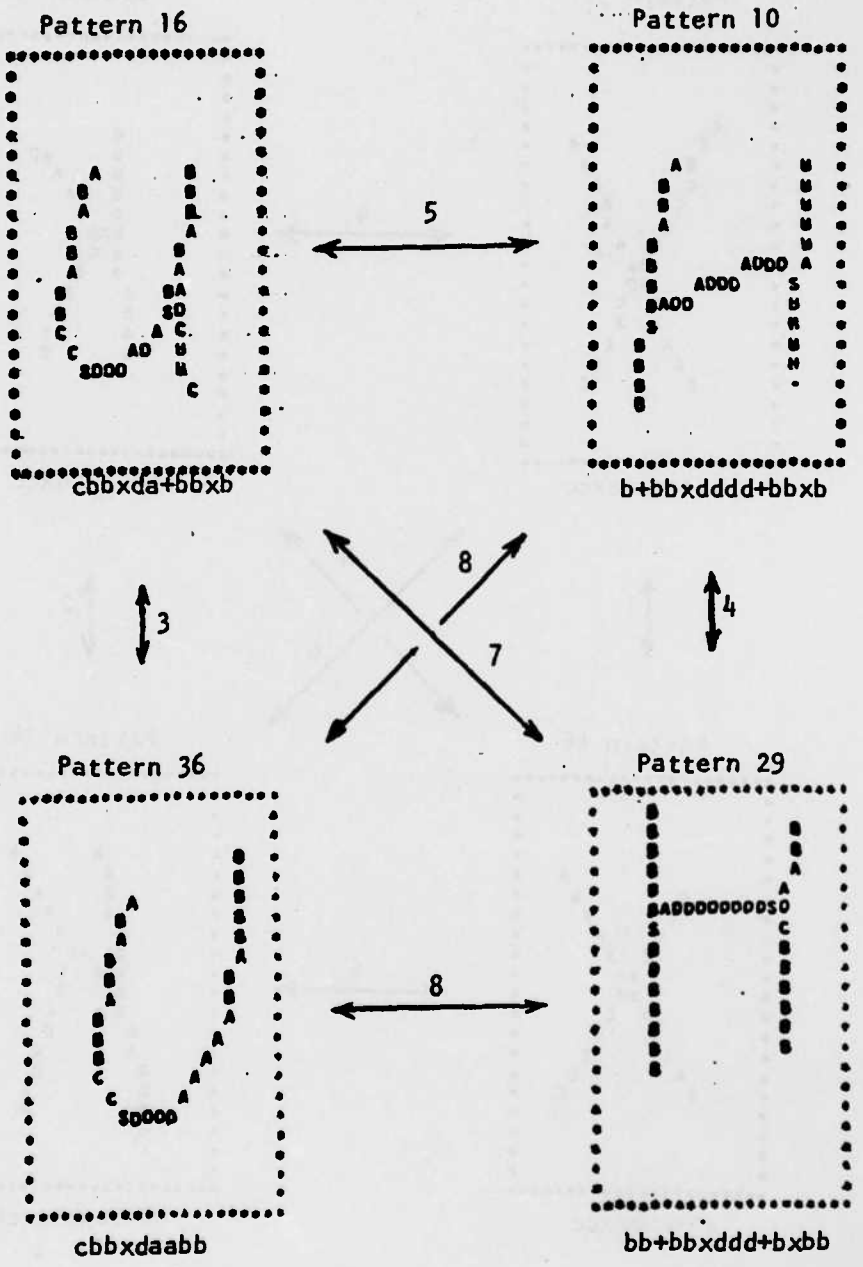
*In this section, all the experiments use unweighted distance. However, we exclude the possibility that a primitive namely, a, b, c, d, can not substitute a relation symbol, namely, +, x, *, (,), and vice versa.



(a)

Figure 4.5 The distances between similar and dissimilar patterns.

Figure 4.5 Continued



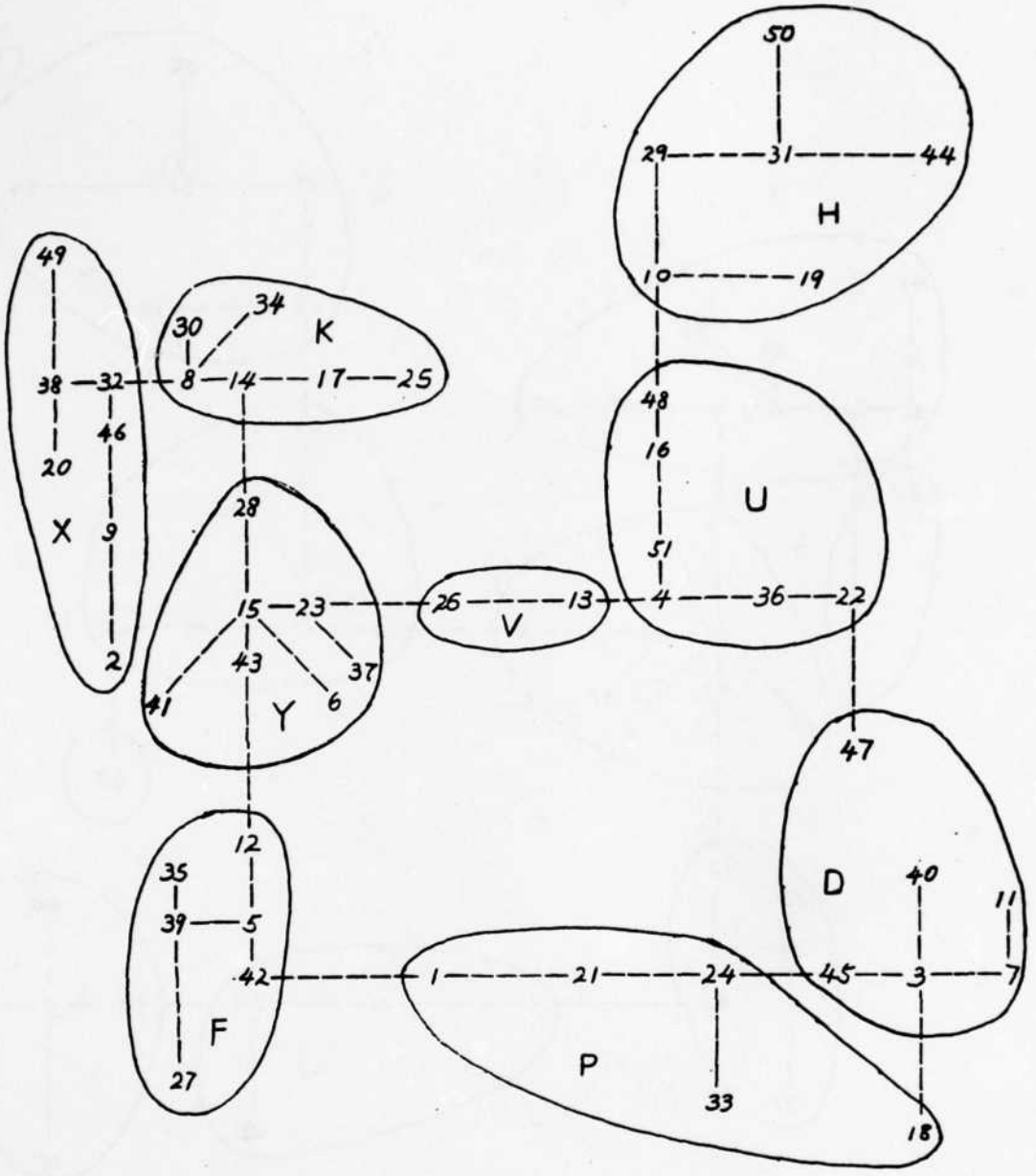


Figure 4.6 The minimum spanning tree of the 51 character patterns

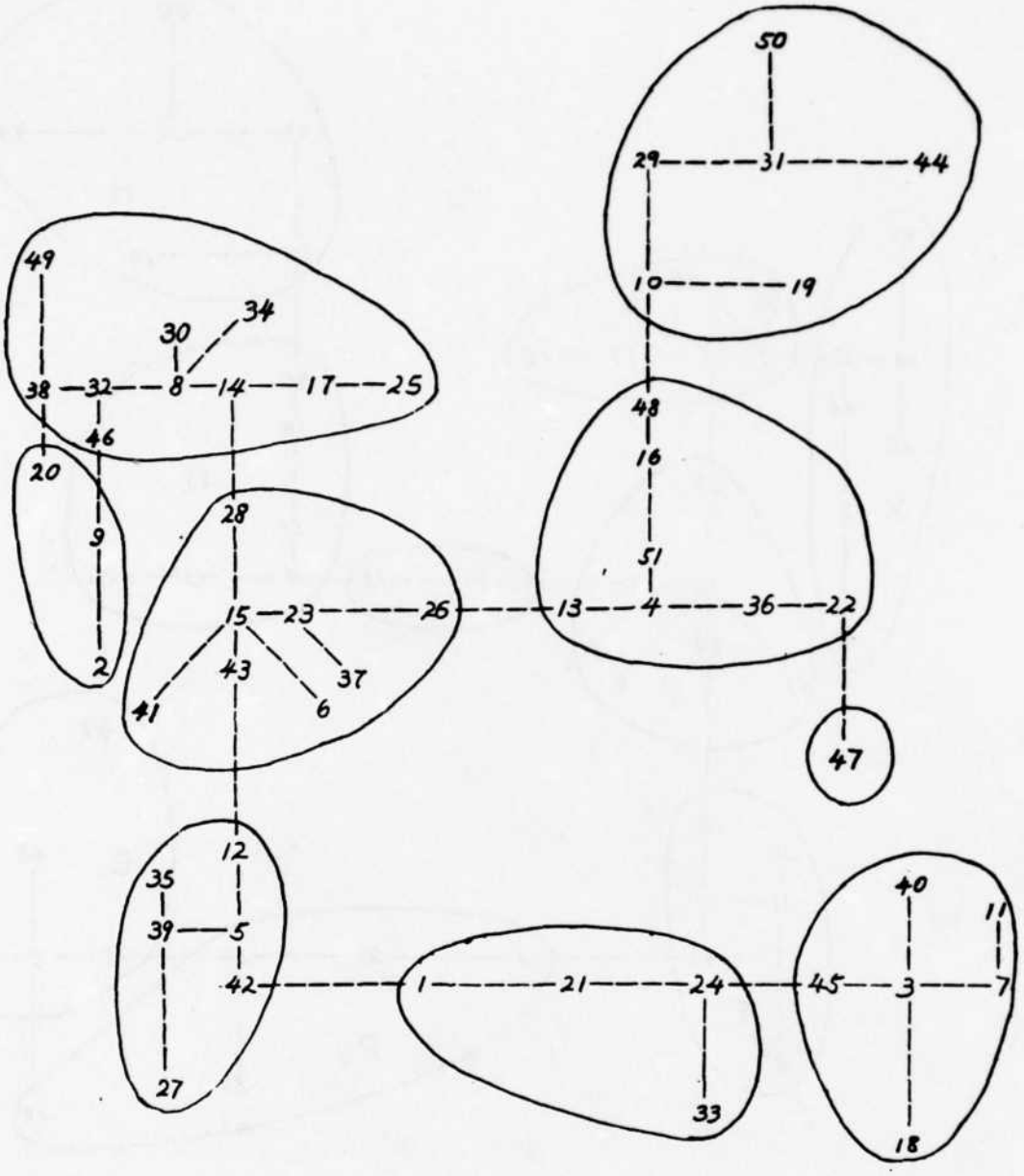


Figure 4.7 The result of using K-nearest neighbor recognition rule - K=1, t=6

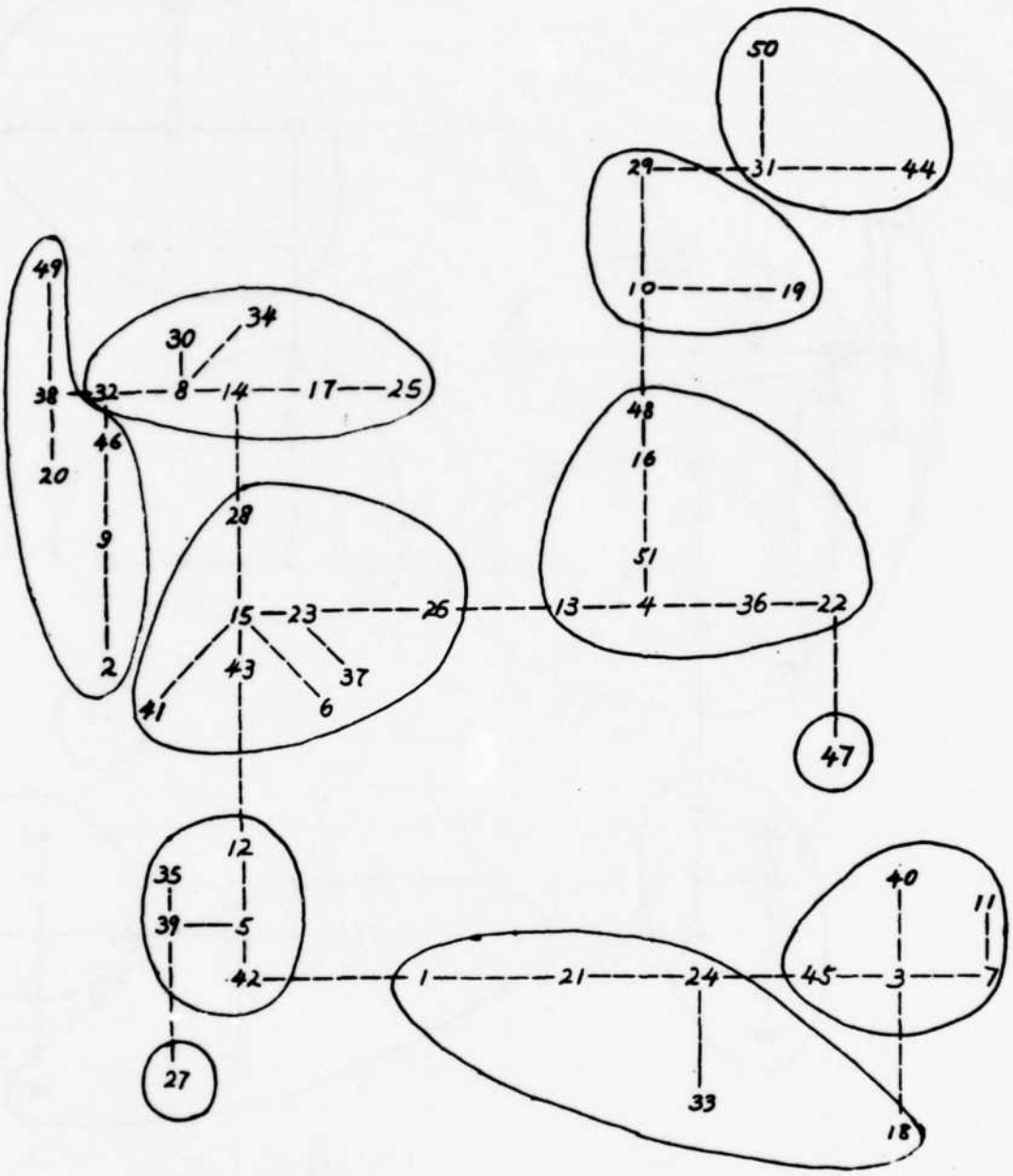


Figure 4.8 The result of using K-nearest neighbor recognition rule - K=3, t=6

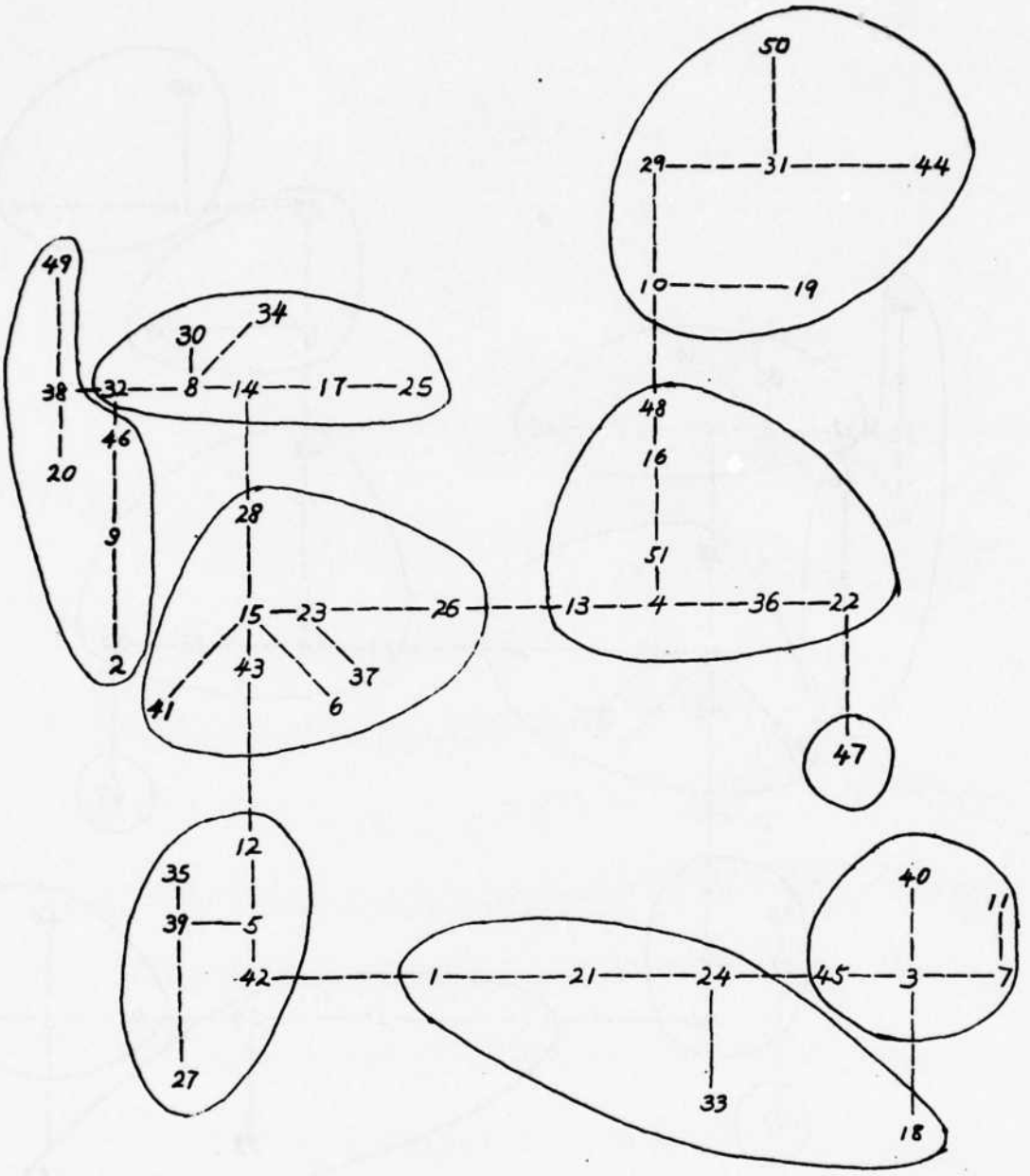


Figure 4.9 The result of using K-nearest neighbor recognition rule - K=3, t=6.5

cluster	1	2	3	4	5	6	7	8	9
representation	1	2	3	4	5	6	7	8	9
	21		18	10	12	15	11	14	20
	33		24	13	27	19		17	
			40	16	35	23		25	
			45	22	39	26		28	
				29	42	41		30	
				31		43		32	
				36				34	
				44				37	
				47				38	
				48				46	
				51				49	
								50	

(a) the first iteration

cluster	1	2	3	4	5	6	7	8	9
representation	21	2	3	22	39	15	11	8	20
	1	9	7	4	5	6		14	38
	24		18	10	12	23		17	46
	33		40	13	27	26		25	49
			45	16	35	28		30	
				19	42	37		32	
				29		41		34	
				31		43		50	
				36					
				44					
				47					
				48					
				51					

(b) the second iteration

Table 4.2 The three iteration results using Algorithm 4.2

Table 4.2 Continued

cluster	1	2	3	4	5	6	7	8	9
representation	24	9	3	22	39	15	11	8	38
	1	2	7	4	5	6		14	20
	21	46	18	10	12	23		17	49
	33		40	13	27	26		25	
			45	16	35	28		30	
				19	42	37		32	
				29		41		34	
				31		43		50	
				36					
				44					
				47					
				48					
				51					

(c) the third iteration

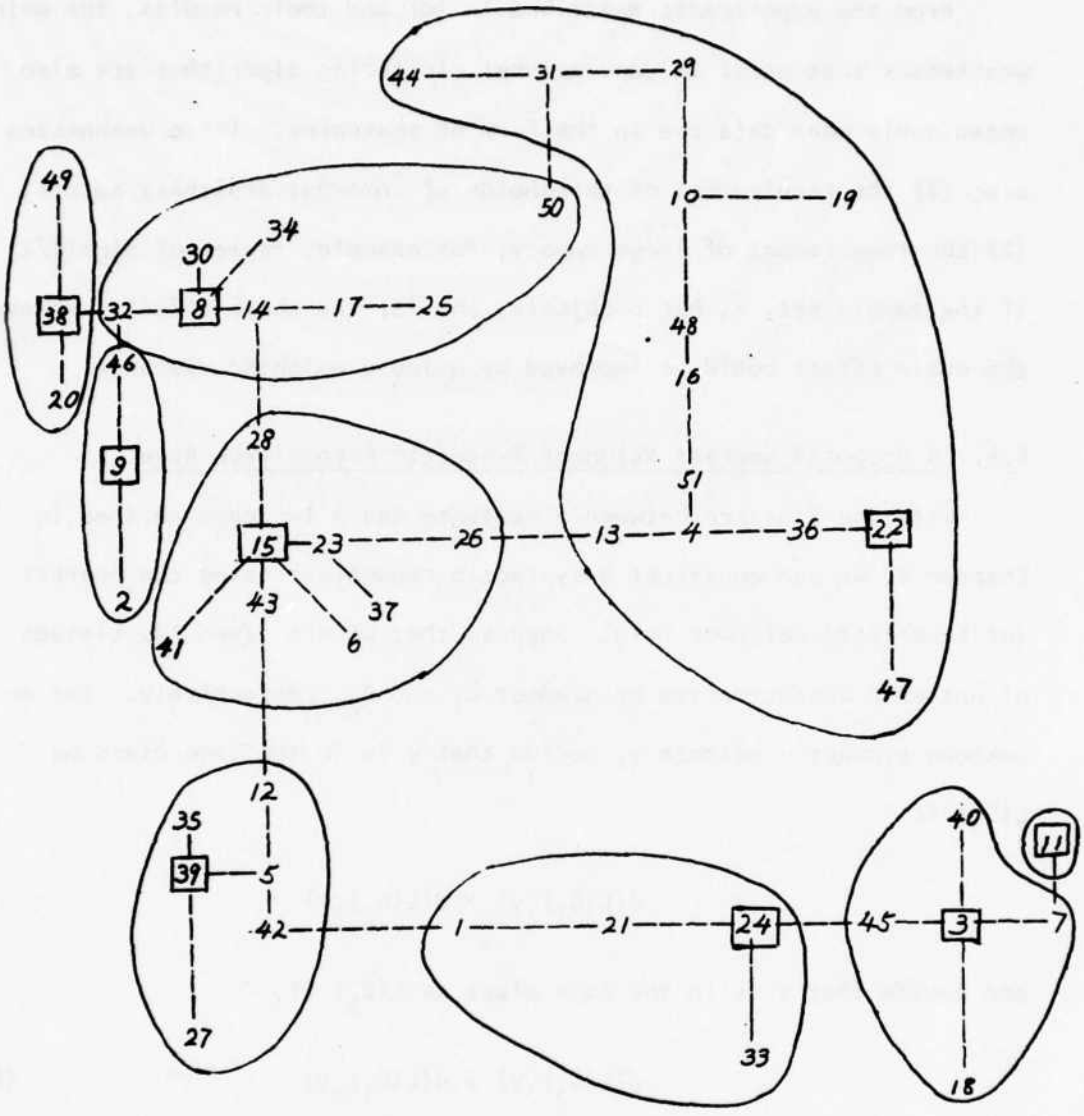


Figure 4.10 The result of classification according to clustering centers

(e) Remark

From the experiments described in (d) and their results, the main weaknesses that exist in conventional clustering algorithms are also unavoidable when data are in the form of sentences. These weaknesses are; (1) the requirement of thresholds of somewhat arbitrary nature, (2) the requirement of large memory, for example, tables of $n(n-1)/2$ if the sample set, X, has n objects, and (3) the chain effect. However, the chain effect could be improved by using a weighted distance.

4.4. A Proposed Nearest Neighbor Syntactic Recognition Rule

With the distance between a sentence and a language defined in Chapter 2, we can construct a syntactic recognizer using the nearest (or K-nearest) neighbor rule. Suppose that we are given two classes of patterns characterized by grammar G_1 and G_2, respectively. For an unknown syntactic pattern y, decide that y is in the same class as $L(G_1)$ if

$$d(L(G_1),y) > d(L(G_2),y)$$

and decide that y is in the same class as $L(G_2)$ if,

$$d(L(G_2),y) > d(L(G_1),y) \tag{4.4}$$

The distance $d(L(G_1),y)$ can be determined by a minimum-distance ECP constructed for G_1. Consequently, a grammatical inference procedure is required to infer a grammar for each class of pattern samples. Since the parser also gives the structural description of y, the syntactic recognizer gives both the classification and description of y as its output. We shall summarize the procedure in the following algorithm.

Algorithm 4.3

Input: m sets of syntactic pattern samples

$X_1 = \{x_1^1, x_2^1, \ldots, x_{n_1}^1\}, \ldots, X_m = \{x_1^m, x_2^m, \ldots, x_{n_m}^m\}$

and a pattern y with unknown classification.

Output: The classification and structural description of y.

Method:

Step 1. Infer m grammars $G_1, G_2, \ldots, G_m$, from $X_1, X_2, \ldots, X_m$, respectively.

Step 2. Construct minimum-distance ECP's, $E_1, E_2, \ldots, E_m$ for $G_1, G_2, \ldots, G_m$, respectively.

Step 3. Calculate $d(L(G_k), y)$ for all $i = 1, \ldots, m$. Determine ℓ such that

$$d(L(G_\ell), y) = \min_k d(L(G_k), y)$$

y is then classified as class ℓ. In the meantime, the structural description of x can be obtained from E_ℓ.

4.5. A Clustering Procedure for Syntactic Patterns

4.5.1. The Algorithm

Using the distance defined in Chapter 2 as a similarity measure between a syntactic pattern and a set of syntactic patterns, we can perform a cluster analysis to syntactic patterns. The procedure again involves error-correcting parsing and grammatical inference. In contrast to the nearest neighbor rule in Section 4.4 which uses a supervised inference procedure, the procedure described in this section is basically non-supervised. When the syntactic pattern samples are observed sequentially, a grammar can be easily infered for the sample observed at each stage of the clustering procedure. We propose the following clustering procedure for syntactic patterns:

Algorithm 4.4

Input: A set of syntactic pattern samples $X = \{x_1, x_2, \ldots, x_n\}$ where x_i is a string of terminals or primitives. A threshold t.

Output: The assignment of x_i, $i = 1,\ldots,n$ to m clusters and the grammar $G^{(k)}$, $k = 1,\ldots,m$, characterizing each cluster.

Method:

Step 1. Input the first sample x_1, infer a grammar $G_1^{(1)}$ from x_1, $L(G_1^{(1)}) \supseteq \{x_1\}$.

Step 2. Construct an error-correcting parser $E_1^{(1)}$ for $G_1^{(1)}$

Step 3. Input the second sample x_2, use $E_1^{(1)}$ to determine whether or not x_2 is similar to x_1 by comparing the distance between $L(G_1^{(1)})$ and x_2, i.e., $d(x_2, L(G_1^{(1)}))$, with a threshold t.

(i) If $d(x_2, L(G_1^{(1)})) < t$, x_1 and x_2 are put into the same cluster (Cluster 1). Infer a grammar $G_2^{(1)}$ from $\{x_1, x_2\}$.

(ii) If $d(x_2, L(G_1^{(1)})) \geq t$, initiate a new cluster for x_2 (Cluster 2) and infer a new grammar $G_1^{(2)}$ from x_2. In this case, there are two clusters characterized by $G_1^{(1)}$ and $G_1^{(2)}$, respectively.

Step 4. Repeat Step 2, construct error-correcting parsers for $G_2^{(1)}$ or $G_1^{(2)}$ depending upon $d(x_2, L(G_1^{(1)})) < t$ or $d(x_2, L(G_1^{(1)})) \geq t$, respectively.

Step 5. Repeat Step 3 for a new sample. Until all the pattern samples are observed, we have m clusters characterized by $G_{n_1}^{(1)}$, $G_{n_2}^{(2)}, \ldots, G_{n_m}^{(m)}$, respectively.

The parsers (non-error-correcting) constructed according to $G_{n_1}^{(1)}, G_{n_2}^{(2)}, \ldots, G_{n_m}^{(m)}$ could then form a syntactic recognizer directly for the m-class recognition problem.

The threshold t is a design parameter. It can be determined from a set of pattern samples with known classifications. For example, if we know that the sample x_i is from Class 1 characterized by $G^{(1)}$ and the sample x_j is from Class 2 characterized by $G^{(2)}$, then $t < d(x_i, x_j)$. Or, more generally speaking,

$$t < \text{Min } \{d(L(G^{(2)}), x_i) \ , \ d(L(G^{(1)}), x_j) \ \} \tag{4.5}$$

For m classes characterized by $G^{(1)}, G^{(2)}, \ldots, G^{(m)}$, respectively, we can choose

$$t < \min_{k,\ell} \{d(L(G^{(\ell)}), x^{(k)}) \ \}, \ k \neq \ell \tag{4.6}$$

where $x^{(k)}$ is a pattern sample known from Class k and $L(G^{(\ell)})$ is the grammar characterizing Class ℓ ($\ell \neq k$). If the above required information is not available, an appropriate value of t will have to be determined on an experimental basis until a certain stopping criterion is satisfied (for example, with a known number of clusters).

4.5.2 An Experiment

Let the same set of pattern samples used in Section 4.3 be tested by Algorithm 4.4. The subroutines of finding string representations for the 51 character patterns are still used here. In addition, we will have subroutines for grammatical inference and error-correcting parsing.

(1) Grammatical Inference (Step 1, 3, and 5 in Algorithm 4.4)

By comparing its distances to existing clusters with a threshold t, an input pattern sample is assigned to an existing cluster or a new cluster. In either case, a grammar is first inferred for the single input sample. The inferred grammar is then merged into the grammar (by merging their productions) characterizing the assigned cluster. The subroutine REDUCE combines productions of the two grammars, removes identical productions, and unifies nonterminals. To use a simple inference procedure, input samples are assumed to be generated by finite-state grammars. However, non-self-embedding context-free productions are used to describe concatenation of branches. Each branch is described by finite-state productions. For the character P shown in Figure 4.1, the inferred grammar is given in Table 4.3.

(2) Error-Correcting Parser (ECP - Step 2 and 4)

The error-correcting parser used in this example is the MDECP given in Algorithm 2.1 and 2.2. Certain realistic assumptions are made to reduce the number of error productions. For example, we do not allow a substitution error that occurred between a, b, c, or d and +, x, *, (,or). Also, the concatenation symbol + or x cannot be inserted at the end of a string.

A simplified flow chart for the complete experiment is given in Figure 4.11.

(3) Experimental Results

Following the clustering procedure described in Section 4.5, three experiments were performed: (i) using unweighted (Levenshtein) distance with threshold t = 3, (ii) using unweighted (Levenshtein) distance with

$G_1^{(0)} = (V_N, V_T, P, S_0)$

$V_N = \{S_0, BA, KA, BC, EA, EB\}$

$V_T = \{(, +, x, *,), b, d\}$

P: $S_0 \rightarrow BA + KA *$

$KA \rightarrow (BA + BC)$

$BA \rightarrow b\ BA$

$BA \rightarrow b$

$BC \rightarrow d\ BC$

$BC \rightarrow d\ EA$

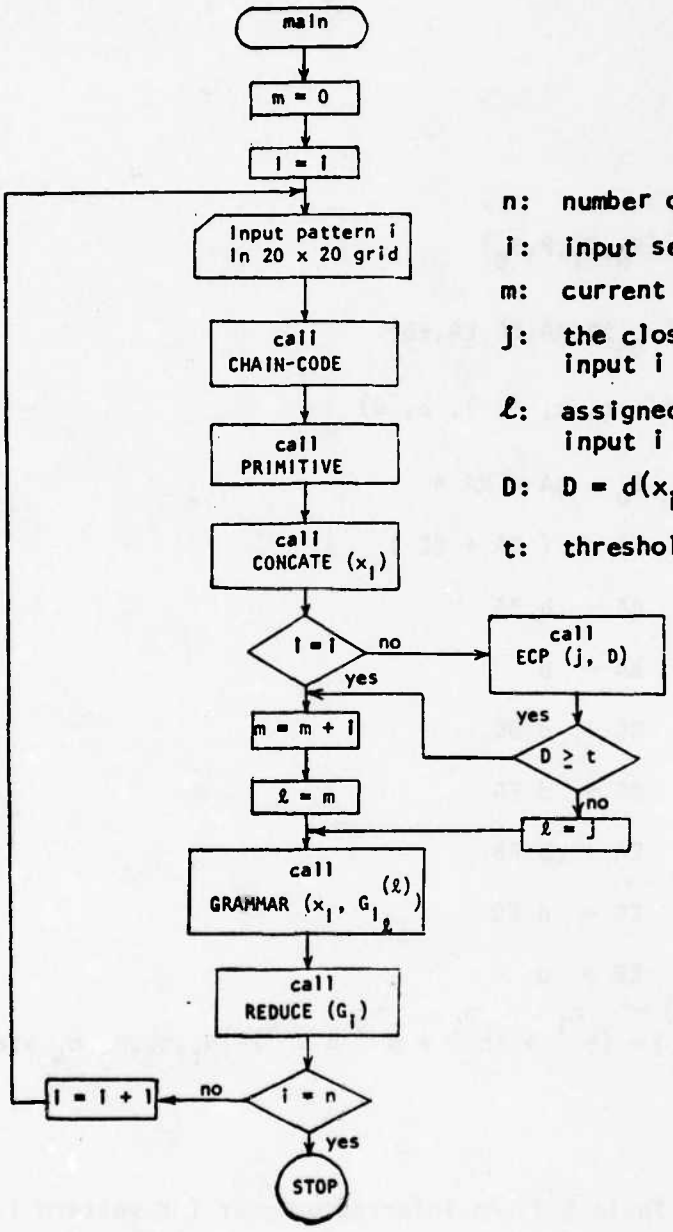
$EA \rightarrow b\ EB$

$EB \rightarrow d\ EB$

$EB \rightarrow d$

$L(G_1^{(0)}) = \{b^{n_1} + (b^{n_2} + d^{n_3}\ b\ d^{n_4})^* | n_1, n_2, n_3, n_4$ are positive integers$\}$.

Table 4.3 An inferred grammar for pattern 1.



Where $G_i = \{V_i, \Sigma_i, P_i, SS\}$ is the reduced grammar equivalent to $G_{i_1}^{(1)} \cup G_{i_2}^{(2)} \cup \ldots \cup G_{i_m}^{(m)}$.

Figure 4.11 Flow chart of the clustering Algorithm 4.4

threshold t = 4, and (iii) using weighted distance with threshold t = 3. The results of clustering analysis are listed in Table 4.4. In Table 4.4 pattern samples are grouped according to the clustering result of the experiment (iii); pattern numbers in the first column correspond to those listed in Figure 4.4. The number in the parentheses of the last column in Table 4.4 represents the weighted distance between the pattern and the most similar cluster; that is, the cluster to which the pattern is assigned.

For Experiment (i), there were 11 clusters formed. Pattern 18 was not assigned to the same cluster with other character P's, and Pattern 27 was not assigned to the same cluster with other F's. For Experiment (ii), t was increased from 3 to 4, only 7 clusters were formed. Both Pattern 18 and Pattern 27 were correctly clustered. However, all the X's and the K's were assigned to the same cluster. In both experiments, Patterns 13, 26 and 47 were not correctly clustered. For Experiment (iii), 10 clusters were formed; all the patterns except Pattern 47 were correctly clustered. The weights associated with each error productions used in the experiment are given in Table 4.5.

The final inferred grammar G_n, n = 51, from the third experiment, is listed in Table 4.6. The grammar is the union of the inferred grammars for each cluster; namely, $G^{(0)}$, $G^{(1)}$, $G^{(2)}$,---,$G^{(9)}$. Nonterminals S_0, S_1, S_2,---,S_9 in the grammar are the start symbols of $G^{(0)}$, $G^{(1)}$, $G^{(2)}$,---,$G^{(9)}$, respectively. In order to carry out the parsing for all clusters using a single parser, we merge all the productions originated from S_0, S_1,---,S_9 into a single grammar G_{51} by adding productions SS $\rightarrow S_0$, SS $\rightarrow S_1$,---,SS $\rightarrow S_9$ where SS is the start symbol of G_{51}.

Pattern No.	String Representation	True Character	Unweighted t=3	Unweighted t=4	Weighted t=3
1	bbb+(b+dddbdd)*	P	0	0	0(-)
18	bbb+(bbb+dddbaa)*	P	0	0	0(2.2)
21	bbb+(bbadcbad)*	P	8	0	0(3.0)
24	b+(bb+ddcbad)*	P	0	0	0(1.5)
33	bb+(bb+dxddcaad)*	P	0	0	0(1.5)
2	aaa+cddxaaxcbb	X	1	1	1(-)
9	aa+ccxaxc	X	1	1	1(1.5)
20	aa+cbxaaxcc	X	1	1	1(0.6)
32	aaa+cxaaxb	X	1	1	1(1.0)
38	aa+ccxaaxcc	X	1	1	1(0.)
46	aa+cxaaxcc	X	1	1	1(0.)
49	baa+ccxbbxcc	X	1	1	1(2.7)
3	(bbb+ddcbaa)*	D	2	2	2(-)
7	(bbb+dxddcbbaa)*	D	2	2	2(1.0)
11	(bbb+dxddbbad)*	D	2	2	2(1.5)
40	(bbbbb+dddcbaad)*	D	2	2	2(1.0)
45	(bb+ddcbdd)*	D	2	2	2(0.9)
4	cbbbxdabbb	U	3	3	3(-)
16	cbbxda+bbxb	U	3	3	3(2.0)
22	bbbxddabb	U	3	3	3(0.5)
36	cbbxdaabb	U	3	3	3(0.)
48	bbbxda+bbxb	U	3	3	3(0.5)
51	cbbxdabbbb	U	3	3	3(0.)
5	bb+(b+dd)xd	F	4	4	4(-)
12	bb+(bb+dd)xd	F	4	4	4(0.)
27	bbb+(dx(b+dd))xdd	F	9	4	4(2.0)
35	bb+(bb+d)xdd	F	4	4	4(0.)
39	bb+(bb+dd)xdd	F	4	4	4(0.)
42	bb+(b+ddd)xd	F	4	4	4(0.)
6	bbbb+ccxbb	Y	5	5	5(-)
15	bbb+cxaa	Y	5	5	5(2.0)
23	bbb+bcxaa	Y	5	5	5(0.9)
28	bb+cbxaaa	Y	5	5	5(1.0)
37	ba+bcxaa	Y	5	5	5(1.0)
41	dab+cbxba	Y	5	5	5(2.9)
43	bbb+cxa	Y	5	5	5(0.)
8	ba+bbxaaxcc	K	6	1	6(-)
14	bb+bbxaaxcc	K	6	1	6(0.9)
17	bb+bbbxaaxbcc	K	6	1	6(0.9)
25	b+bbbxaaxbc	K	6	1	6(0.)
30	baa+bbxaaxcc	K	6	1	6(0.)
34	bb+bbxa+axcb	K	6	1	6(1.6)
10	b+bbxdddd+bbxb	H	7	6	7(-)

Table 4.4 The result clusters of the 51 characters

Table 4.4 Continued

Pattern No.	String Representation	True Character	Unweighted t=3	Unweighted t=4	Weighted t=3
19	b+bbbxd+bxdxb	H	7	6	7(1.5)
29	bb+bbxddd+axbb	H	7	6	7(1.0)
31	ba+abxdd+axbbb	H	7	6	7(2.0)
44	b+dxabxdd+bxbb	H	7	6	7(3.0)
50	ba+bxa+abxbbb	H	7	6	7(3.0)
13	cbbbxabbba	V	3	3	8(-)
26	bbbbbxabaa	V	5	5	8(0.5)
47	bbbxddaabcddd	D	10	7	9(-)

Rule	Weight
$E_a \rightarrow a$	0.
$E_a \rightarrow b$	.9
$E_a \rightarrow c$	2.0
$E_a \rightarrow d$	1.0
$E_a \rightarrow aE_a$	0.
$E_a \rightarrow dE_a$	0.5
$E_a \rightarrow +E_a$	0.6
$E_a \rightarrow xE_a$	0.5
$E_a \rightarrow \lambda$	0.9
$E_b \rightarrow b$	0.
$E_b \rightarrow a$	1.0
$E_b \rightarrow c$	0.6
$E_b \rightarrow d$	1.5
$E_b \rightarrow aE_b$	1.0
$E_b \rightarrow bE_b$	0.
$E_b \rightarrow dE_b$	1.0
$E_b \rightarrow +E_b$	1.0
$E_b \rightarrow xE_b$	1.0
$E_b \rightarrow \lambda$	0.5
$E_c \rightarrow c$	0.
$E_c \rightarrow b$	1.0
$E_c \rightarrow bE_c$	0.9
$E_c \rightarrow cE_c$	0.
$E_c \rightarrow \lambda$	0.5
$E_d \rightarrow d$	0.
$E_d \rightarrow a$	1.2
$E_d \rightarrow b$	2.0

Table 4.5 New rules added to the original grammars

Table 4.5 Continued

Rule	Weight
$E_d \rightarrow c$	0.5
$E_d \rightarrow dE_d$	0.
$E_d \rightarrow xE_d$	1.0
$E_d \rightarrow \lambda$	2.1
$E_x \rightarrow x$	0.
$E_x \rightarrow dE_x$	1.0
$E_x \rightarrow xE_x$	0.5
$E_+ \rightarrow +$	0.
$E_+ \rightarrow aE_+$	1.0
$E_+ \rightarrow \lambda$	0.5
$E_* \rightarrow *$	0.
$E_) \rightarrow)$	0.
$E_) \rightarrow)E_)$	0.5
$E_(\rightarrow ($	0.
$E_(\rightarrow dE_($	0.5
$E_(\rightarrow xE_($	0.5
$E_(\rightarrow (E_($	0.5

$G = (V_N, V_T, P, SS)$, where

$V_N = \{XY \mid X \text{ and } Y \text{ are alphabets}\} \cup \{S_I \mid I = 0,1,\ldots,9\}$

$V_T = \{a, b, c, d, +, x, *, (,)\}$

P:

1	$SS \rightarrow S_0$	26	$S_3 \rightarrow KD$
2	$S_0 \rightarrow BA + KA\ *$	27	$KD \rightarrow BG \times BK$
3	$KA \rightarrow (\ BA + BC)$	28	$BK \rightarrow d\ EI$
4	$BA \rightarrow b\ BA$	29	$EI \rightarrow a\ BA$
5	$BA \rightarrow b$	30	$SS \rightarrow S_4$
6	$BC \rightarrow d\ BC$	31	$S_4 \rightarrow BA + KF$
7	$BC \rightarrow d\ EA$	32	$KE \rightarrow (\ BA + EB\ )$
8	$EA \rightarrow b\ EB$	33	$KF \rightarrow KE \times EB$
9	$EB \rightarrow d\ EB$	34	$SS \rightarrow S_5$
10	$EB \rightarrow d$	35	$S_5 \rightarrow BA + KG$
11	$SS \rightarrow S_1$	36	$KG \rightarrow BQ \times BA$
12	$S_1 \rightarrow BD + KB \times BG$	37	$BQ \rightarrow c\ BQ$
13	$KB \rightarrow BE \times BD$	38	$BQ \rightarrow c$
14	$BD \rightarrow a\ BD$	39	$S_2 \rightarrow KH\ *$
15	$BD \rightarrow a$	40	$KH \rightarrow (\ BA + EB \times BI\ )$
16	$BE \rightarrow c\ EB$	41	$EF \rightarrow b\ EF$
17	$BG \rightarrow c\ BA$	42	$SS \rightarrow S_6$
18	$SS \rightarrow S_2$	43	$S_6 \rightarrow EF + KI \times BQ$
19	$S_2 \rightarrow KC\ *$	44	$KI \rightarrow BA \times BD$
20	$KC \rightarrow (\ BA + BI\ )$	45	$S_1 \rightarrow BD + KJ \times BA$
21	$BI \rightarrow d\ BI$	46	$KJ \rightarrow BQ \times BD$
22	$BI \rightarrow d\ EE$	47	$SS \rightarrow S_7$
23	$EE \rightarrow c\ EF$	48	$S_7 \rightarrow BA + KK + KL$
24	$EF \rightarrow b\ BD$	49	$KK \rightarrow BA \times EB$
25	$SS \rightarrow S_3$	50	$KL \rightarrow BA \times BA$

Table 4.6 The grammar G_{51}

Table 4.6 Continued

51	$S_2 \rightarrow KM$ *	82	$S_3 \rightarrow KS$
52	KM → (BA + EB x BR)	83	KS → BA x BK
53	BR → d BR	84	BK → d BK
54	BR → d EG	85	$S_5 \rightarrow BA + KT$
55	EG → b EG	86	KT → BU x BD
56	EG → b EH	87	$S_0 \rightarrow BA + KU$ *
57	EH → a EB	88	KU → (BA + EJ)
58	$SS \rightarrow S_8$	89	EJ → d EJ
59	$S_8 \rightarrow KN$	90	$S_8 \rightarrow KW$
60	KN → BG x BS	91	KW → BA x BS
61	BS → a EF	92	$S_4 \rightarrow BA + KZ$
62	$S_6 \rightarrow BA + KI \times BQ$	93	KY → (EB x KE)
63	$S_5 \rightarrow BA + KJ$	94	KZ → KY x EB
64	$S_3 \rightarrow KO + KL$	95	$S_5 \rightarrow BA + MA$
65	KO → BG x BT	96	MA → BG x BD
66	BT → d BD	97	$S_7 \rightarrow BA + KK + MC$
67	$S_6 \rightarrow BA + KI \times BU$	98	MC → BD x BA
68	BU → b BQ	99	$S_7 \rightarrow EF + MD + MC$
69	$S_0 \rightarrow BA + KP$ *	100	MD → EI x EB
70	KP → (BA + BV)	101	$S_1 \rightarrow BD + MA \times BQ$
71	BV → d BV	102	$S_0 \rightarrow BA + ME$ *
72	BV → d EF	103	ME → (BA + EB x CB)
73	$S_7 \rightarrow BA + KK + KK \times BA$	104	CB → d CB
74	$S_1 \rightarrow BD + KJ \times BQ$	105	CB → d EL
75	$S_0 \rightarrow BA + KR$ *	106	EL → c EH
76	KR → (CA)	107	EH → a EH
77	CA → b CA	108	$S_6 \rightarrow BA + KI + MF$
78	CA → b EI	109	MF → BD x BG
79	EI → a EJ	110	EI → a EI
80	EJ → d EK	111	$S_5 \rightarrow EF + KT$
81	EK → c EG	112	$S_1 \rightarrow BD + KJ \times BQ$

Table 4.6 Continued

113	S_2 → KU *
114	S_5 → BK + MG
115	MG → BG x EF
116	S_4 → BA + MH
117	MH → KE x EB
118	S_5 → BA + MI
119	MI → BQ x BD
120	S_7 → BA + MJ x EB + MK
121	MJ → EB x EI
122	MK → BA x BA
123	S_2 → ML *
124	ML → (BA + CC)
125	CC → d CC
126	CC → d EN
127	EN → c EA
128	SS → S_9
129	S_9 → MM
130	MM → BA x CD
131	CD → d CD
132	CD → d EQ
133	EQ → a EQ
134	EQ → a ER
135	ER → b BE
136	S_3 → MN + KL
137	MN → BA x BT
138	S_1 → EF + KG x BQ
139	S_7 → EF + KI + MP
140	MP → EI x BA

4.6. Conclusions and Remarks

In Sections 4.4 and 4.5, we have demonstrated that, by using an error-correcting parser, the distance between a syntactic pattern and a group of syntactic patterns can be determined. Such a distance can be used for the nearest neighbor recognition and the cluster analysis for syntactic patterns. Essential parts in both applications include grammatical inference and error-correcting parsing. If the correct classifications of pattern samples are known, the proposed nearest neighbor syntactic recognition rule can be applied to determine the classification and structural description of an unknown pattern. Using Algorithm 2.3 to determine the average distance between a sentence and the K-nearest sentences in the language, the proposed rule for a single nearest neighbor can be easily extended to the K-nearest neighbors.

When the correct classifications of pattern samples are unknown, a non-supervised procedure must be used. In this case, the proposed clustering procedure can be applied. Using error-correcting parsers in cluster analysis, after the clustering result is obtained, we could only implement conventional non-error-correcting parsers for recognition. Furthermore, the grammars inferred could be in finite-state form, the construction of conventional parsers for finite-state grammars is straightforward, and the parsing procedure is, in general, deterministic and efficient. The proposed clustering procedure can certainly be extended to syntactic patterns represented by trees since tree grammar inference procedures [93] and error-correcting tree automata have already been developed. When a general error-correcting parser is used, the computer time required for clustering analysis could be slow. (For

example, using a CDC 6500 computer with FORTRAN IV programming language, the average computer time for analyzing each pattern in Experiment (iii) is 37 sec.) Nevertheless, this requirement may not be critical since a clustering algorithm is used primarily for pattern analysis rather than recognition. Besides, inference procedures for special grammars [74], (such as finite-state and precedence grammars) can always be employed to speed up the analysis.

The grammar inferred for each cluster often generates not only the sentences (syntactic patterns) already in the cluster, but also sentences with similar structures. For example, the grammar $G_1^{(0)}$ in Table 4.3 is an inferred grammar for Pattern 1, bbb+(b+dddbdd)*. However, $L(G_1^{(0)}) = \{b^{n_1} + (b^{n_2} + d^{n_3} b\, d^{n_4})^* | n_1, n_2, n_3$ and n_4 are positive integers$\}$ which represents character P of different sizes (Cluster 0). Consequently, the unweighted distance between Pattern 18 (a character P) and Cluster 0 is 2 although the unweighted distance between Pattern 18 and Pattern 1 is 4. For this reason, the clustering procedure proposed in Section 4.5 appears to be more effective and flexible than that of computing the distance between an input pattern sample and a set of prototype or reference patterns on a sentence-by-sentence basis.

With syntactic or linguistic representations of data being more and more common in pattern recognition, speech and language analysis, and database systems [98-101], nearest neighbor recognition rule, and clustering procedures for syntactic patterns should find their applications in these areas.

CHAPTER 5

A SYNTACTIC APPROACH TO TEXTURE ANALYSIS

5.1 Introduction

Research on texture analysis--modeling, synthesis, classification, and discrimination--has received increasing attention in recent years [78]. Texture is a term for the quality of a surface. The feature that dominates a texture scene is the repetitive or quasi-repetitive pattern. Texture information is valuable in scene segmentation, especially in those cases in which the contrast between the object to be observed and the background is poor. A survey of research in this area can be found in Zucker [79]. Applications of texture analysis include terrain classification [80-81], radiographic image interpretation [82,83], microscopic cell image analysis [84-86], materials inspection [87], and many others.

Most of the previous research has concentrated on the statistical approach [80-90]. In this approach, statistical properties are calculated from a set of local measurements taken from the pattern. Weszka, Dyer, and Rosenfeld [81] give a comparative study of several frequently used features for texture classification.

An alternative approach to the statistical one for texture analysis is the structural approach [78]. A texture is considered to be defined by subpatterns which occur repeatedly according to a set of well-defined placement rules within the overall pattern. Furthermore, the subpattern itself is made of structured elements. Compared with the statistical

approach, the structural approach appears to be easier to interpret.

Zucker [79] proposed the idea of texture modeling in terms of an ideal texture and its transformations. According to Zucker, an ideal texture is a deep, unobservable, highly-structured perfect pattern in which local primitives (fundamental building blocks) are extended into a global structure such as regular tesselation. He believes that transformation rules can be defined from a representation of an ideal texture to that of a natural texture. In our work, we shall propose a tree grammar which defines such rules.

Carlucci [91] has formulated a system called "texture language" for the description of some simple repetitive subpatterns such as polygons or open polygonals. Texture patterns are treated as graphs with the basic elements in the texture language representation being lines and vertices. The structure of a subpattern is then represented as a tree. From a practical point of view, Carlucci's system may encounter difficulties during preprocessing, such as difficulty in the extraction of lines and vertices in a texture region.

Ehrich and Foith [42], propose a tree language approach for the description of the structure of waveforms called "relational trees." They believe that information about textures in an image can be obtained by sequential analysis of individual scan lines. The change of gray levels of a scan line gives a random waveform which can be represented by a relational tree.

We shall propose a texture model based on the structural approach. In this model, a texture pattern is divided into fixed-size windows. Repetition of subpatterns or a portion of a subpattern may appear in a

window. For all cases, we shall treat a windowed pattern as a subpattern. A tree grammar is then used to characterize windowed patterns of the same class. This model can be used for texture synthesis as well as discrimination. Since the windowed patterns are also a part of the global structure of the texture, a higher level of syntax rules can also be constructed for the arrangement of windowed patterns.

5.2 A Syntactic Model for Texture Analysis

We propose the following syntactic model for texture analysis: its primitive, window, tree representation, and tree grammar being described here.

5.2.1 The Primitive

We choose a single pixel with different gray levels to be the pattern primitive. For a picture of ℓ gray levels, we have ℓ different primitives. However, the size of a primitive can certainly be larger than a single pixel. For example, a window of n x n pixels with a relatively uniform gray level would be a good primitive.

5.2.2 The Window

From the structural point of view, texture is the placement of structured subpatterns. (See Figure 5.1 (b)). However, in a natural scene, the exact boundary of a subpattern is usually vague and unidentifiable. Often, subpatterns of the same texture appear in various sizes, shapes, or brightness. In some cases, there even exists a situation in which there are no well-defined subpatterns. For examples of this see Figures 5.1 (c) and 5.1 (d). In addition, the placement of subpatterns can be irregular and distorted such as shown in Figure 5.1 (a). Nevertheless,

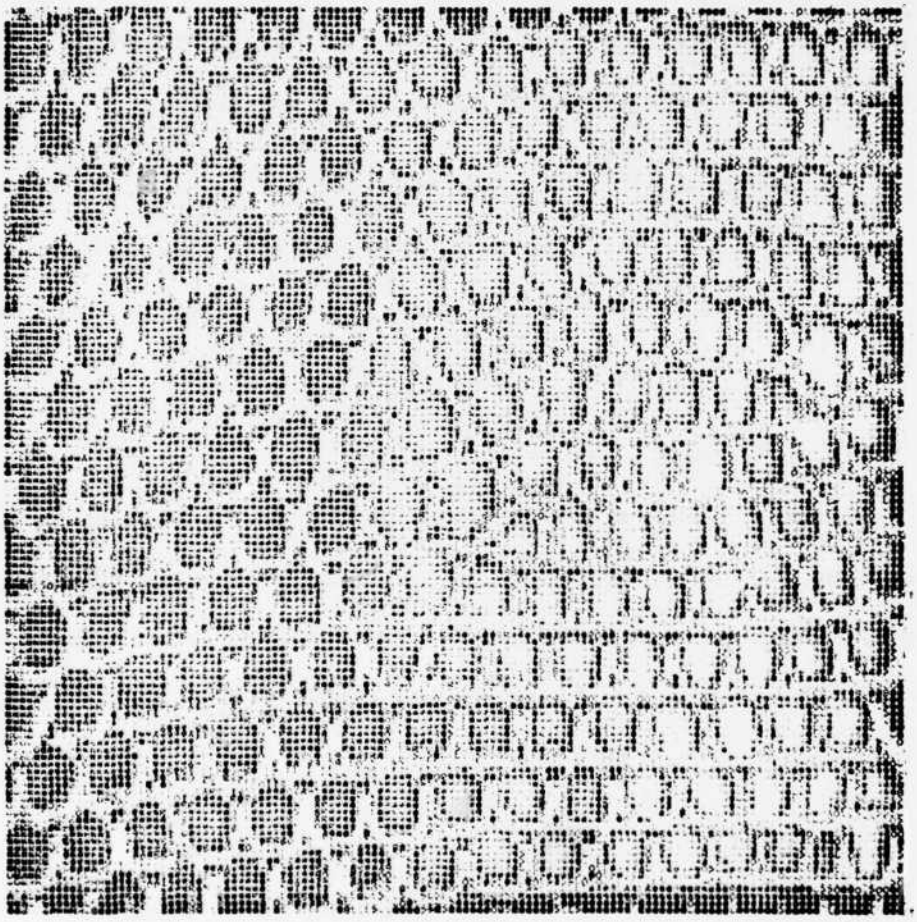


Figure 5.1(a) Pattern D22. Reptile Skin.

Figure 5.1 Four texture patterns obtained from digitizing pictures found in Brodatz's book, Textures.

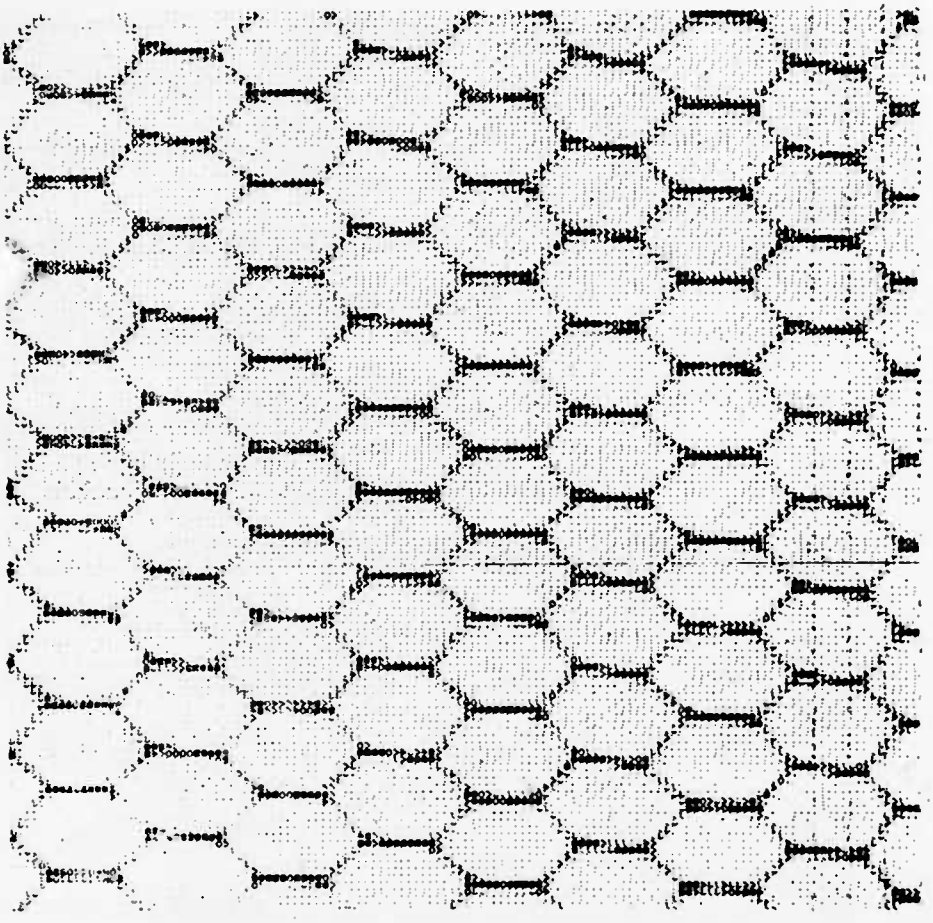


Figure 5.1(b) Pattern D34. Netting.

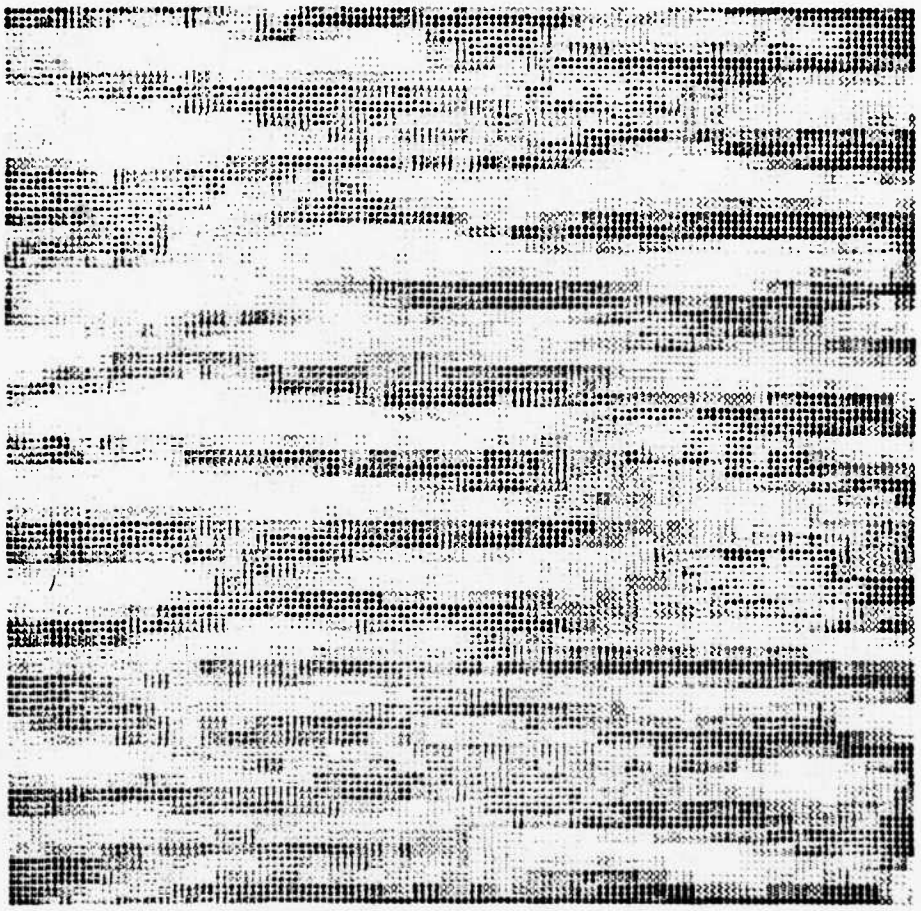


Figure 5.1(c) Pattern D38. Water.

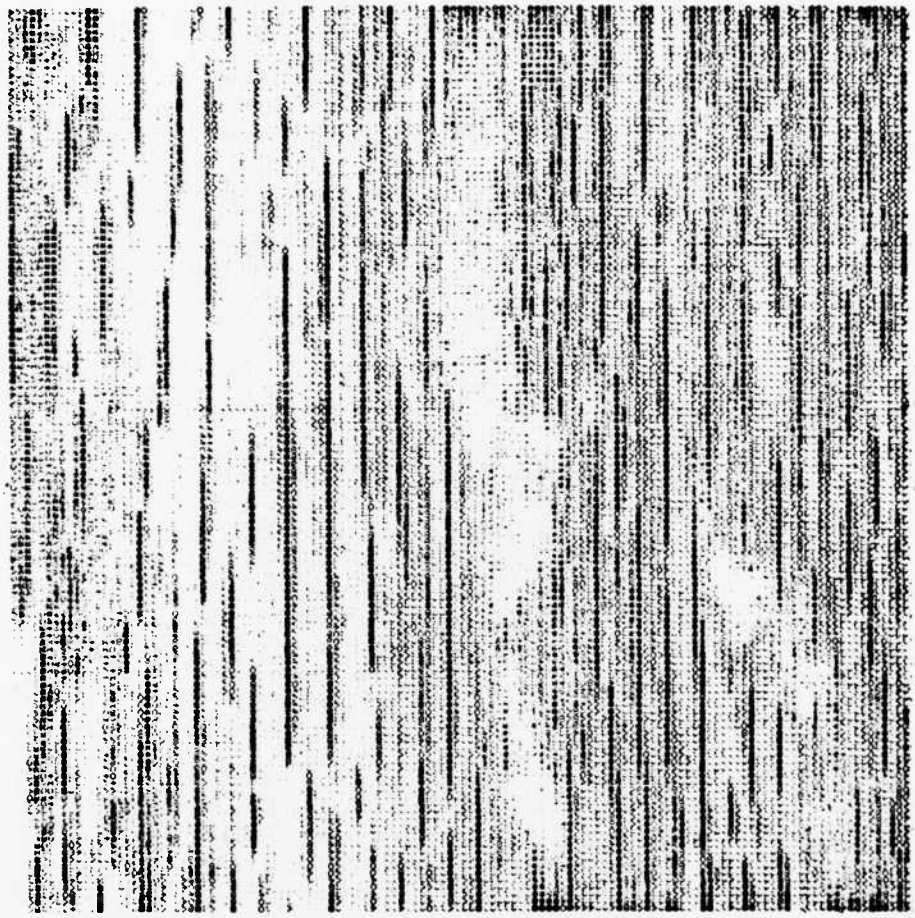
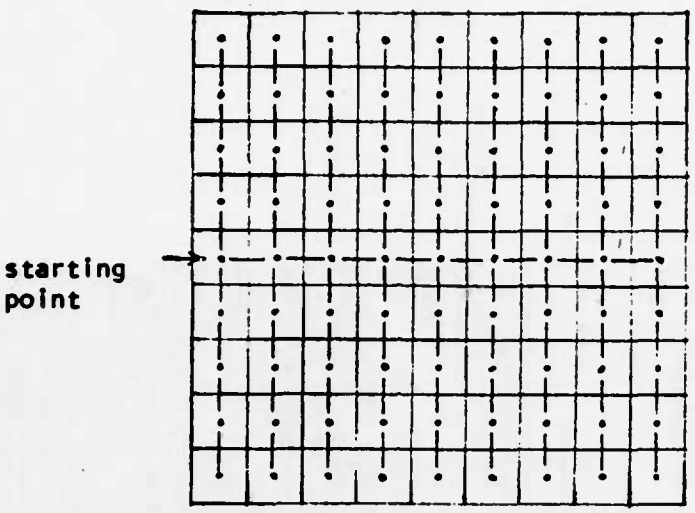
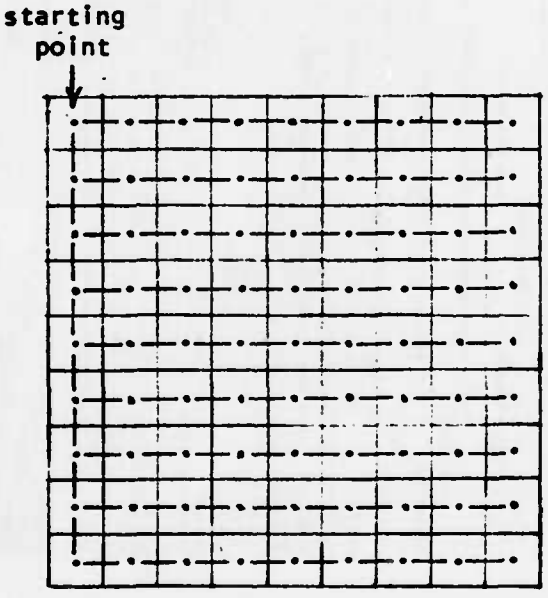


Figure 5.1(d) Pattern D68. Woodgrain.

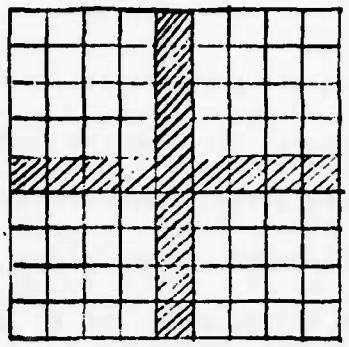


(a) Structure A



(b) Structure B

Figure 5.2 Two tree structures for texture modeling.



(a)

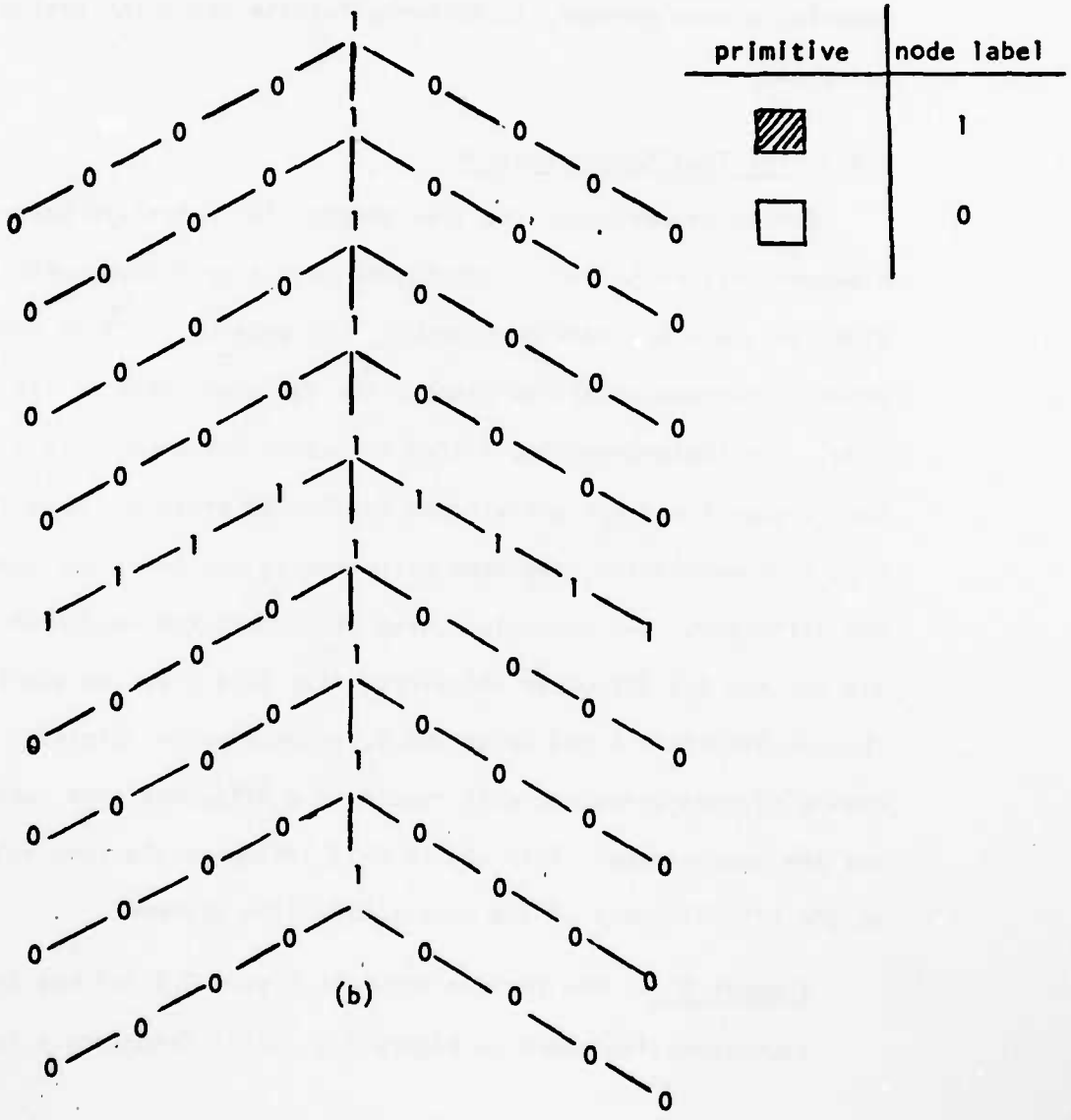


Figure 5.3 Pattern and its tree representation.

a small subframe of the overall texture pattern does maintain some of the characteristics of the texture. To make the syntactic approach practically feasible, pictures are divided into fixed-size windows. A grammar is then used to characterize the windowed pattern of the given texture. Assuming that the window is of size k x k, there are ℓ^{k^2} possible patterns. The set of all the windowed patterns of a particular texture is a subset of the ℓ^{k^2} patterns. Consequently, a high-dimensional regular grammar, for example, a tree grammar, is suitable for the characterization of texture patterns.

5.2.3 The Tree Representation

Before we construct the tree grammar for a texture pattern, each windowed pattern is first transformed into a tree representation. Each pixel in a k x k window corresponds to a node on its tree representation. Hence, a pattern primitive becomes the assigned label to its corresponding node. For implementation, a tree structure can be arbitrarily chosen, but is then fixed for all windows during the process. That is, for all tree representations, the tree structure is the same, but node labels are different. Two convenient tree structures are suggested in Figures 5.2 (a) and 5.2 (b) where the window size is 9 x 9. We shall refer to them as Structure A and Structure B, respectively. Clearly, a different choice of tree structure will result in a different tree representation for the same window. This choice will influence the complexity as well as the effectiveness of the constructed tree grammar.

Example 5.1. The pattern shown in Figure 5.3 (a) has the tree representation shown in Figure 5.3 (b) if Structure A is used.

5.2.4 The Tree Grammar

Example 5.2. The following tree grammar G_1 will generate the tree representation shown in Figure 5.3 (b):

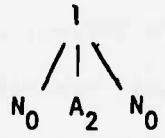
$G_1 = (V_1, r, P_1, A_1)$ over $\langle \Sigma, r \rangle$

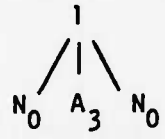
$V_1 = \{A_1, A_2, A_3, A_4, A_5, A_6, N_0, V_0, 1, 0\}$

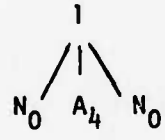
$\Sigma = \{$  $,$  $\}$

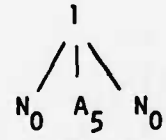
$r = \{0,1,2,3\}$

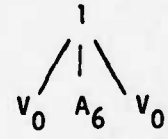
P_1:

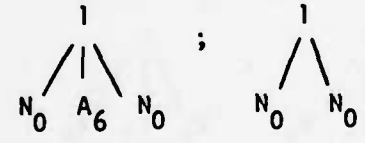
$A_1 \rightarrow$ 

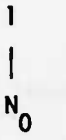
$A_2 \rightarrow$ 

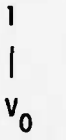
$A_3 \rightarrow$ 

$A_4 \rightarrow$ 

$A_5 \rightarrow$ 

$A_6 \rightarrow$ 

$N_0 \rightarrow$  ; 0

$V_0 \rightarrow$  ; 1

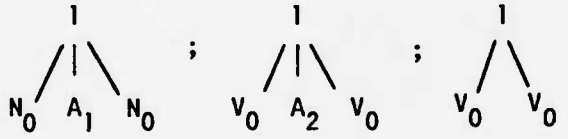
Example 5.3. Grammar G_1 in Example 5.2 will only accept patterns of a cross in the midle of the 9 x 9 window such as the texture pattern shown in Figure 5.4. In a natural texture, we will most likely have some distortions of the perfect pattern such as the pattern shown in Figure 5.5. Grammar G_2 will generate patterns having a shifting of the cross in Figure 5.3 (a) anywhere within the window.

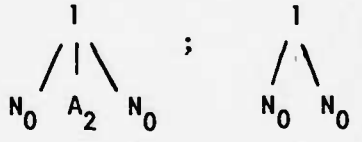
$G_2 = (V_2, r, P_2, S_2)$ over $<\Sigma, r>$

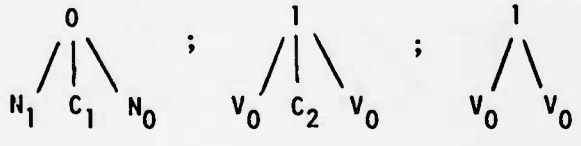
$V_2 = \{A_1, A_2, C_1, C_2, D_1, D_2, E_1, E_2, F_1, F_2, G_1, G_2, H_1, H_2, I_1, I_2, J_1, J_2,$
$N_0, N_1, N_2, N_3, N_4, V_0, 1, 0\}$

$S_2 = \{A_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1, J_1\}$

P_2:

$A_1 \rightarrow$  ; ;

$A_2 \rightarrow$  ;

$C_1 \rightarrow$  ; ;

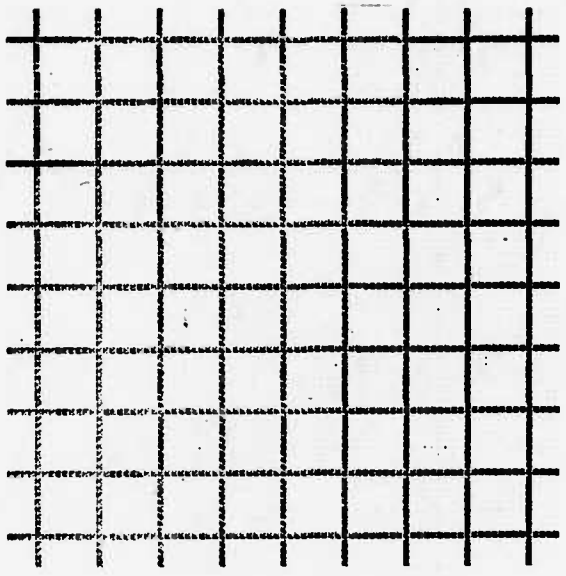


Figure 5.4 A regular texture pattern.

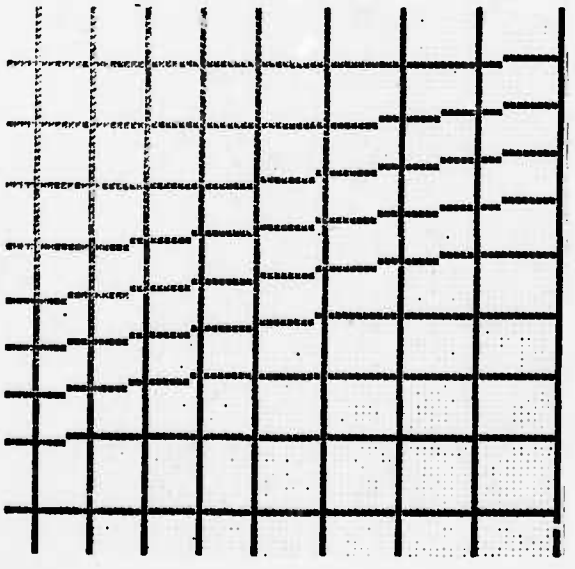
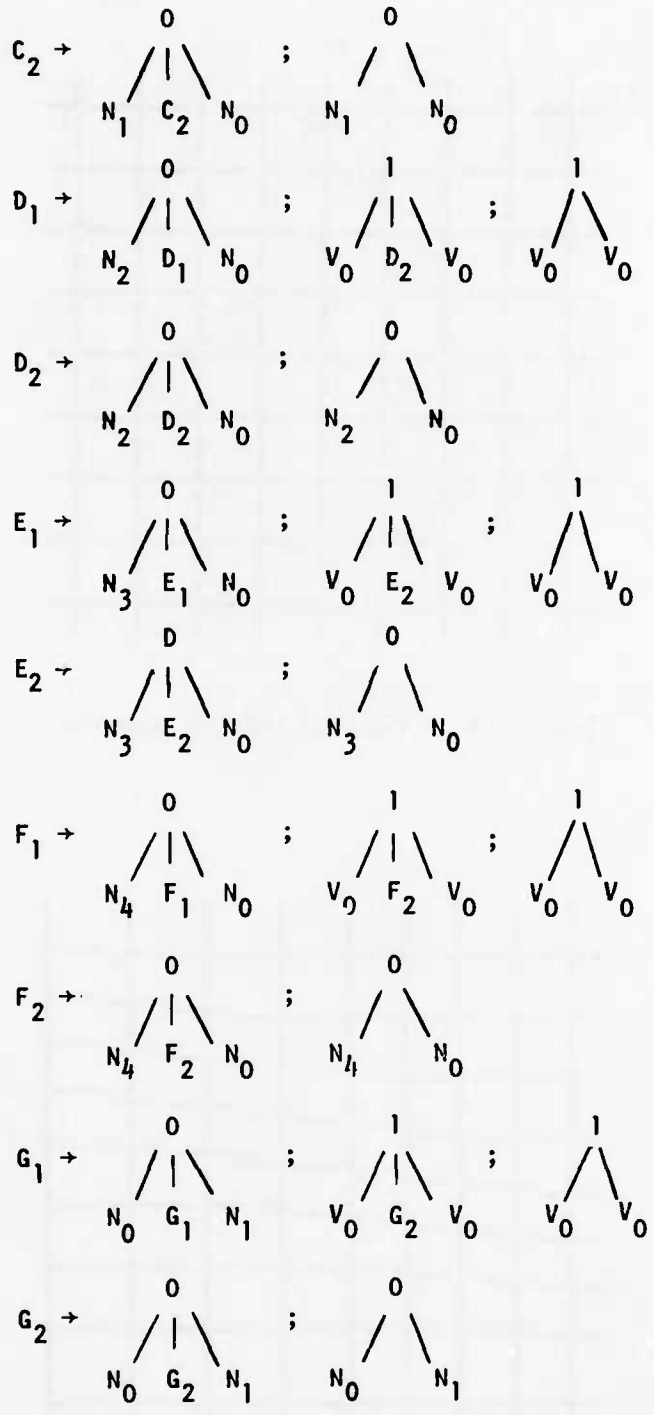
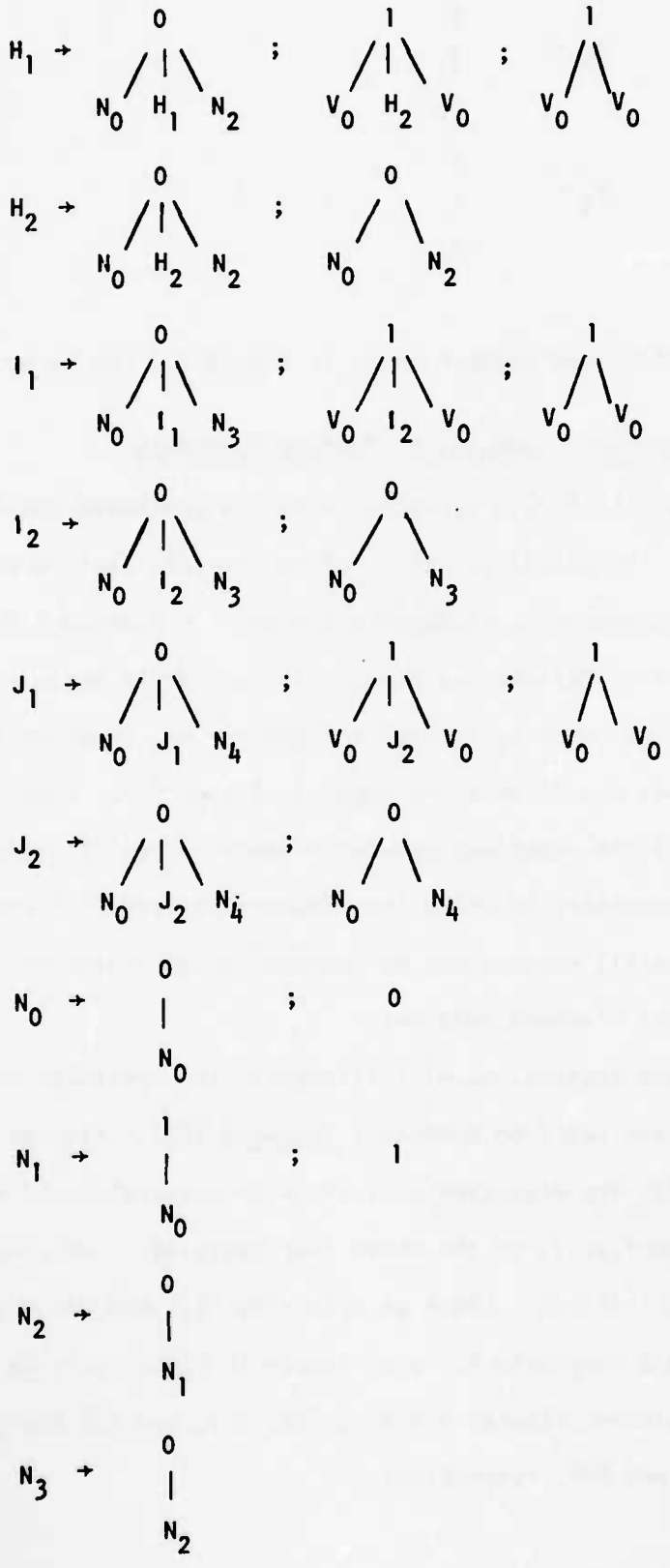


Figure 5.5 A distorted texture pattern which can be generated by G_2.


C2 → 0 ; 0
N1 C2 N0 N1 N0
D1 → 0 ; 1 ; 1
N2 D1 N0 V0 D2 V0 V0 V0
D2 → 0 ; 0
N2 D2 N0 N2 N0
E1 → 0 ; 1 ; 1
N3 E1 N0 V0 E2 V0 V0 V0
E2 → D ; 0
N3 E2 N0 N3 N0
F1 → 0 ; 1 ; 1
N4 F1 N0 V0 F2 V0 V0 V0
F2 → 0 ; 0
N4 F2 N0 N4 N0
G1 → 0 ; 1 ; 1
N0 G1 N1 V0 G2 V0 V0 V0
G2 → 0 ; 0
N0 G2 N1 N0 N1


H1 → 0 N0 H1 N2 ; 1 V0 H2 V0 ; 1 V0 V0
H2 → 0 N0 H2 N2 ; 0 N0 N2
I1 → 0 N0 I1 N3 ; 1 V0 I2 V0 ; 1 V0 V0
I2 → 0 N0 I2 N3 ; 0 N0 N3
J1 → 0 N0 J1 N4 ; 1 V0 J2 V0 ; 1 V0 V0
J2 → 0 N0 J2 N4 ; 0 N0 N4
N0 → 0 N0 ; 0
N1 → 1 N0 ; 1
N2 → 0 N1
N3 → 0 N2

$$N_4 \rightarrow \begin{array}{c} 0 \\ | \\ N_3 \end{array}$$

$$V_0 \rightarrow \begin{array}{c} 1 \\ | \\ V_0 \end{array} \quad ; \quad 1$$

The distorted pattern shown in Figure 5.5 can be accepted by G_2.

5.3 Illustrative Examples of Texture Synthesis

In Section 5.2, a syntactic model is presented for describing windowed patterns. The global structure of the overall texture pattern depends on the arrangement of windowed patterns. In Example 5.3, we constructed the grammar G_2 for the acceptance of the texture pattern shown in Figure 5.5. However, when G_2 is used for generation, numerous patterns might be produced, one of which is shown in Figure 5.6. Therefore, in order to preserve the coherence between windows, a set of higher level syntax rules is necessary in which the windowed pattern is treated as a primitive, and the overall texture can be represented as a tree which decides the placement of windowed patterns.

In this section, we will illustrate the synthesis of patterns D22, D34, D38, and D68 from Brodatz's Textures [92]. Figures 5.1 (a), (b), (c), and (d) are digitized pictures with resolutions of 400μ, 400μ, 100μ and 400μ, respectively, of the above four patterns. For simplicity, we use only two primitives: black as primitive "1," and white as primitive "0." By setting a threshold for gray levels in Figure 5.1, we obtain four binary pictures, Figures 5.7 (a), (b), (c), and (d) for patterns D22, D34, D38, and D68, respectively.

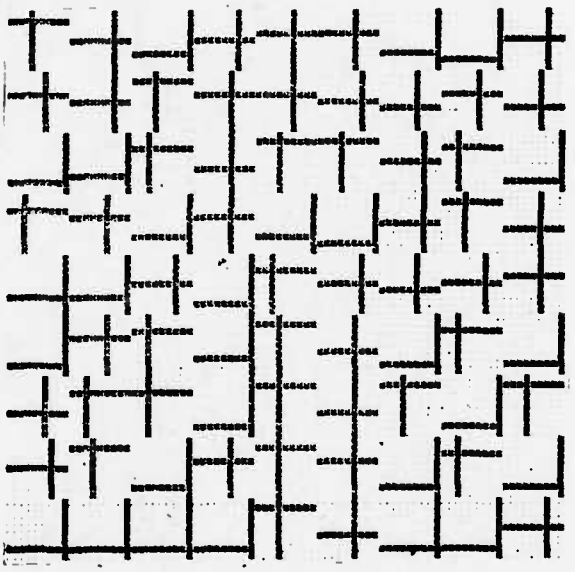


Figure 5.6 A random texture pattern which could be generated by G_2.

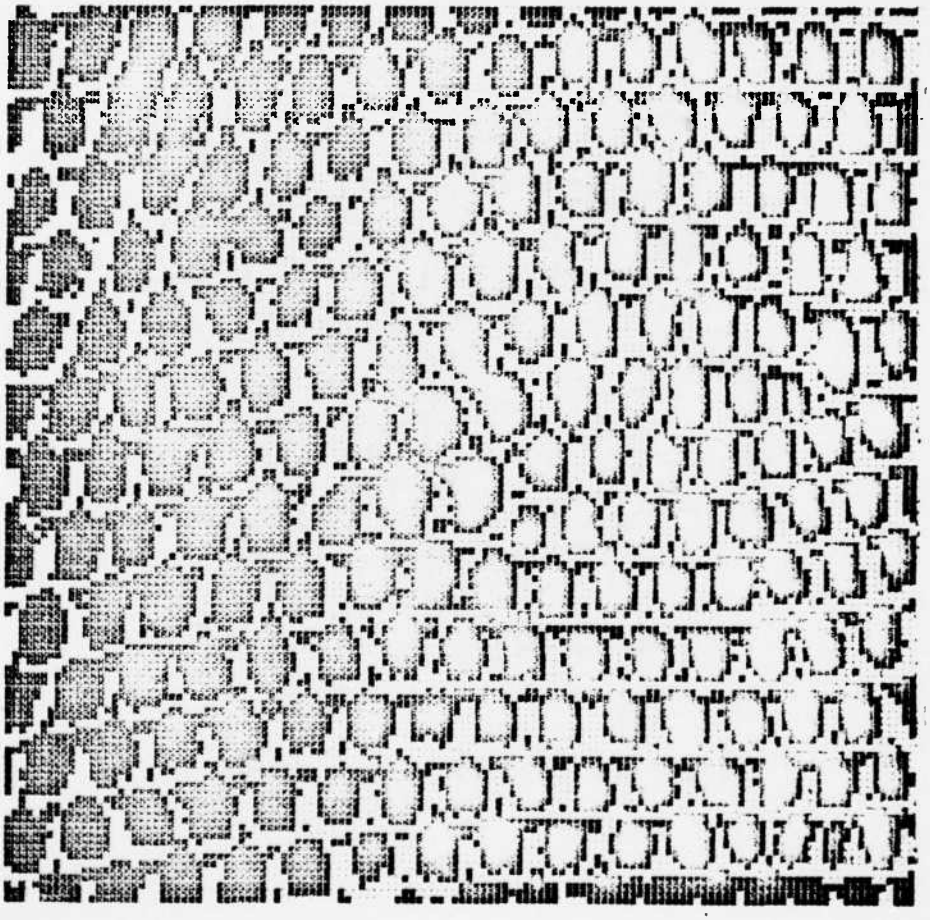


Figure 5.7(a) Binary picture of pattern D22.

Figure 5.7 Binary pictures of patterns in Figure 5.1.

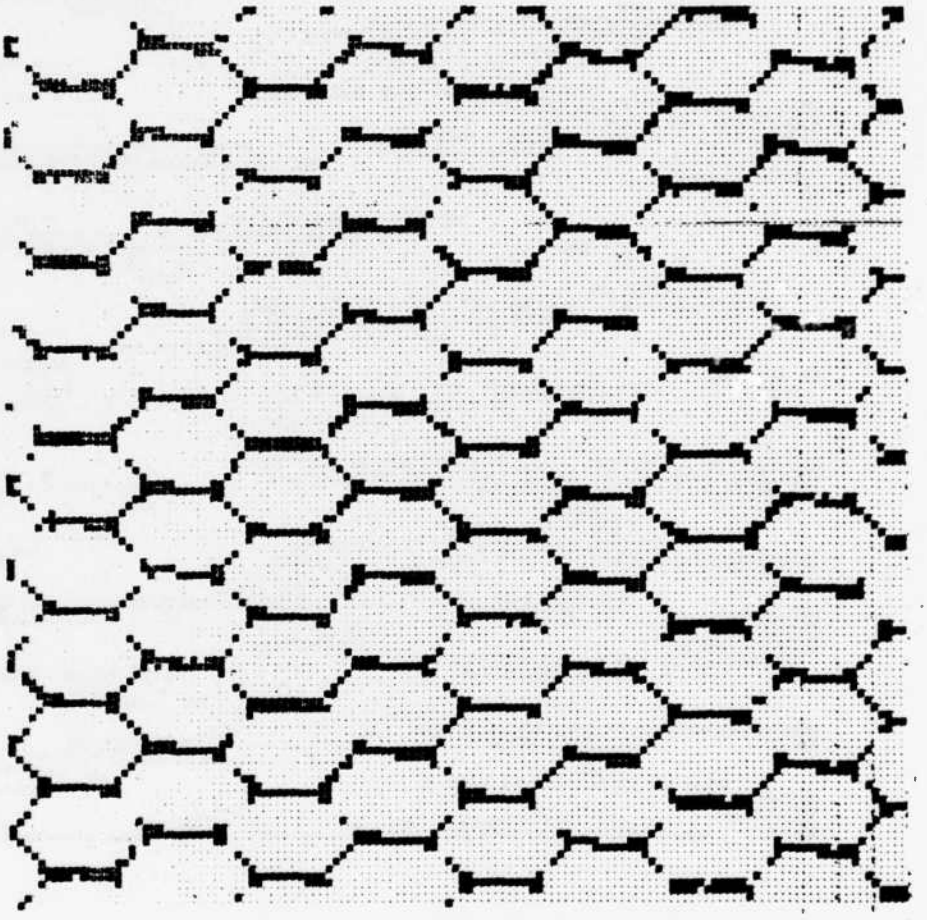


Figure 5.7(b) Binary picture of pattern D34.

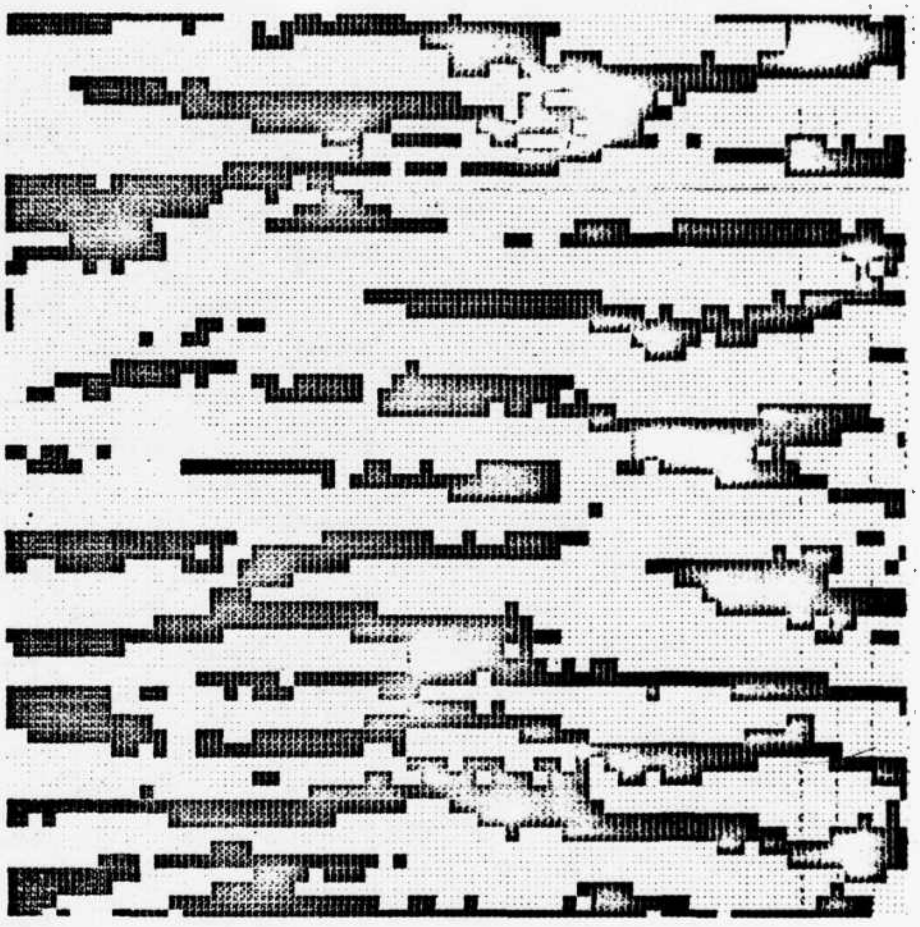


Figure 5.7(c) Binary picture of pattern D38.

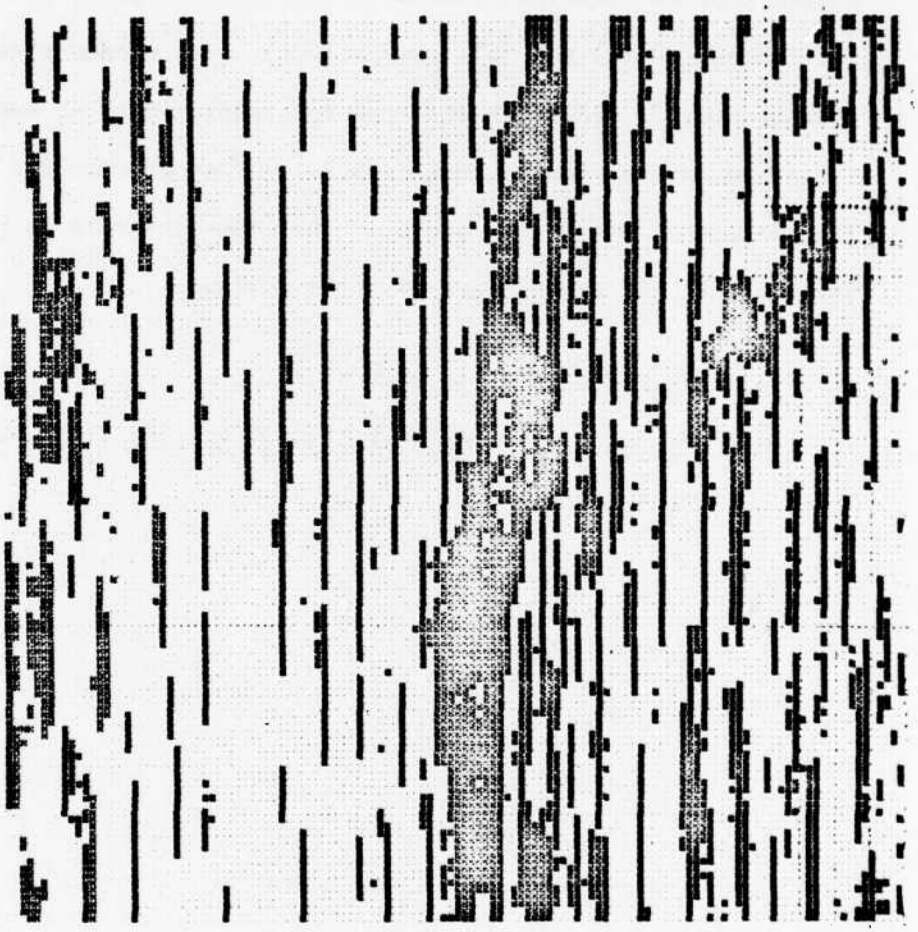


Figure 5.7(d) Binary picture of pattern D68.

5.3.1 Regular Tesselation

The texture pattern D34 is a hexagonally tesselated pattern which is nearly what Zucker called "ideal texture."

Example 5.4. A syntheses of hexagonal tesselation is as follows: Assume that we have two windowed patterns; namely, A_1 and C_1 shown in Figures 5.8 (a) and (b), respectively, with window-size 9 x 9. Tree grammar G_3 generates the tree representations, A_1 and C_1, using Structure A described in Section 5.2.3. Tree grammar G_3' generates the placement rule for A_1 and C_1. The placement rule is given in Structure B. G_3 and G_3' are given as follows:

$$G_3 = (V_3, r, P_3, \{A_1, C_1\}) \text{ over } \langle \Sigma, r \rangle$$

$$V_3 = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, C_1, C_2, C_3, C_4, C_5, C_6, C_7, N_1, N_2, N_3, N_4, N_0, 1, 0\}$$

P_3:

$A_1 \rightarrow$ 1 (N_0 A_2 N_0)

$A_2 \rightarrow$ 1 (N_0 A_3 N_0)

$A_3 \rightarrow$ 1 (N_0 A_4 N_0)

$A_4 \rightarrow$ 0 (N_1 A_5 N_1)

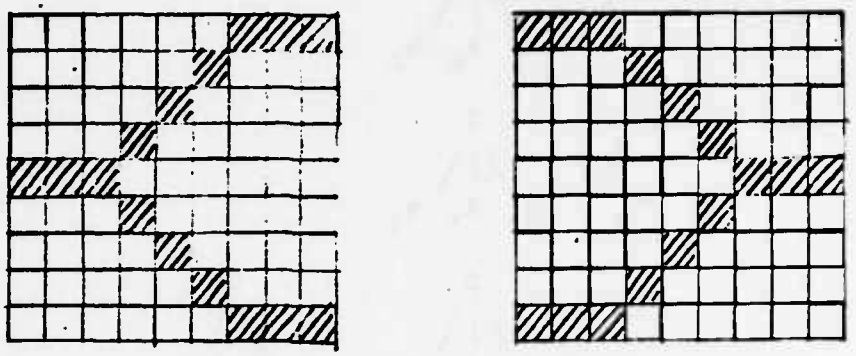


Figure 5.8 Windowed patterns of hexagonal tesselation.

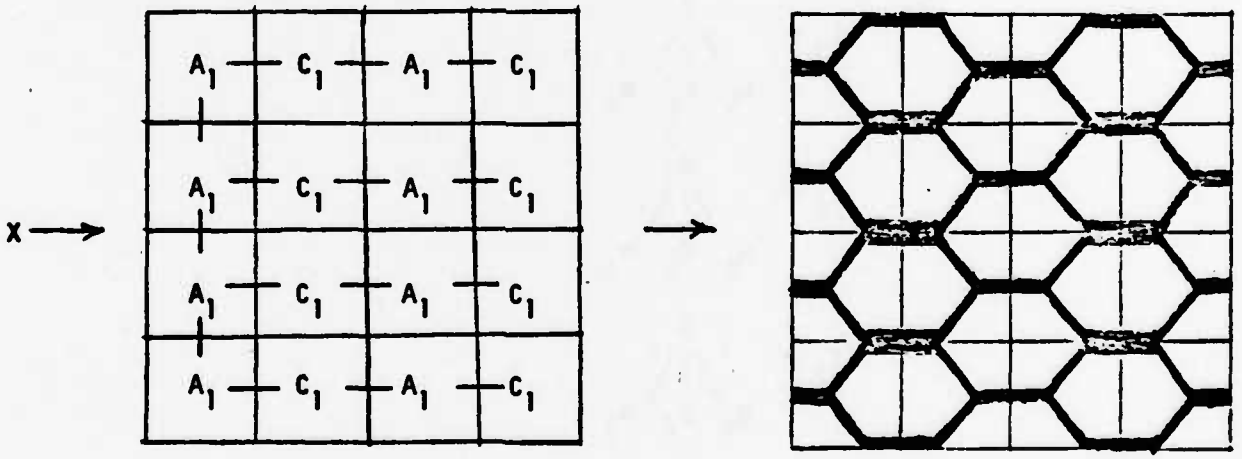
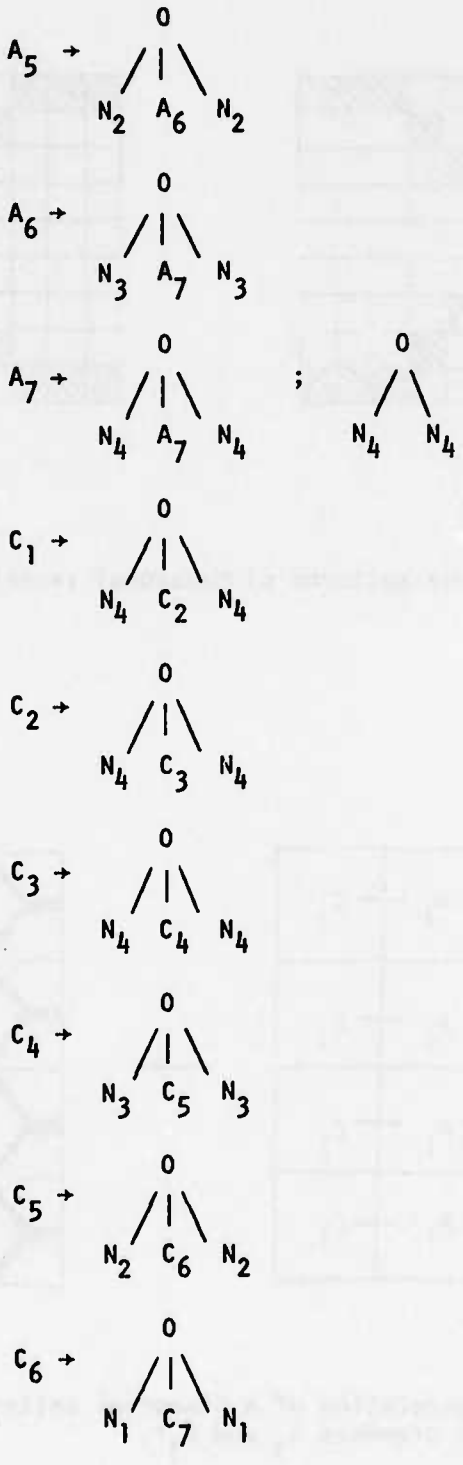
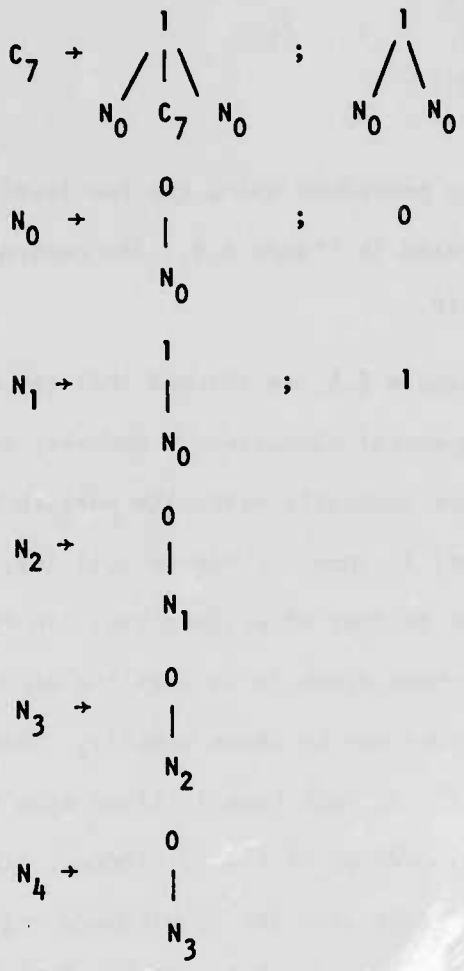


Figure 5.9 The generation of a hexagonal pattern by using grammars G_3 and G_3'.


A5 → O N2 A6 N2
A6 → O N3 A7 N3
A7 → O N4 A7 N4 ; O N4 N4
C1 → O N4 C2 N4
C2 → O N4 C3 N4
C3 → O N4 C4 N4
C4 → O N3 C5 N3
C5 → O N2 C6 N2
C6 → O N1 C7 N1

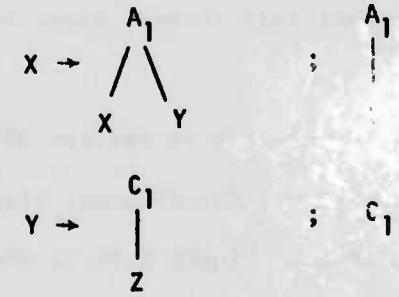


$G_3' = (V_3', r', P_3', X)$ over $\langle \Sigma', r' \rangle$

$V_3' = \{X, Y, Z, A_1, C_1\}$

$\Sigma' = \{A_1, C_1\}$, $r' = \{0, 1, 2\}$

P_3':



$$Z \rightarrow \begin{matrix} A_1 \\ | \\ Y \end{matrix} \quad ; \quad A_1$$

The generating procedure using the two-level syntax rules, G_3 and G_3' is illustrated in Figure 5.9. The generated pattern is shown in Figure 5.10.

Example 5.5. In Example 5.4, we assumed that the window size matched the size of the hexagonal subpattern. However, the 9 x 9 window in Example 5.4 does not perfectly match the pattern D34 in Figure 5.7 (b). An improvement is shown in Figure 5.11 (b), where the hexagons have a similar size to that of pattern D34. In Figure 5.11 (b), window frames have been drawn in so that the windowed patterns and their repetitive order can be shown clearly. There are 20 different windowed patterns within each heavily lined area in a 4 x 5 arrangement. The larger pattern, made up of the 20 windows, also repeats itself. The grammar G_4 that generates the 20 windowed patterns, is given on the left-hand side of Appendix E-1. Figure 5.11 (a) shows the placement rule in Structure B for the pattern in Figure 5.11 (b). The symbol in each cell of Figure 5.11 (a) belongs to the set of starting symbols in grammar G_4. From each starting symbol, the corresponding windowed pattern of Figure 5.11 (b) can be generated. The grammar that generates the placement rule is also given in Appendix E-1 as grammar G_4'.

Example 5.6. The uneven brightness in pattern D34, e.g., darker for horizontal lines and lighter for diagonal lines, can be simulated by using a stochastic grammar. Figure 5.12 is the resulting

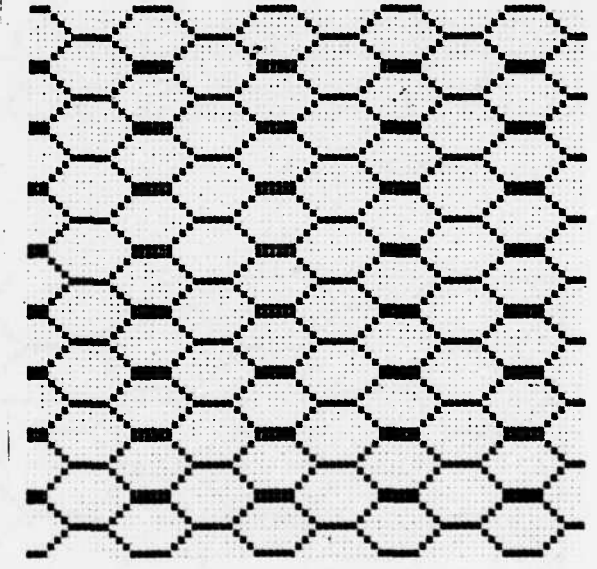
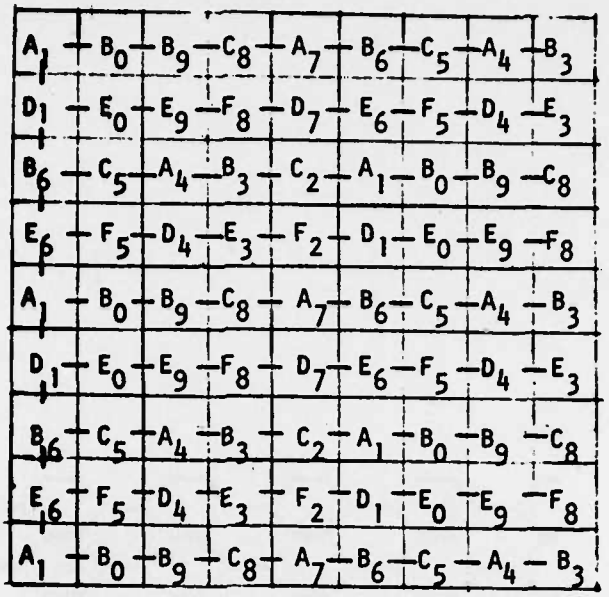
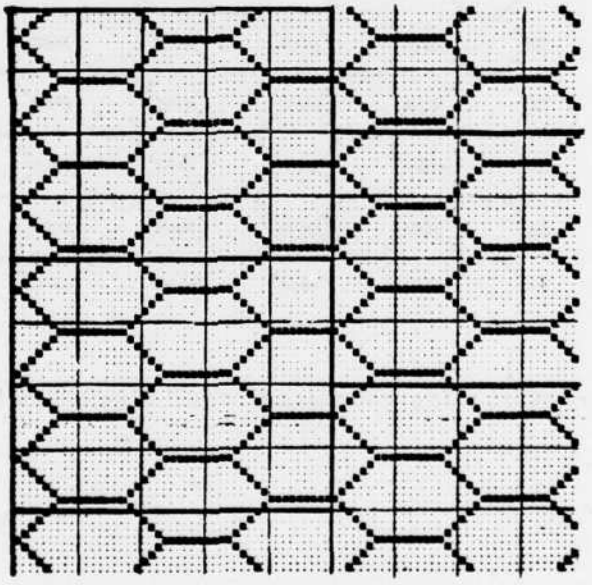


Figure 5.10 The generated regular hexagonals.



(a) Placement rule



(b) Result Pattern

Figure 5.11 Generated regular hexagonals.

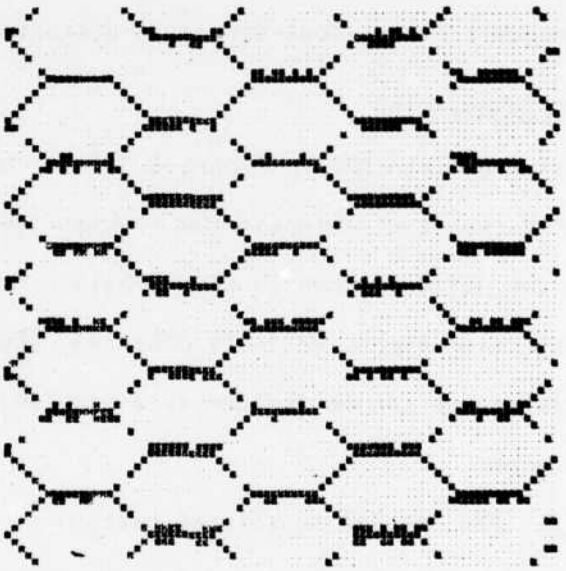


Figure 5.12 Simulated result of pattern D34.

pattern from using stochastic grammar G_{S_4} in the generation of the "noisy version." The grammar G_{S_4} is given on the right-hand side in Appendix E-1.

In this subsection, three examples were given: (1) to illustrate the synthesis of an ideal texture for a matching sized window and subpattern, (2) for an unmatched size window and subpattern, and (3) for a noisy version. However, the noisy version described in Example 5.6 is the result of local noise. In the following subsection, we shall describe the synthesis of a global structure-distorted texture pattern.

5.3.2 Irregular Tesselation

Let us examine pattern D22 in Figure 5.7 (a). We may consider that pattern D22 is the result of twisting the regular tesselation of an ideal texture such as the pattern shown in Figure 5.13. From a single window the trend of distortion cannot be fully detected. For texture synthesis, such a global distortion can be treated as a problem of the placement of windowed patterns.

Example 5.7. The regular tesselated pattern shown in Figure 5.13 is composed of two basic patterns shown in Figure 5.14 (a) and (b). A distorted tesselation can result from shifting a series of basic patterns in one direction. Let us use the set of patterns resulting from shifting a basic pattern as the set of primitives. There will be 81 such windowed pattern primitives. We shall refer to them simply as primitives in this example. Figure 5.15 shows several of them. Each primitive is given a name of two symbols. "X_i," where $X \in \{A,B,C,D,E,F,G,H,I\}, i \in \{1,2,\ldots,9\}$. Starting from X_i,

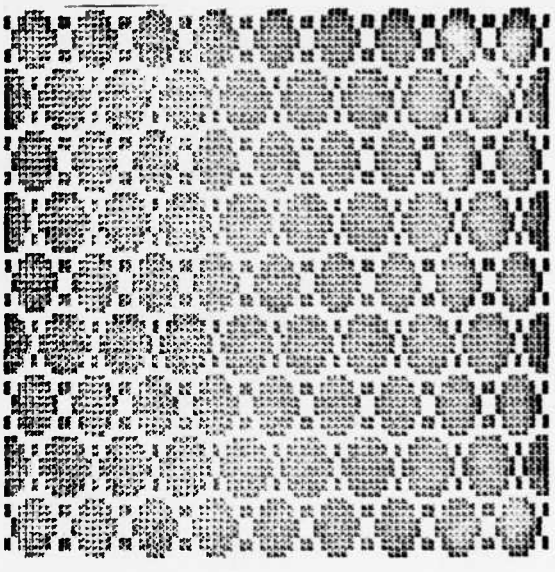
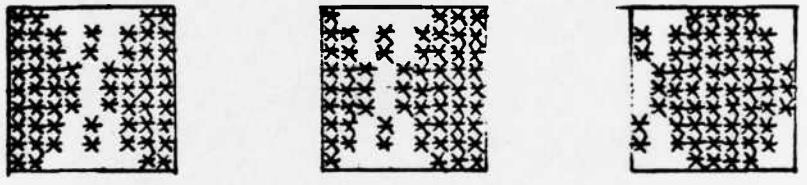


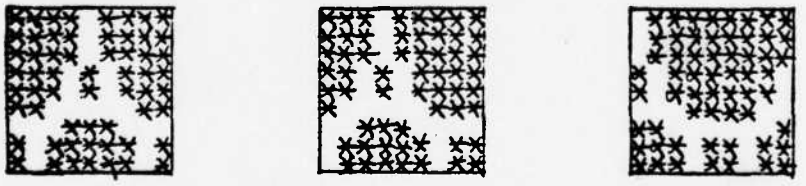
Figure 5.13 The ideal texture of pattern D34.

(a) (b)

Figure 5.14 Basic pattern of Figure 5.13.



A_1 A_2 A_5



D_1 D_2 D_5

Figure 5.15 Windowed pattern primitives.

the pattern resulting from shifting one column to the left will be named X_{i+1}, and the pattern resulting from shifting one row up will be named Y_1. Grammar G_5 in Appendix E-2 is constructed for the generation of the 81 primitives.

Several synthesis results are given in Figure 5.16 (a), (b), and (c). Tree representations using Structure B that decide the placement of windowed patterns are shown at the left-hand side of each pattern.

Using the same idea as that in Example 5.6, a stochastic grammar can be used to add local distortions. An example of this is shown in Figure 5.16 (d). We can also construct a grammar for the placement of windowed patterns for certain types of structure distortion. For example, a twisted upward or downward pattern, or an insertion of an extraneous row of subpatterns, etc.

5.3.3 Random Pattern

The texture pattern D38 and D68 in Figures 5.7 (c) and (d) show a higher degree of randomness than D22 and D34. No clear tesselation or subpattern exists in the pattern.

Example 5.8. The water waves in pattern D38 can be described as a belt extending in the horizontal direction, varying in width and twisting upward or downward. Assuming that, at most, one belt can appear in a window, we shall use Structure A for tree representation and a stochastic tree grammar G_6 to describe such patterns. In each production rule in G_6, the left-hand side nonterminal is the present state and the right-hand side generates the width and the position of the belt that the present state represented, as well as the next state. Figure 5.17 illustrates this generation process. The

A_5	B_5	C_5	D_5	E_5	F_5	G_5	H_5	I_5
A_1	B_1	C_1	D_1	E_1	F_1	G_1	H_1	I_1
A_6	A_6	B_6	C_6	D_6	E_6	F_6	G_6	H_6
A_2	A_2	B_2	B_2	C_2	D_2	E_2	F_2	G_2
A_7	A_7	B_7	B_7	C_7	D_7	E_7	F_7	G_7
A_3	A_3	A_3	B_3	B_3	C_3	D_3	E_3	F_3
A_8	A_8	A_8	B_8	B_8	C_8	C_8	D_8	E_8
A_3	A_3	A_3	B_3	B_3	C_3	C_3	C_3	D_3
A_6	A_6	A_6	A_6	B_6	B_6	B_6	C_6	C_6

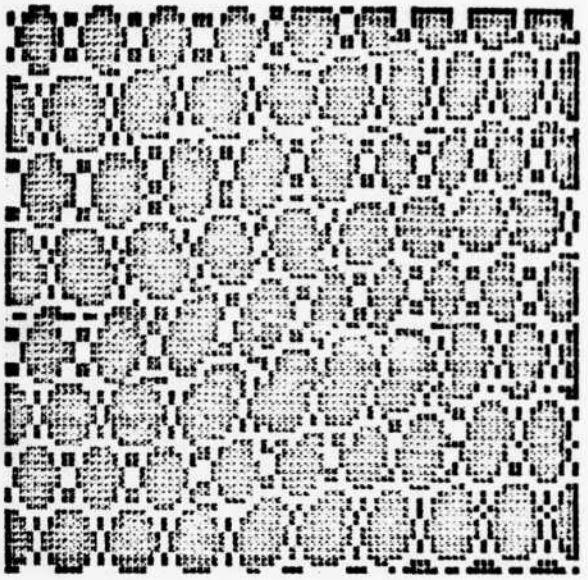


Figure 5.16(a)

Figure 5.16 Synthesis results of pattern D22.

A_5	A_5	I_1	I_2	H_2	H_2	H_3	H_3	H_4
C_1	B_1	B_1	A_1	A_2	I_6	I_7	I_8	A_4
E_5	E_5	D_5	C_6	C_7	B_7	B_8	C_8	C_9
G_1	F_1	E_2	D_3	B_3	A_4	B_4	C_5	D_5
I_5	I_6	I_7	I_9	B_8	D_9	F_1	H_1	I_2
A_5	A_5	A_6	B_7	G_4	G_5	G_6	G_7	G_8
A_1	A_1	B_1	B_2	A_3	I_8	I_8	H_8	H_8
A_5	A_5	A_5	A_5	A_6	A_7	I_3	I_3	I_3
A_1	A_1	A_1	A_1	A_1	A_2	A_2	A_2	A_2

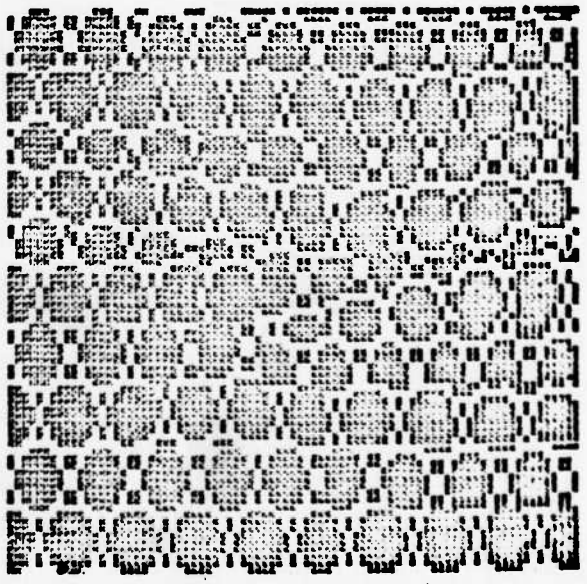


Figure 5.16(b)

H_4	H_4	I_5	A_1	A_1	I_7	H_8	H_8	H_9
G_6	H_7	I_8	A_5	A_5	I_3	H_4	G_5	G_5
F_3	G_3	H_4	A_1	A_1	H_8	G_9	F_9	F_9
F_6	G_7	H_7	A_5	A_5	H_4	G_4	F_5	F_5
E_3	F_3	H_3	A_1	A_1	H_9	F_9	E_9	E_9
D_6	F_6	H_7	A_5	A_5	H_4	F_5	D_5	C_5
C_3	E_3	G_3	A_1	A_1	G_9	E_9	C_9	B_9
C_4	E_7	G_7	A_5	A_5	G_4	E_4	C_4	B_4
C_3	E_3	G_3	A_1	A_1	G_9	F_9	C_9	A_9

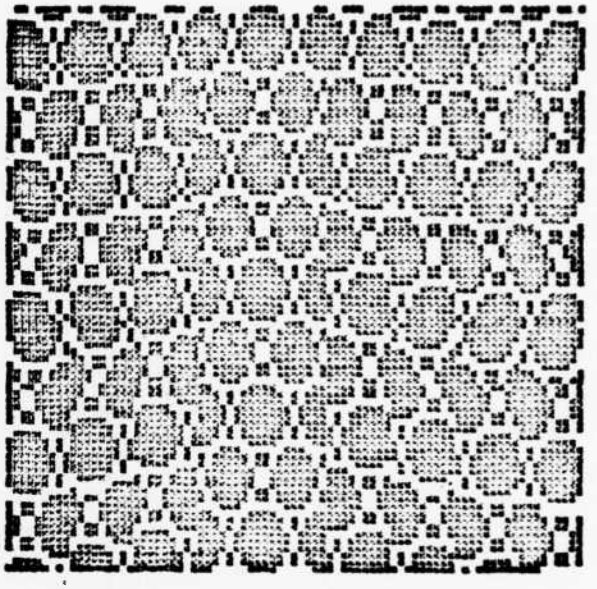


Figure 5.16(c)

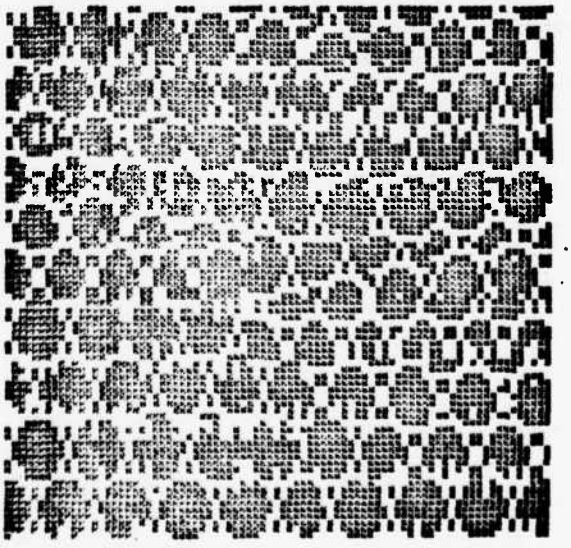


Figure 5.16(d)

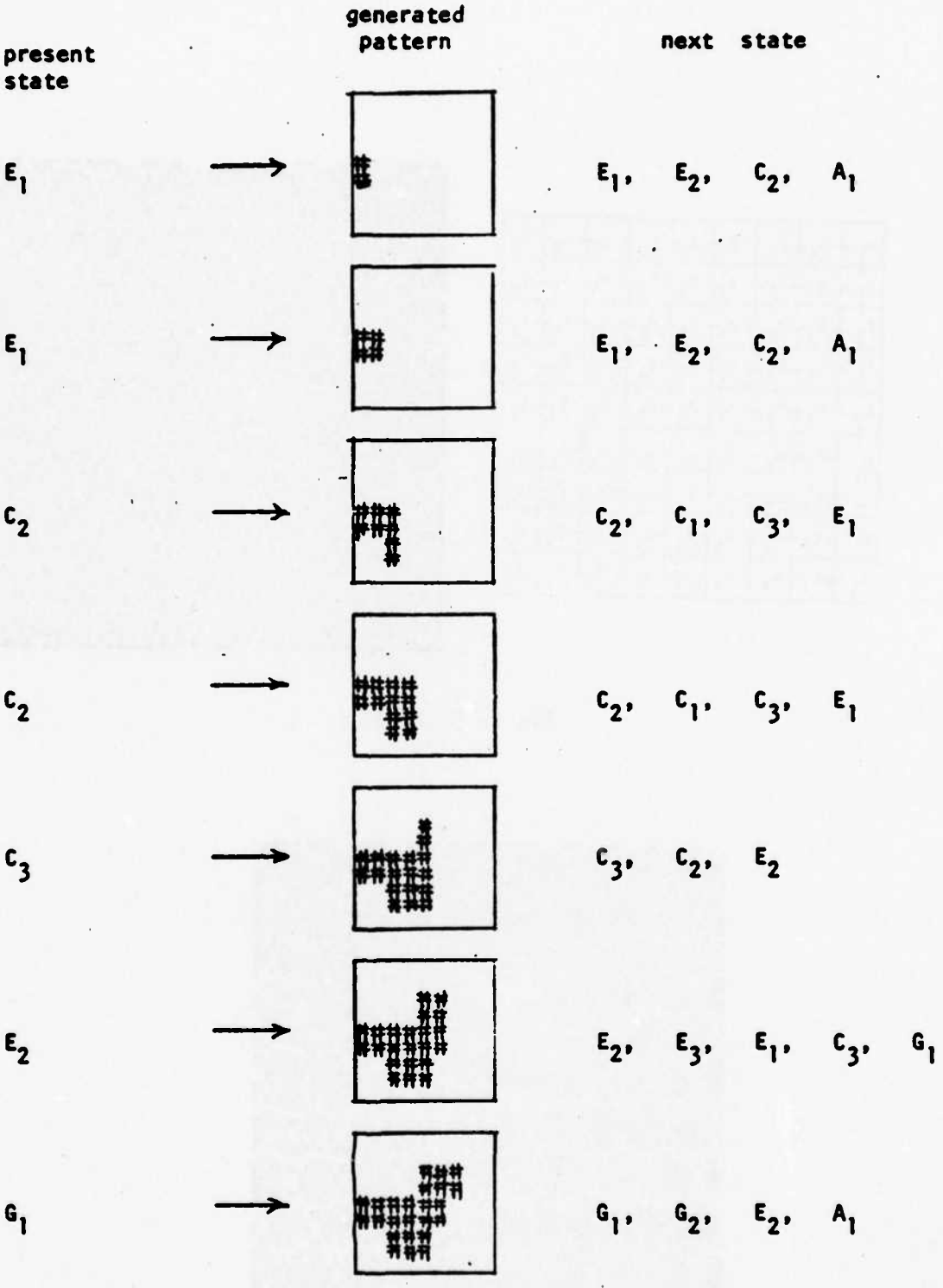


Figure 5.17 The syntactic generation of water waves.

grammar G_6 is given in Appendix F-1. G_6 is also the discrimination grammar for pattern D38 which will be discussed in Section 5.4. The production rules associated with zero probability are unused rules during pattern generation. They are added for pattern discrimination. The probabilities associated with the production rules in G_6 are arbitrarily assigned. By varying the assignment of probabilities, patterns with a different degree of brightness and fluctuation can be generated. Some resulting patterns are shown in Figure 5.18.

Example 5.9. The texture pattern of D68, the wood grain pattern, consists of long vertical lines. It is particularly convenient for syntactic description when Structure A is used for tree representation. The subpattern (vertical line) and its repetition can be fully characterized by the stochastic grammar G_7. Therefore, there is no need to generate the overall pattern window by window. The grammar G_7 is given as follows:

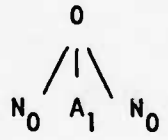
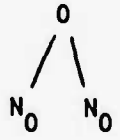
$G_7 = (V_7, r, P_7, A_1)$ over $\langle \Sigma, r \rangle$

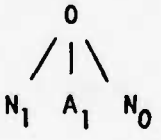
$V_7 = \{A_1, N_0, N_1, 0, 1\}$

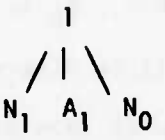
$r = \{0, 1, 2, 3\}$

$\Sigma = \{0, 1\}$

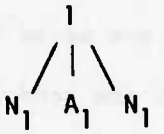
P_7:

$A_1 \rightarrow$  , 0.5 ;  , 0.05 ;

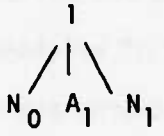
 , 0.09 ;



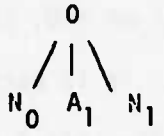
, 0.09 ;



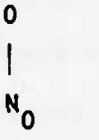
, 0.09



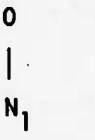
, 0.09 ;



, 0.09 ;

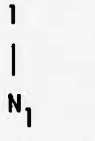
$N_0 \rightarrow$ 

, 0.90 ;

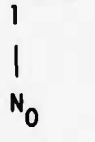


, 0.05 ;

0 , 0.05 ;

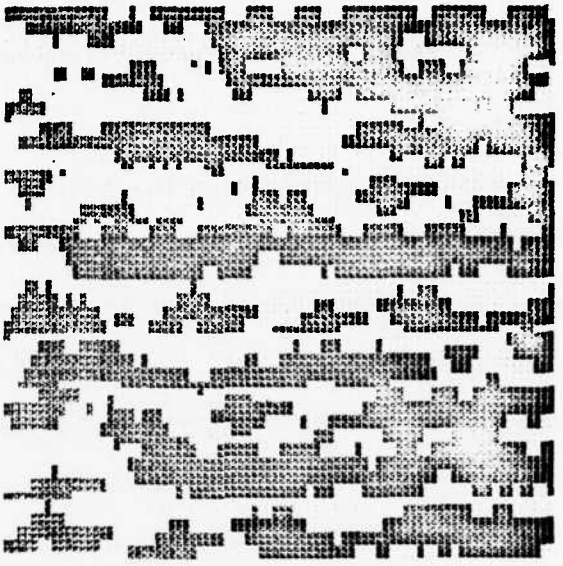
$N_1 \rightarrow$ 

, 0.85 ;

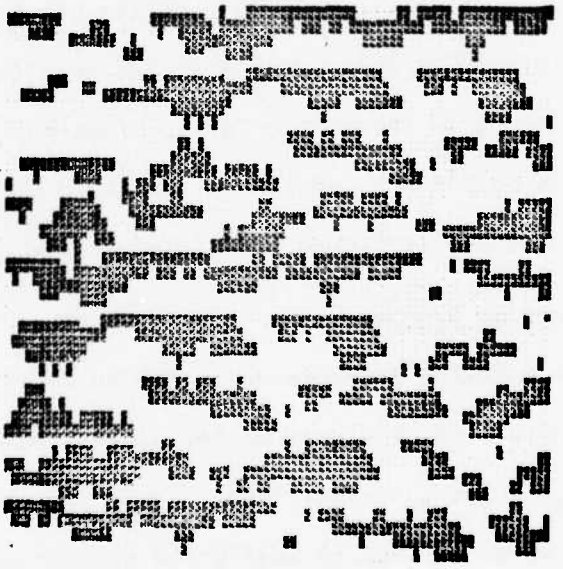


, 0.10 ;

1 , 0.05



(a)



(b)

Figure 5.18 Synthesis results of pattern D38.

The density of grains (vertical lines) depends on the probabilities associated with production rules. Pictures in Figure 5.19 are generated from G_7 using different probability assignments.

5.4 Texture Discrimination

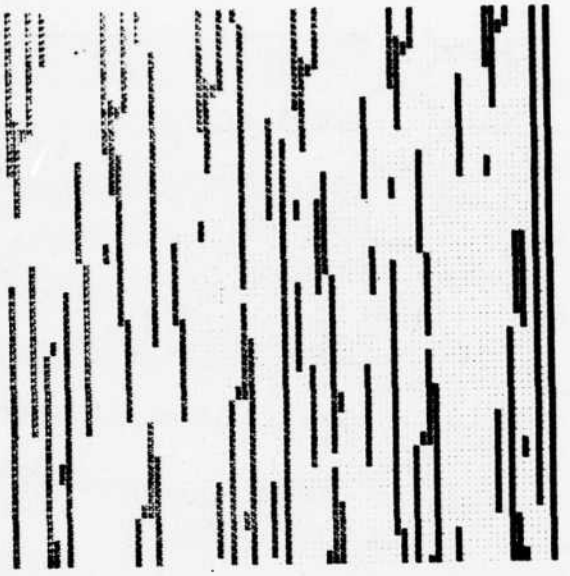
The proposed texture model can also be used for texture discrimination. In Section 5.3, we illustrated how a texture pattern was generated window-by-window. The construction of a grammar in modeling the variation of size, shape, and brightness, as well as noise and distortion was illustrated by examples. We also discussed in Section 5.2 that a pattern in a small subframe (window) maintains some of the characteristics of the overall texture. Under this assumption, we shall restrict the problem of texture discrimination to the recognition of windowed patterns only. Each picture is processed window-by-window.

5.4.1 Data Preparation

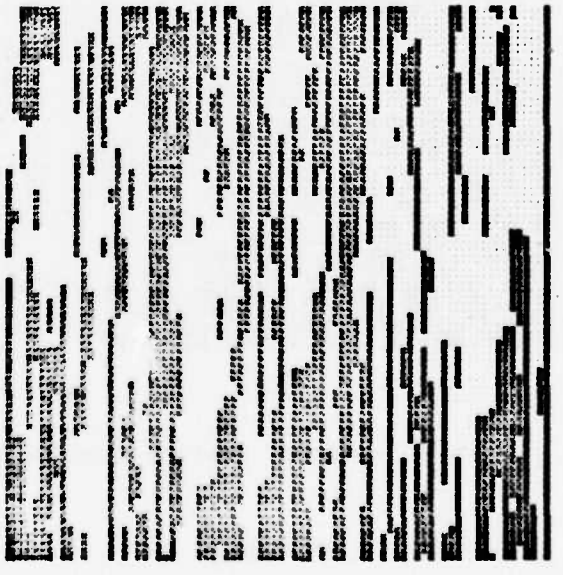
The pattern shown in Figure 5.20 consists of patterns D22, D34, D38, and D68. There are 180 x 180 pixels with 128 gray levels. We shall use two primitives (two gray levels) for discrimination. The picture shown in Figure 5.21 is obtained by setting a threshold at gray level 44. Window frames are drawn in Figure 5.21. The window size is 9 x 9.

5.4.2 Discrimination Grammars

The texture modeling grammars described in Section 5.3 are used for discrimination here. Let the grammar for pattern D22, D34, D38, and D68 be G_{22}, G_{34}, G_{38}, and G_{68}, respectively. From the viewpoint of discrimination, we would like to modify the grammar so that overlaps between $L(G_{22})$, $L(G_{34})$, $L(G_{38})$, and $L(G_{68})$ (languages generated from



(a)



(b)

Figure 5.19 Synthesis results of pattern D68.

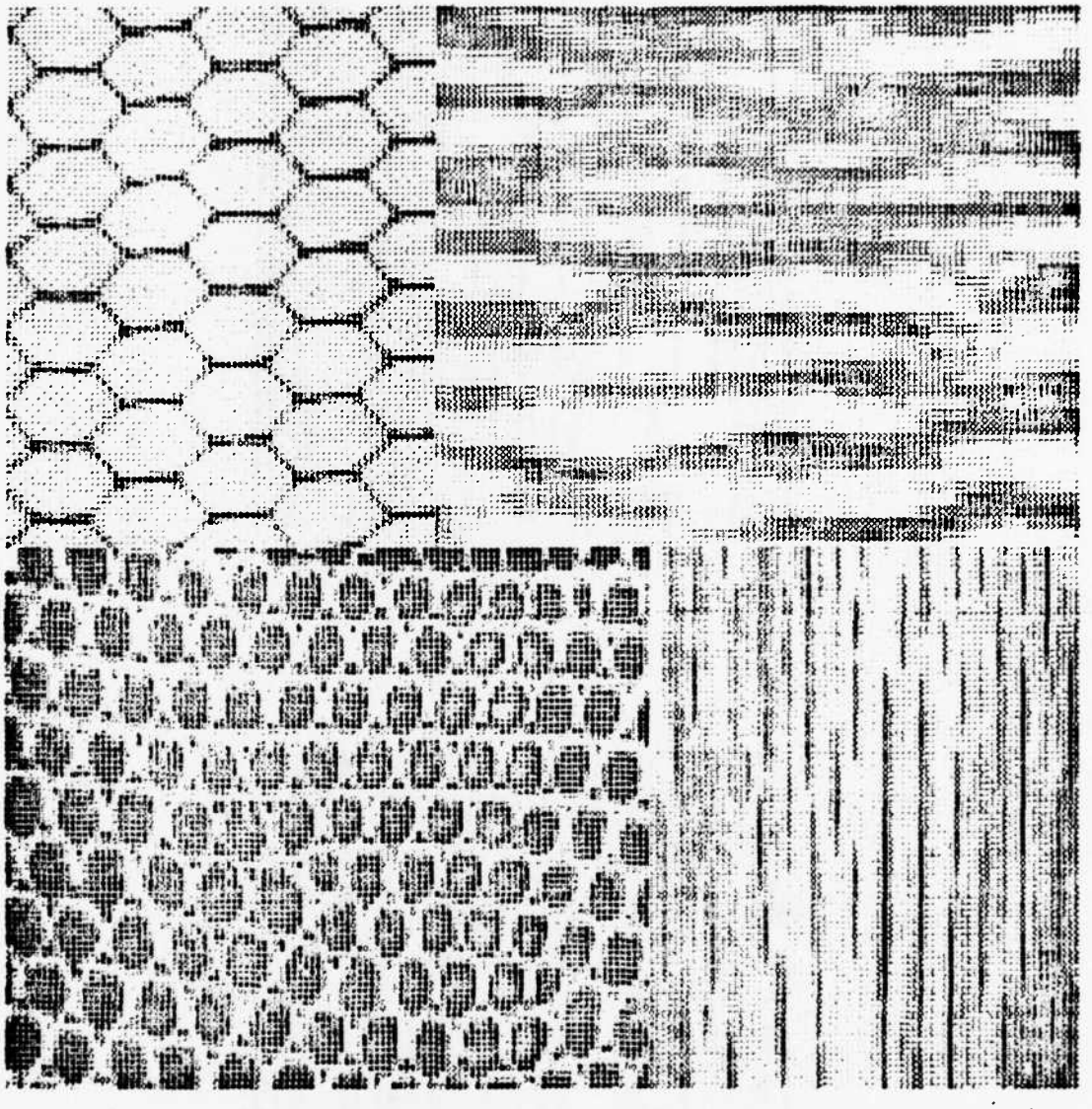


Figure 5.20 Pictorial data for texture discrimination.

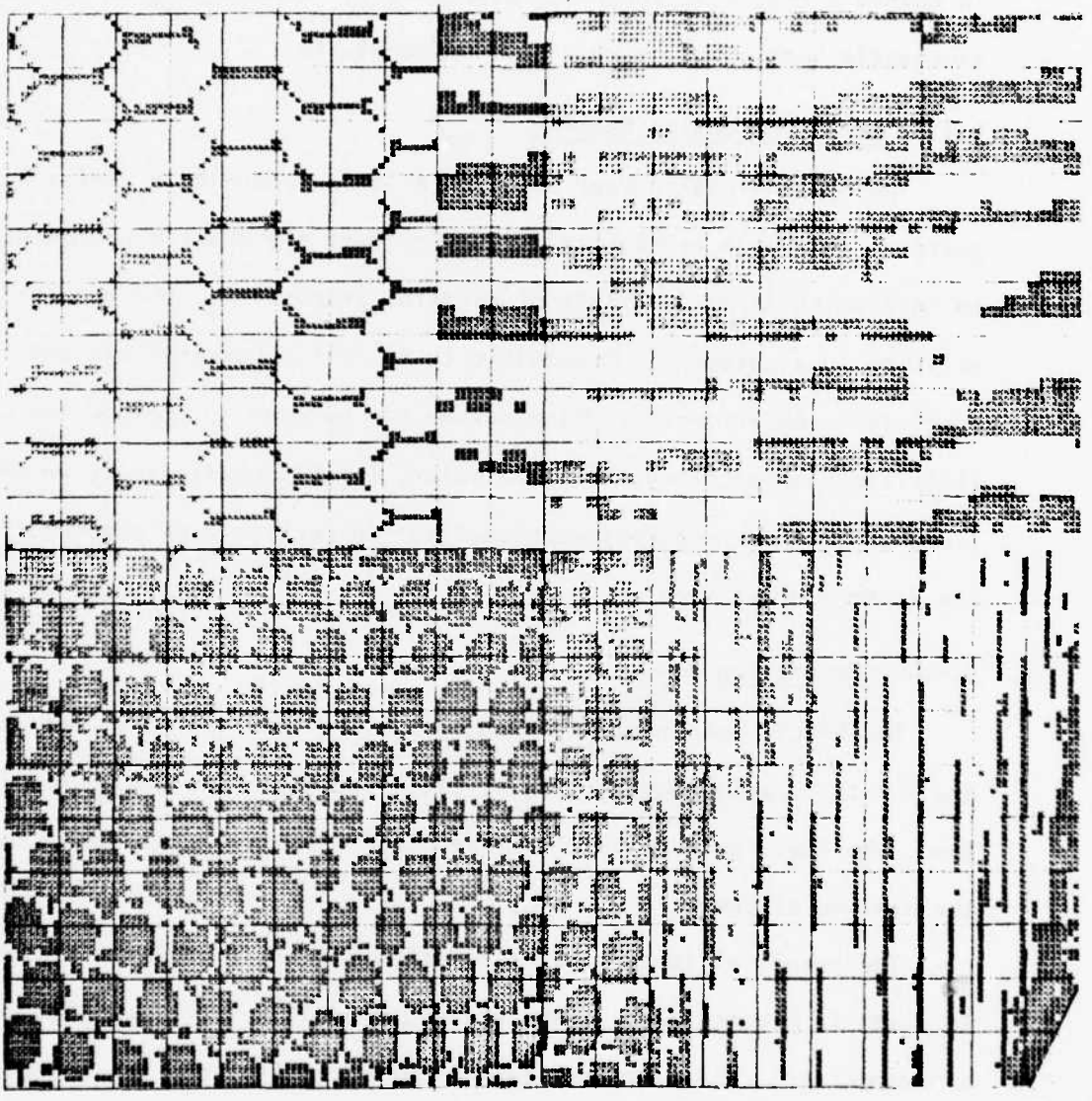


Figure 5.21 Binary picture of Figure 5.20.

G_{22}, G_{34}, G_{38}, and G_{68}, respectively) will be as small as possible. Whereas, each language itself needs to be as general in characterizing each class of texture as possible. Grammar G_{22}, G_{34}, and G_{68} are given in Appendis F-2, F-3, and F-4, respectively. Grammar G_{38} is the non-stochastic version of grammar G_6 in Appendix F-1.

5.4.3 Error-Correcting Parsing

The nonventional parser usually fails to recognize a "noisy" pattern. Although we have tried to construct the discrimination grammars to include as large a variety of patterns as possible, the uncertainty existing in a pattern is impossible to be fully characterized and predicted. An error-correcting parser can be used to improve the classification accuracy. In particular, in this application, we shall use the SPECTA (structure-preserved error-correcting tree automata) as the texture discriminator.

5.4.4 Computation Result

The SPECTA measures the distance between the input tree representation and the texture languages, $L(G_{22})$, $L(G_{34})$, $L(G_{38})$, and $L(G_{68})$ one by one. Then, the input pattern is classified to the texture class which has the minimum distance.

The result of texture discrimination for the picture in Figure 5.21 is given in Figure 5.22. There are 400 windows. Thirty of them are misrecognized. The misrecognition usually results from the unavoidable overlap between two languages or from the reduction of one language to decrease the overlap.

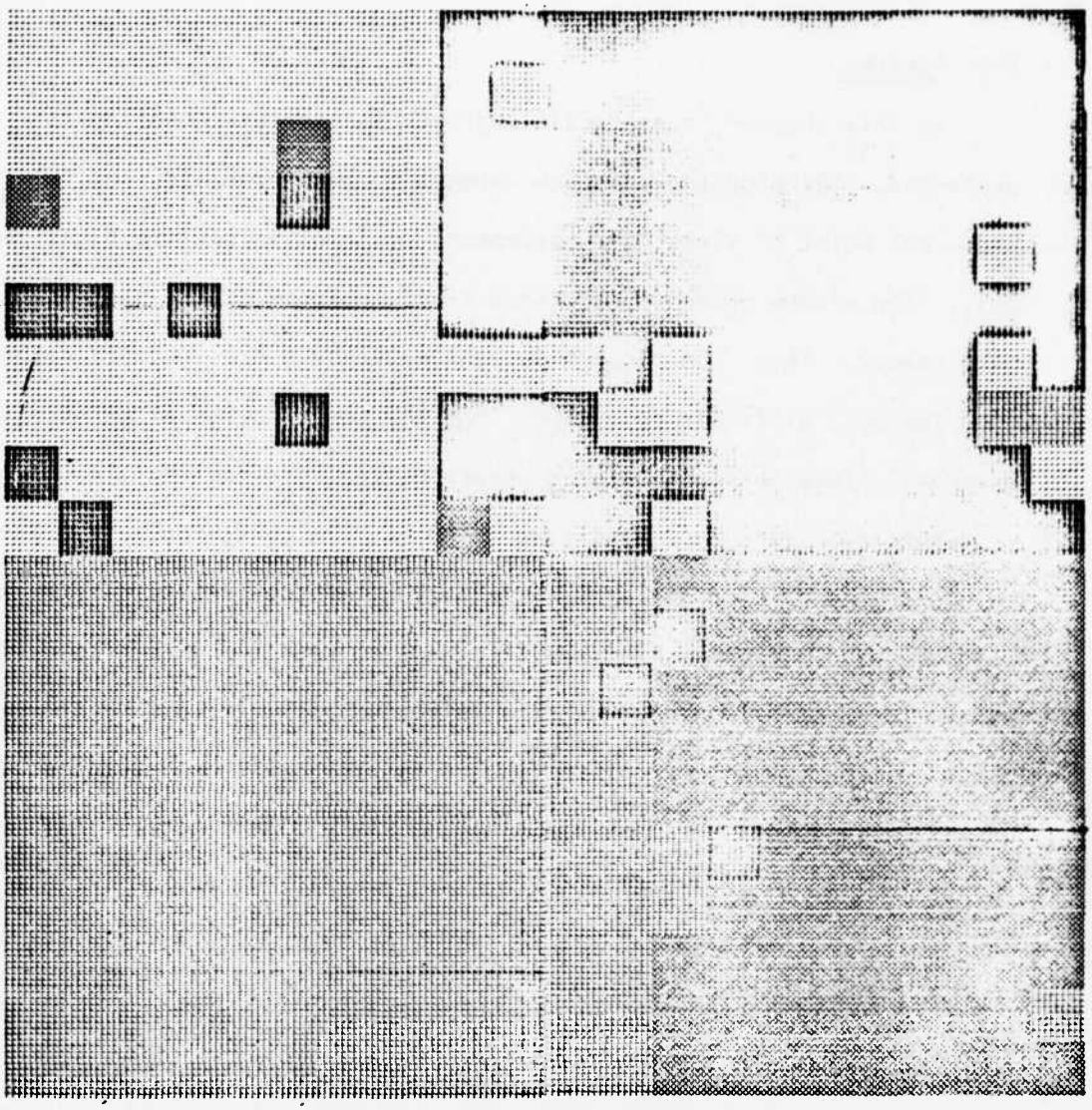
5.5 Remarks

In this chapter, a syntactic approach for texture modeling is presented. The proposed approach appears to be attractive from the practical point of view. The preprocessing involves picture digitization only. The window operation stores a small subframe of the pattern in the main memory. Thus, the process is manageable by a small memory computer.

The most difficult part comes from the construction of an effective grammar. Since no sophisticated preprocessing is used, the linguistic representations are very sensitive to noise. In constructing a grammar, we would like to consider as many variations of the texture pattern as possible. On the other hand, we also need to keep the grammar as simple (as few nonterminals and production rules) as possible to save storage space. Such a compromise often results in a grammar that generates some excessive sentences, but excludes some possible distortions. That is one reason for the necessity of using an error-correcting parser for picture parsing in texture discrimination. The other reason is the uncertainity existing in the picture making the construction of a grammar difficult in order to fit all the possibilities of a texture class.

All the computation examples are programmed in Fortran IV on a PDP-11/45 computer with a 32K core memory. The SPECTA we designed processes all the branches of a tree from the frontiers to the root in parallel, but it should be programmed in series on a general purpose computer. The process can certainly be speeded up by a specially designed processor.

Automatic grammatical inference procedures for tree languages have been recently studied [93]. By combining an inference algorithm with the proposed discrimination procedure, an automation of the entire training and testing process as proposed in Chapter 4 can be implemented.



Color Code

Pattern D34 Pattern D22 Pattern D68 Pattern D38

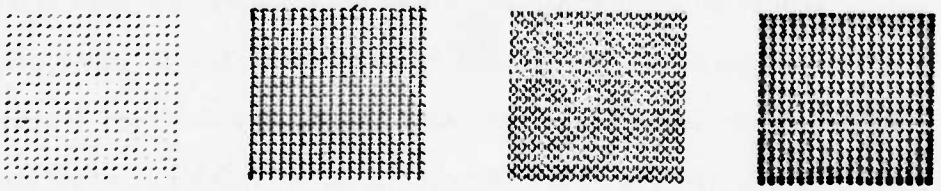


Figure 5.22. Discrimination result.

CHAPTER 6

SUMMARY OF RESULTS AND SUGGESTIONS FOR FURTHER RESEARCH

6.1. Summary of Results and Conclusions

The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. In syntactic pattern recognition, a pattern is described in terms of its subpatterns, primitives and the relations among them. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely, substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. A stochastic deformation model and stochastic error transformation rules are also proposed. In searching for the most likely correction, the formulation of error-correcting parser (ECP) for context-free languages and context-free programmed languages are based on the minimum-distance criterion for non-stochastic model and the maximum-likelihood criterion for stochastic model.

The error-correcting parsing technique for string languages has been extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. Two types of ECTA are proposed; a SPECTA corrects substitution errors only, and a GECTA corrects all five types of errors.

By way of using language transformations, the distance between two sentences - strings, or trees, can be determined. This idea provides a necessary tool for the clustering analysis for syntactic patterns. The algorithms of constructing minimum spanning tree and clustering centers in statistical pattern recognition are extended to syntactic patterns. A definition of distance between a sentence and a language is proposed. Based on this definition, a new clustering procedure is proposed, where a grammar is inferred to characterize a formed cluster, and then updated when a new pattern is assigned to the cluster. An error-correcting parser, in this case, is employed to measure the distance between an input syntactic pattern and a formed cluster, or a language. Therefore, the procedure yields not only the clustering results but also the syntax rules characterizing each cluster.

Finally, using the error-correcting parsing techniques, a real data example on texture modeling and discrimination are presented. In texture modeling, the idea of window operation is used. Texture patterns are divided into fixed size windows. Windowed patterns belonging to the same class of texture are then characterized by a tree grammar. This tree grammar is used for texture synthesis as well as discrimination. However, in texture synthesis, the coherence between windowed patterns is also essential to the overall pattern. It is proposed to use a higher level syntax, for example, another tree grammar, as a monitor for the placement of windowed patterns. Consequently, structural distortion can be simulated by changing the placement of windowed patterns, whereas, local noise can be simulated by using stochastic grammars for characterizing windowed patterns. In texture discrimination, a set of SPECTA, where each is constructed for one class of texture, is used as discriminator. An input windowed pattern is analyzed

by the SPECTA's then classified according to the nearest neighbor rule.

6.2. Suggestions for Further Research

The proposed similarity measure and error-correcting parsing scheme provide a formal model and a useful tool for recognition under uncertainty, for example, in a clustering problem where correct classification for all samples are unknown, or in a noisy environment, such as the analysis of texture patterns. To improve the recognition results, the following problems need further investigation:

(1) To improve the parsing efficiency. The weakness of error-correcting parsing is the need of long computing time. In the case of error-correcting parsing for context-free languages, sequential method has been proposed to reduce the computing time. However, further improvement is still needed. In the texture discrimination example, patterns are divided into fixed-size windows, and then each window is classified by using SPECTA. Both window operation and SPECTA are parallel operations. Parallel processing techniques are expected to be very useful in improving the computation efficiency.

(2) The inference of updated grammars. - The knowledge of classification obtained from using error-correcting parsers is accumulated by way of updating pattern grammars. Different from the existing grammatical inference procedure which often require all the training data be available at the same time, training data are given sequentially in the grammar updating problem. To update a grammar to be simple and effective is a problem for future study.

(3) The inference of a set of weights associated with error transformations. - Using weighted distance can improve the clustering results. The problem of finding a set of appropriate weights from training samples requires further study.

(4) The inference of texture grammars. - Due to the noise and variation of texture patterns, it is cumbersome to construct texture grammars manually. To be more practical, the proposed syntactic model for texture analysis needs an efficient inference procedure. There are two problems to be studied; the first is the inference of stochastic tree grammar for windowed patterns, and the second is to infer placement rules for a texture pattern. In the second problem, the regularity, or the repetition of the basic texture patterns has to be determined. Other problems such as choosing a suitable window size based on density or coarseness of textures, and finding the relations between window size, the effectiveness of texture grammar and parsing efficiency, etc. are also interesting problems to be investigated.

(5) To apply the proposed syntactic texture model to analyze real world data, such as aerial photographs, X-ray images and LANDSAT data, etc.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Fu, K. S., Sequential Methods in Pattern Recognition and Machine Learning, Academic Press, New York, 1968.

2. Andrews, H. C., Introduction to Mathematical Techniques in Pattern Recognition, Wiley, New York, 1972.

3. Duda, R. O. and P. E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1972.

4. Kaneff, S., editor, Picture Language Machines, Academic Press, New York, 1970.

5. Fu. K. S., Syntactic Methods in Pattern Recognition, Academic Press, 1974.

6. Fu, K. S., "Stochastic Automata, Stochastic Language and Pattern Recognition," Journal of Cybernetics, Vol. 1, No. 3, 1971.

7. Fu, K. S. and T. Huang, "Stochastic Grammars and Languages," International Journal of Computer and Information Sciences, Vol. 1, No. 2, June 1972.

8. Fu, K. S.,"Stochastic Language for Picture Analysis," Computer Graphics and Image Processing, Vol. 2, pp. 433-453, 1973.

9. Page, C. and A. Fillpski, "Discriminant Grammars, an Alternative to Parsing for Pattern Classification," Proceedings 1977 IEEE Workshop on Picture Data Description and Management, April 20-22, Chicago, IL.

10. Persoon, E. and K. S. Fu, "Sequential Classification of Strings Generated by SCFG's, Int. Journal of Computing and Information Sciences, Vol. 4, No. 3, September, 1975.

11. Ali, F. and T. Pavlidis, "Description and Recognition of Handwritten Numerals," Proceedings, 1977 IEEE Workshop on Picture Data Description and Management, April 20-22, Chicago, IL.

12. Fu, K. S. and B. K. Bhargave, "Tree Systems for Syntactic Pattern Recognition," IEEE Trans. on Computers, Vol. C-22, No. 12, pp. 1087-1099, December, 1973.

13. Aho, A. V. and T. G. Peterson, "A Minimum Distance Error-Correcting Parser for Context-Free Languages," SIAM J. Comput., Vol. 4, December, 1972.

14. Aho, A. V. and J. D. Ullman, The Theory of Parsing, Translation, and Compiling, Vol. 1, Prentice Hall, 1972.

15. Irons, E. T., "An Error-Correcting Parse Algorithm," Comm. ACM, Vol. 6, No. 11, November, 1963.

16. Gries, D., Compiler Construction for Digital Computers, Wiley, 1971.

17. Wirth, N., "A Programming Language for the 360 Computers," J. ACM, Vol. 15, January 1968.

18. Leinius, R. P., "Error Detection and Recovery for Syntax Directed Compiler System," Ph.D. Thesis, University of Wisconsin, 1970.

19. Graham, S. L. and S. P. Rhodes, "Practical Syntactic Error Recovery," Comm. ACM, Vol. 18, No. 11, November, 1975.

20. Levy, J. P., "Automatic Correction of Syntax Errors in Programming Languages," Acta Informatica 4, pp. 271-292, 1975.

21. James E. G. and D. P. Partridge, "Adaptive Correction of Program Statements," Comm. ACM, January, 1973.

22. Horning, J. J., What the Compiler Should Tell the User, Lecture Notes in Computer Science, Vol. 21, Ed. by G. Goos and J. Harmanis, Springer-Verlag, 1974.

23. Aho, A. V. and J. D. Ullman, "Error Detection in Precedence Parsers," Math. Systems Theory, Vol. 7, No. 2, 1973.

24. Lyon, G., "Syntax-Directed Least-Error Analysis for Context-Free Language: A Practical Approach," Comm. ACM, Vol. 17, No. 1, January, 1974.

25. Morgan, H. L., "Spelling Correction in System Programs," Comm. ACM, Vol. 13, February, 1970.

26. Wagner, R. A. and M. J. Fisher, "The String to String Correction Problem," J. ACM, Vol. 21, No. 1, January 1974.

27. Wagner, R. A., "Order-n Correction for Regular Languages," C. ACM, Vol. 17, No. 5, May, 1974.

28. Thomason, M. G., "Errors on Regular Language," IEEE Trans. on Computer, Vol. C-23, No. 6, January 1974.

29. Souza, C. R. and R. A. Scholtz, "Probabilistic Generation, Transmission and Syntactical Decoding of Context-Free Programmed Language," Proceedings 8th Annual Allerton Conference, Monticello, Illinois, October 1970.

30. Bahl, L. R. and F. Jelinek, "Decoding for Channels with Insertions, Deletions, and Substitutions with Applications to Speech Recognition," IEEE Trans. on Infor. Theory, Vol. IT-21, No. 4, July, 1975.

31. Fung, L. W. and K. S. Fu, "Stochastic Syntactic Decoding for Pattern Classification," IEEE Trans. on Computers, Vol. C-24, No. 6, July, 1975.

32. Thompson, R. A., "Language Correction Using Probabilistic Grammars," IEEE Trans. on Computers, Vol. C-25, No. 3, March, 1976.

33. Thomason, M. G. and R. C. Gonzalez, "Syntactic Recognition of Imperfectly Specified Patterns," IEEE Trans. on Computers, January, 1975.

34. Earley, J. "An Efficient Context-free Parsing Algorithm," C. ACM, Vol. 13, pp. 94-102, 1970.

35. Ledley, R. S. et al., In Optical and Electro-Optical Information Processing, ed. by J. T. Tippett, et al., MIT Press, Cambridge, Mass. p. 591, 1965.

36. Guzman, A., "Decomposition of a Visual Scene into Three-Dimensional Bodies," FJCC Proceedings, Vol. 33, pp. 291-304, 1968.

37. Narasimhan, R. and V. S. N. Reddy, "A Syntax-Aided Recognition Scheme for Handprinted English Letters," Pattern Recognition, Vol. 3, pp. 345-361, 1971.

38. Moayer, B. and K. S. Fu, "Syntactic Pattern Recognition of Fingerprints," Purdue University, TR-EE 74-36, December, 1974.

39. Li, R. Y. and K. S. Fu, "Tree System Approach for LANDSAT Data Interpretation," Purdue-LARS Symposium on Machine Processing of Remotely Sensed Data, West Lafayette, Indiana, June, 1976.

40. Pavlidis, T., "Linguistic Analysis of Waveforms," In Software Engineering, Vol. 4, J. T. Tou, ed. Academic Press, pp. 203-225, 1971.

41. Horowitz, S. L., "A Syntactic Algorithm for Peak Detection in Waveforms with Applications to Cardiograph," C ACM, Vol. 18, No. 5, May, 1975.

42. Ehrich, R. W. and J. P. Foith, "Representation of Random Waveforms by Relational Trees," IEEE Trans. on Computers, Vol. C-25, No. 7, July, 1976.

43. Stockman, G., L. Kanal, and M. C. Kyle, "Structural Pattern Recognition of Carotid Pulse Waves Using a General Waveform Parsing System," C. ACM, Vol. 19, No. 12, December, 1976.

44. Feder, J., "Plex Language," Information Sciences, Vol. 3, pp. 225-241, 1971.

45. Pratt, T. W., "Pair Grammars, Graph Languages and String-to-Graph Translations," J. of Computer and System Sciences, Vol. 5, 1971.

46. Fordon, W. A. and K. S. Fu, "Computer-Aided Differential Diagnosis of Hypertension," Purdue University TR-EE 76-27, August, 1976.

47. Huang, T. and K. S. Fu, "Stochastic Syntactic Analysis for Programmed Grammars and Syntactic Pattern Recognition," Computer Graphics and Image Processing, Vol. 1, pp. 257-283, 1972.

48. Swain, P. H. and K. S. Fu, "Stochastic Programmed Grammars for Syntactic Pattern Recognition," Pattern Recognition, Vol. 4, 1972.

49. Chou, S. M. and K. S. Fu, "Transition Networks for Pattern Recognition," Purdue University, TR-EE 75-39, December, 1975.

50. Fu, K. S. and B. K. Bhargava, "Tree Systems for Syntactic Pattern Recognition," IEEE Trans. on Computers, Vol. C-22, No. 12, pp. 1087-1099, December, 1973.

51. Pavlidis, T., "Linear and Context-Free Graph Grammars," J. ACM, Vol. 19, No. 1, pp. 11-22, January, 1972.

52. Pfaltz, J. L. and A. Rosenfeld, "Web Grammar," Proceeding 1st International Joint Conference on Artificial Intelligence, Washington, D.C., 1969.

53. Brayer, J. M. and K. S. Fu, "Web Grammars and Their Application to Pattern Recognition," Purdue University, TR-EE 75-1, December, 1975.

54. Milgram, D. L. and A. Rosenfeld, "Array Automata and Array Grammars," IFIP Congress 71, North-Holland, Amsterdam, 1971.

55. Siromoney, G., Siromoney, R., and K. Krithivasan, "Abstract Families of Matrices and Picture Langauges," Computer Graphic and Image Processing, Vol. 1, pp. 284-307, 1972.

56. Wang, P. S., "Sequential/Parallel Matrix Array Languages," Journal of Cybernetics, Vol. 5, No. 4, October-December, 1975.

57. Tanaka, E. and K. S. Fu, "Error-Correcting Parser for Formal Language," Purdue University, TR-EE 76-7, March, 1976.

58. Tai, K. C., "Syntactic Error Correction in Programming Languages," Ph.D. Thesis, Department of Computer Science, Cornell University.

59. Hopcroft, J. E. and J. D. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, 1969.

60. Rosenkrantz, D. J., "Programmed Grammars - a New Device for Generating Formal Languages," Ph.D. Thesis, Columbia University, 1976.

61. Tanaka, E. and T. Kasai, "Synchronization and Substitution Error-Correcting Codes for the Levenshtein Metric," IEEE Trans. on Information Theory, Vol. IT-22, No. 2, March, 1976.

62. Fu. K. S., "On Syntactic Pattern Recognition and Stochastic Languages," Frontiers of Pattern Recognition, ed. S. Watanabe, Academic Press, 1972.

63. Huang, T. and K. S. Fu, "On Stochastic Context-Free Language," Information Sciences, Vol. 3, pp. 201-224, 1971.

64. Lee, H. C. and K. S. Fu, "A Stochastic Syntax Analysis Procedure and Its Application to Pattern Classification," IEEE Trans. on Computers, Vol. C-21, No. 7, 1972.

65. Lee, H. C. and K. S. Fu, "A Syntactic Pattern Recognition System with Learning Capability," Information Systems, Coins IV, ed. by J. T. Tou, Plenum Press, 1974.

66. Brainerd, W. S., "Tree Generating Regular Systems," Inf. and Control, 14, pp. 217-231, 1969.

67. Doner, J., "Tree Acceptors and Some of Their Applications," J. of Compt. and Sys. Sci., 4, pp. 406-451, 1970.

68. Rounds, W. C., "Mappings and Grammars on Trees," Math. System Theory, Vol. 4, No. 3, 257-286, 1970.

69. Thatcher, J. W., "Characterizing Derivation Trees of Context-Free Grammars Through a Generation of Finite Automata Theory," J. of Comp. and Syst. Science, 1, 1967, 317-322.

70. Thatcher, J. W., "Tree Automata: An Informal Survey," Currents in the Theory of Computing, ed. by A. V. Aho, Prentic-Hall, 1973.

71. Bhargava, B. K. and K. S. Fu, "Application of Tree System Approach to Classification of Bubble Chamber Photographs," Purdue University, TR-EE 72-30, November, 1972.

72. Bhargava, B. K. and K. S. Fu, "Stochastic Tree Systems for Syntactic Pattern Recognition," Proceedings, Allerton Conference on Circuit and System Theory, 1974.

73. Todd, W. J. and M. F. Baumgardner, "Land Use Classification of Marion County, Indiana,Data," LARS Information Note 101673, LARS, Purdue University, 1973.

74. Fu, K. S. and T. L. Booth, "Grammatical Inference: Introduction and Survey - Part I and Part II," IEEE Trans. Sys. Man and Cybernetics, Vol. SMC-5, January and July, 1975.

75. Ellis, C. A., "Probabilistic Tree Automata," Information and Control, Vol. 19, pp. 401-416, 1971.

76. Boorman, S. A. and D. C. Oliver, "Metrics on Spaces of Finite Trees," J. of Math. Psychology, 10, 26-59, 1973.

77. Freeman, H. "On the Encoding of Arbitrary Geometric Configurations," IEEE Trans. Electron. Comput. EC-10, pp. 260-268, 1961.

78. Lipkin, B. S. and A. Rosenfeld, Eds., Picture Processing and Psychopictorics, New York: Academic Press, Inc., 1970, pp. 289-381.

79. Zucker, S. W., "Toward a Model of Texture," Computer Graphics and Image Processing, 5, 1976, pp. 190-202.

80. Haralick, R. M., K. Shanmugam and I. Dinstein, "Textural Features for Image Classification," IEEE Trans. on SMC, Vol. SMC-3, No. 6, Nov. 1973.

81. Weszka, J. S., C. R. Dyer and A. Rosenfeld, "A Comparative Study of Texture Measures for Terrain Classification," IEEE Trans. on SMC, Vol. SMC-6, No. 4, April, 1976.

82. Conners, R. W. and C. A. Harlow, "Some Theoretical Considerations Concerning Texture Analysis of Radiographic Images," Proceedings of the 1976 IEEE Conference on Decision and Control, Clearwater Beach, FL, December 1-3, 1976.

83. Sutton, R. N. and E. L. Hall, "Texture Measures for Automatic Classification of Pulmonary Disease," IEEE Trans. on Computers, Vol. C-21, No. 7, July 1972.

84. Bacus, J. W. and E. E. Gose, "Leukocyte Pattern Recognition," IEEE Trans. on SMC, Vol. SMC-2, No. 4, pp. 513-526, September, 1972.

85. Young, I. T., "The Classification of White Blood Cells," IEEE Trans. on Biomed. Eng., Vol. BME-19, No. 4, pp. 291-298, July 1972.

86. Mui, J. K., J. W. Bacus and K. S. Fu, "A Scene Segmentation Technique for Microscopic Cell Images," Proceedings of the Third International Joint Conference on Pattern Recognition, Caronado, CA, November 8-11, 1976.

87. Weszka, J. S. and A. Rosenfeld, "An Application of Texture Analysis to Materials Inspection," Pattern Recognition, Vol. 8, No. 4, October, 1976.

88. McCormick, B. H. and S. N. Jayaramamurthy, "A Decision Theory Method for the Analysis of Texture," Int. J. of Compt. and Inf. Sci., Vol. 4, No. 1, 1975.

89. McCormick, B. H. and S. N. Jayaramamurthy, "Time Series Model for Texture Synthesis," Int. J. of Compt. and Inf. Sci., Vol. 3, No. 4, 1974.

90. Schachter, B. J., A. Rosenfeld and L. S. Davis, "Random Mosaic Models for Textures," Computer Science Technical Report Series, University of Maryland, 1976.

91. Carlucci, L., "A Formal System for Texture Languages," Pattern Recognition, Vol. 4, pp. 53-72, 1972.

92. Brodatz, P., Textures, Dover Publications, New York, 1966.

93. Brayer, J. M. and K. S. Fu, "A Note on K-Tail Method of Tree Grammar Inference," IEEE Trans. on SMC, April 1977.

94. DiDay, E. and J. C. Simon, "Clustering Analysis," Digital Pattern Recognition, ed. K. S. Fu, Springer-Verlag, 1976.

95. Shaw, A. C., "A Formal Picture Description Scheme as a Basis for Picture Processing Systems," Inf. and Cont., 14, pp. 9-52, 1969.

96. Bierman, A. W. and J. A. Feldman, "A Survey of Results in Grammatical Inference," Frontiers of Pattern Recognition, ed. by S. Watanabe, Academic Press, 1972.

97. Levenshtein, V. i., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," Sov. Phy. Dokl., Vol. 10, No. 8, pp. 707-710, Feb., 1966.

98. Woods, W. A., "Transition Network Grammars for Natural Language Analysis," Comm. ACM, Vol. 13, No. 10, Oct. 1970.

99. Vigna, P. D., and C. Ghezzi, "Data Structure and Graph Grammars," Tech. Rept., Instituto di Elettrotecnica ed Elettronica Politecnico di Milano, Milano, Italy, 1976.

100. Bonczek, R. H., and A. B. Whinston, "Picture Processing and Automatic Data Base Design," Computer Graphics and Image Processing, Vol. 5, Dec., 1976.

101. Fu, K. S., Applications of Syntactic Pattern Recognition, Communication and Cybernetics Series, Vol. 14, Springer-Verlag, 1977.

APPENDICIES

APPENDIX A

PROOF OF THE CONSISTENCY OF THE MULTIPLE ERROR MODEL

Assume that deformation probabilities on terminal $a \in \Sigma$ is consistent on a single error model; i.e., equation (2.1) in Section 2.4 is satisfied. By summing over all the cases in equation (2.2) we have,

$$\sum_{\alpha\in\Sigma}{}^{*}\, q(\alpha|a) = \{q_D(a)\} + \{\sum_{b\in\Sigma}[q_S(b|a) + q_1(b|a)q_D(a)]\}$$

$$+ \{\sum_{\ell=2}^{\infty}[\prod_{j=1}^{\ell-1}(\sum_{b\in\Sigma} q_1(b|a))][\sum_{b\in\Sigma}(q_S(b|a)$$

$$+ q_1(b|a)q_D(a))]\} . \qquad \text{(A1)}$$

From equation (2.7), the first three terms of (A1) can be reduced to

$$[1-\sum_{b\in\Sigma} q_1(b|a)] + [\sum_{b\in\Sigma} q_1(b|a)]q_D(a) \qquad \text{(A2)}$$

and the fourth and fifth term of (A1) can be reduced to

$$\sum_{\ell=2}^{\infty}[\prod_{j=1}^{\ell-1}(\sum_{b\in\Sigma} q_1(b|a))][1-q_D(a) - \sum_{b\in\Sigma} q_1(b|a)$$

$$+ \sum_{b\in\Sigma} q_1(b|a)q_D(a)] . \qquad \text{(A3)}$$

Then,

$$\sum_{\alpha \in \Sigma} q(\alpha|a) = (A2) + (A3)$$

$$= 1-q_1(a)\ (1-q_D(a))$$

$$+ \sum_{\ell=2}^{\infty} [\prod_{j=1}^{\ell-1} q_1(a)][(1-q_1(a))$$

$$(1-q_D(a))\ ] = 1$$

where $q_1(a) = \sum_{b\ \varepsilon\Sigma} q_1(b\ |a)$

APPENDIX B

SEQUENTIAL CLASSIFICATION ALGORITHMS

Let $\hat{e}$ be a parameter, $0 \leq \hat{e} \leq 1$.

Algorithm 3. Decision Algorithm

Input: String $a_1a_2\ldots a_n$.

Output: C_ℓ, $1 \leq \ell \leq k$.

Method:

Step 1. Set j=0 and compute $r=1 - \max_i P(C_i)$. If $r < \hat{e}$, stop and assign class C_ℓ, where $P(C_\ell) = \max_i P(C_i)$. If $r \geq \hat{e}$, set j=1 and go to Step 2.

Step 2. Parse the jth input symbol by SPA.

Step 3. If Step 2 receives parsing failure flag from all pattern grammars, stop and assign class C_ℓ, where $p(C_\ell|a_1a_2\ldots a_{j-1}\alpha) = \max_i p(C_i|a_1a_2\ldots a_{j-1}\alpha)$, otherwise go to Step 4.

Step 4. If j=n, stop and classify to C_ℓ, where $p(C_\ell|a_1a_2\ldots a_n) = \max_i p(C_i|a_1a_2\ldots a_n)$. If $j\neq n$, compute $r=1 - \max_i p(C_i|a_1a_2\ldots a_j\alpha)$, go to Step 5.

Step 5. If $r < \hat{e}$, stop and assign C_ℓ where $p(C_\ell|a_1a_2\ldots a_j\alpha) = \max_i p(C_i|a_1a_2\ldots a_j\alpha)$, otherwise, If j=n then stop, otherwise, set j=j+1. Go to Step 2.

Algorithm 4. Sequential Parsing Algorithm

Input: A SCFG $G_I = (N_I, \Sigma_I, P_I, S_I)$ and an input string $a_1 a_2 \ldots a_j$. $1 \leq j \leq n$.

Output: $p(a_1 a_2 \ldots a_j | C_I)$ and $p(a_1 a_2 \ldots a_j \alpha | C_I)$.

Method:

Step 1. Set j=0, add item $[Z \rightarrow \cdot S_I, 0, 1, 1]$ to I_j.

Step 2. (a) if $[A \rightarrow \alpha \cdot B\beta, i, p, r]$ is in I_j, and $B \xrightarrow{q} \gamma$ is in P_I, add item $[B \rightarrow \cdot \gamma, j, q, 0]$ to I_j.

(b) if $[A \rightarrow \alpha \cdot, i, p_1, r]$ is in I_j, for all item in I_i of the form $[B \rightarrow \beta \cdot A\gamma, k, p_2, s)$, add item $[B \rightarrow \beta A \cdot \gamma, k, p_3, 0]$ to I_j where $p_3 = p_1 p_2$, unless an item of the form $[B \rightarrow \beta A \cdot \gamma, k, q, t]$ is already in I_j. If this is the case, set $q = q + p_1 p_2$.

Step 3. (a) For each item I_j of the form $[A \rightarrow \alpha \cdot B\beta, i, q, r]$, where $i < j$, find in I_i all the itmes of the form $[C \rightarrow \mu \cdot A\gamma, k, t, s]$. Suppose that there are m such items with values for s equals to $s_1 s_2 \ldots s_m$, respectively, set $r = q(s_1 + s_2 + \ldots + s_m)$.

(b) Locate all items in I_j of the from $[A \rightarrow \alpha \cdot B\beta, j, q, r]$. If there are n such items, number these items such that kth of them is denoted as $[A_k \rightarrow \alpha_k \cdot B_k \beta_k, j, q_k, r_k]$. For the kth item, locate in I_j all items of the form $[C \rightarrow \mu \cdot A_k \gamma, i, t, s]$. if there are m_k such items, we denote the values for s in these items as $s_1 s_2 \ldots s_{m_k}$ then $r_k = q_k(s_1 + s_2 + \ldots + s_{m_k})$, k =1,2...n. Note that either s_i is determined by Step 3(a) already, or

s_1 is not known and is one of the unknowns r_j ($j = 1,2...n$). This given n linear equations in n unknowns from which $r_1, r_2 \ldots r_n$ can be determined.

Step 4. If j=0, to go Step 7; otherwise, go to Step 5.

Step 5. Compute $p(a_1a_2\ldots a_j|G_1)$ as follow, if an item of the from $[Z \rightarrow S_1 \cdot , 0,p,r]$ is in i_j, set $p(a_1a_2\ldots a_j|G_1) = p$, otherwise $p(a_1a_2\ldots a_j|G_1) = 0$.

Step 6. Applying $p(a_1a_2\ldots a_j|G_1)$ and $p(a_1a_2\ldots a_j\alpha|G_1)$ obtained from Step 5 and Step 8 to Algorithm 3, if it is decided to continue, go to Step 7; otherwise, stop and classify.

Step 7. Set j=j+1. For each item in i_{j-1} of the form $[A \rightarrow \alpha \cdot a_j\beta,i,q,r]$, add item $[A \rightarrow \alpha a_j \cdot \beta,i,q,0]$ to i_j, go to Step 8. If no item of the form $[A \rightarrow \alpha \cdot a_j\beta,i,q,r]$ in I_{j-1} exists, set parsing failure flag and $p(a_1a_2\ldots a_j\alpha|G_1) = 0$, $p(a_1a_2\ldots a_j|G_1) = 0$, stop.

Step 8. Locate the items which are added to I_j by Step 7. Suppose that there are n such items, number the mth of them as $[A \rightarrow \alpha a_j \cdot \beta,i_m,p_m,r_m]$. Find all items in I_{i_m} of the form $[B \rightarrow \gamma \cdot A\delta,k,q,s]$ and suppose that there are ℓ_m such items with parameters s denoting as $s_1,s_2\ldots s_{\ell_m}$, then $p(a_1a_2\ldots a_j|G_1) + p(a_1a_2\ldots a_j\alpha|G_1) = \sum_{m=1}^{K_{\ell_m}} p_m \sum_{i=i}^{\ell_m} s_i$, go to Step 2.

APPENDIX C

HIGHWAY GRAMMAR

G_H = (V,P,r,S) over $<\Sigma,r>$ where,

V = $\{S, H_0, X_0, A_{1\cdots8}, B_{1\cdots8}, C_{1\cdots3}, D_{1\cdots3}, E_{1\cdots7}, F_{1\cdots7}, H_{1\cdots7}, M_{1\cdots7}, h, b, \$\}$

r($) = {1}

r(h) = {0,1,3}

r(b) = {0,1,3}

P:

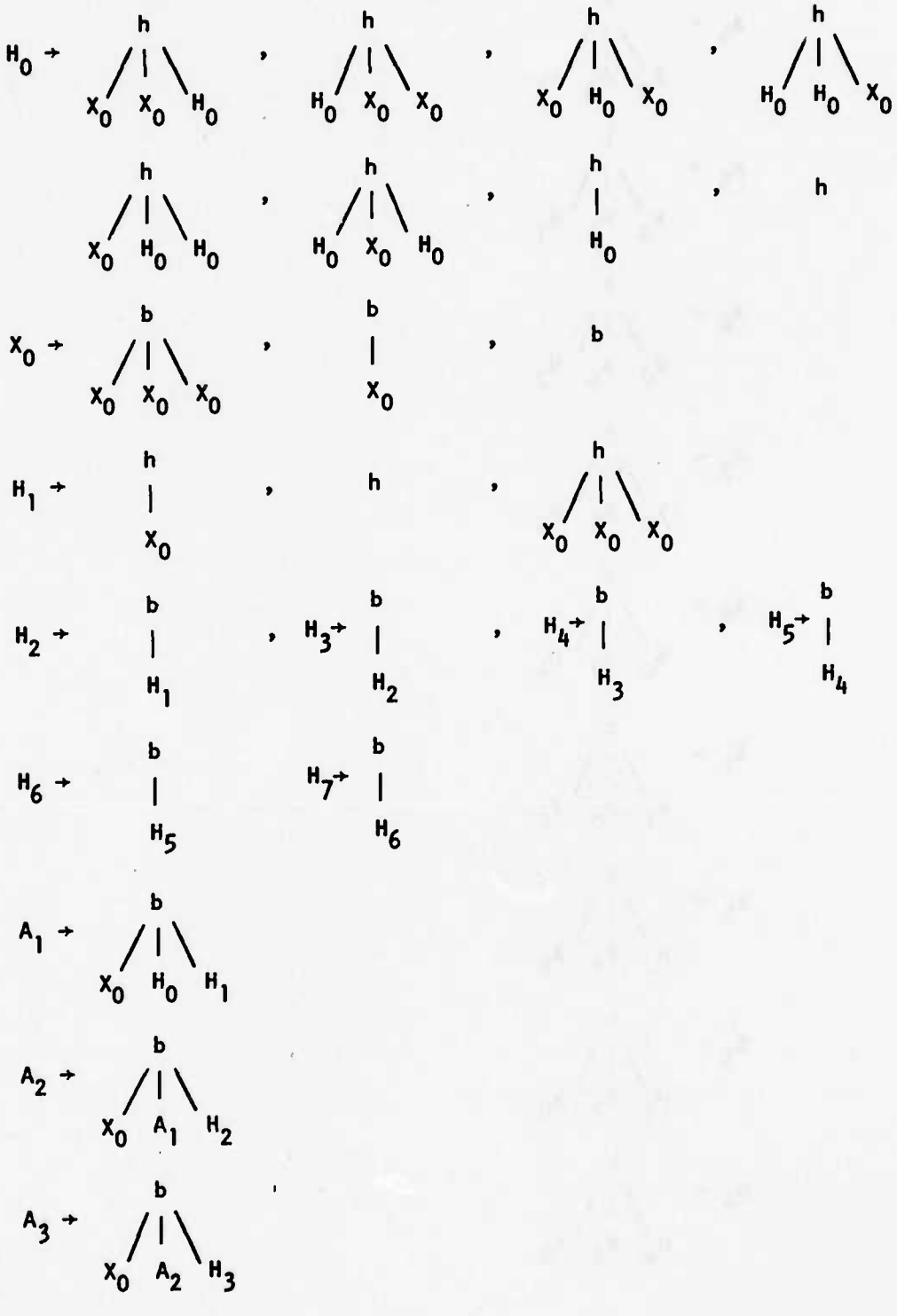
$$S \rightarrow \begin{matrix}\$\\|\\A_i\end{matrix} , \begin{matrix}\$\\|\\B_i\end{matrix} , \quad i = 1\ldots8$$

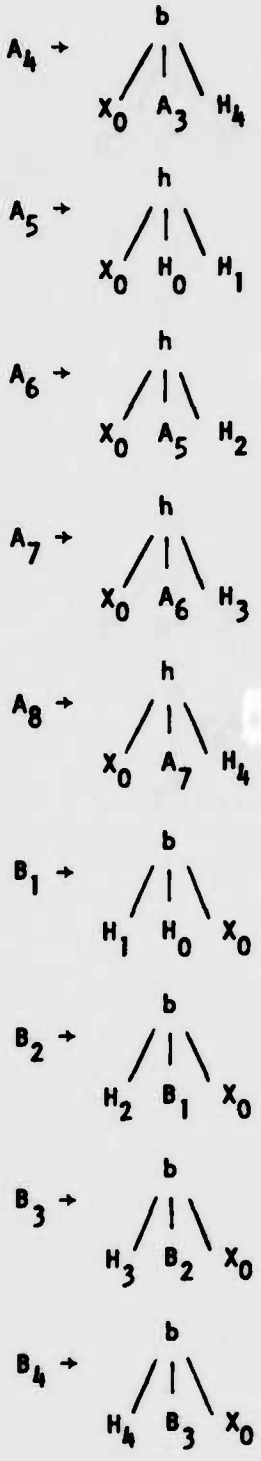
$$S \rightarrow \begin{matrix}\$\\|\\C_i\end{matrix} , \begin{matrix}\$\\|\\D_i\end{matrix} , \quad i = 1\ldots3$$

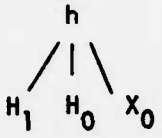
$$S \rightarrow \begin{matrix}\$\\|\\E_i\end{matrix} , \begin{matrix}\$\\|\\F_i\end{matrix} , \quad i = 1\ldots7$$

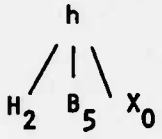
$$S \rightarrow \begin{matrix}\$\\|\\M_i\end{matrix} , \quad i = 1\ldots4$$

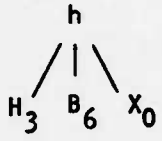
$$S \rightarrow \begin{matrix}\$\\|\\H_0\end{matrix} , \begin{matrix}\$\\|\\X_0\end{matrix} ,$$

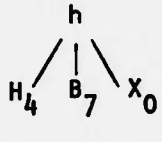

H0 → h (X0 X0 H0) , h (H0 X0 X0) , h (X0 H0 X0) , h (H0 H0 X0)
h (X0 H0 H0) , h (H0 X0 H0) , h (H0) , h
X0 → b (X0 X0 X0) , b (X0) , b
H1 → h (X0) , h , h (X0 X0 X0)
H2 → b (H1) , H3 → b (H2) , H4 → b (H3) , H5 → b (H4)
H6 → b (H5) , H7 → b (H6)
A1 → b (X0 H0 H1)
A2 → b (X0 A1 H2)
A3 → b (X0 A2 H3)

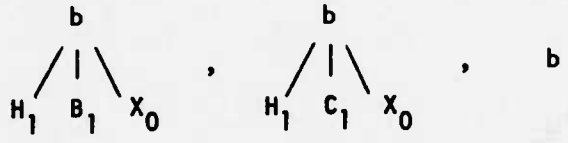

A4 → b X0 A3 H4
A5 → h X0 H0 H1
A6 → h X0 A5 H2
A7 → h X0 A6 H3
A8 → h X0 A7 H4
B1 → b H1 H0 X0
B2 → b H2 B1 X0
B3 → b H3 B2 X0
B4 → b H4 B3 X0

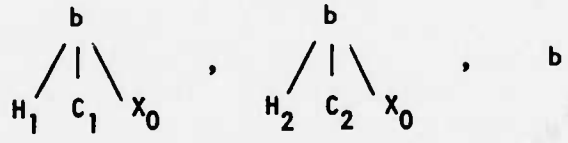
$B_5 \rightarrow$ 

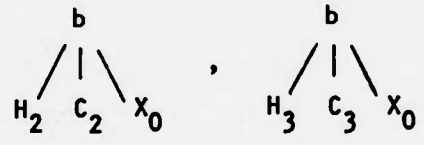
$B_6 \rightarrow$ 

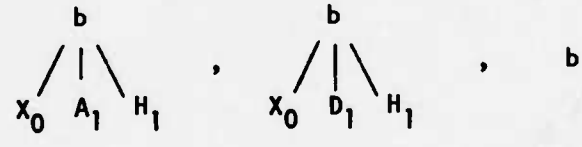
$B_7 \rightarrow$ 

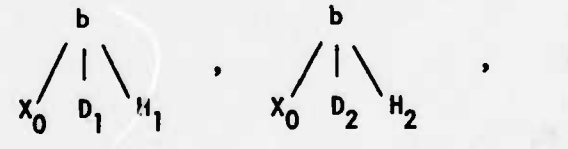
$B_8 \rightarrow$ 

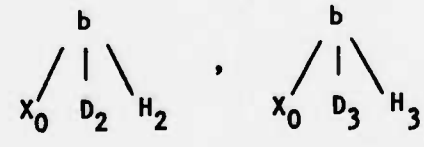
$C_1 \rightarrow$ 

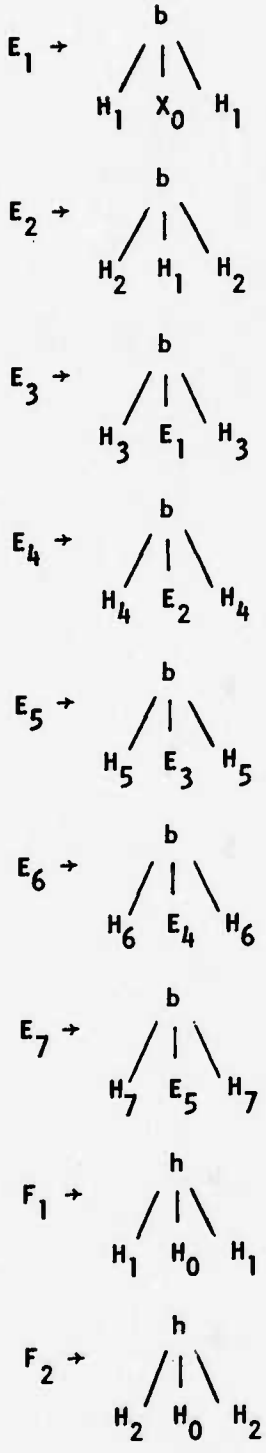
$C_2 \rightarrow$ 

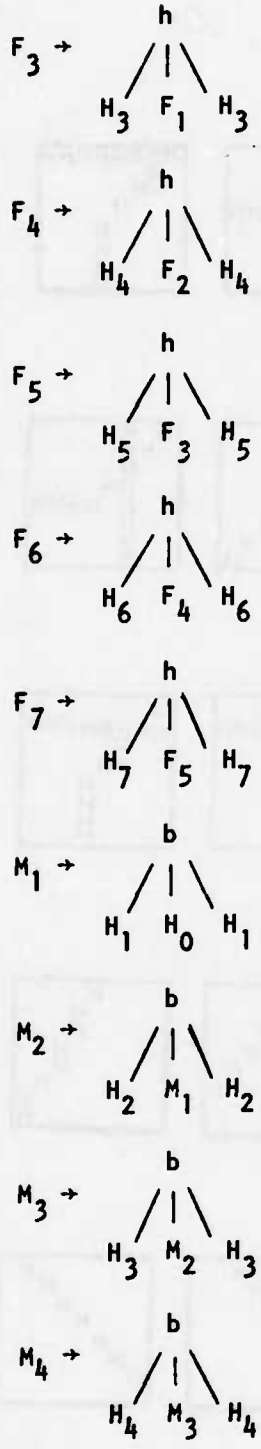
$C_3 \rightarrow$ 

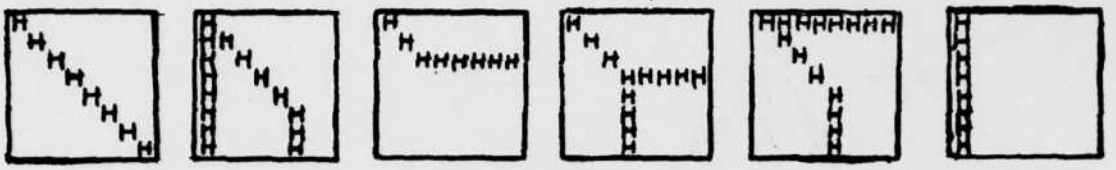
$D_1 \rightarrow$ 

$D_2 \rightarrow$ 

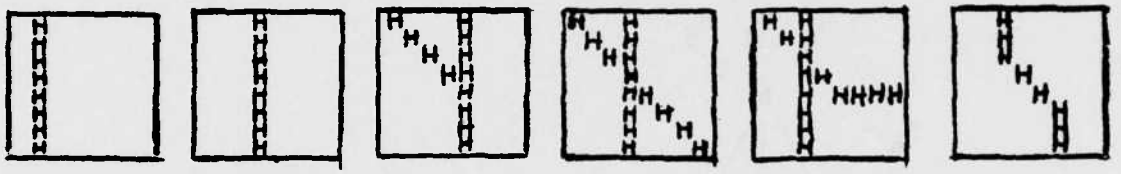
$D_3 \rightarrow$ 


E1 → b H1 X0 H1
E2 → b H2 H1 H2
E3 → b H3 E1 H3
E4 → b H4 E2 H4
E5 → b H5 E3 H5
E6 → b H6 E4 H6
E7 → b H7 E5 H7
F1 → h H1 H0 H1
F2 → h H2 H0 H2

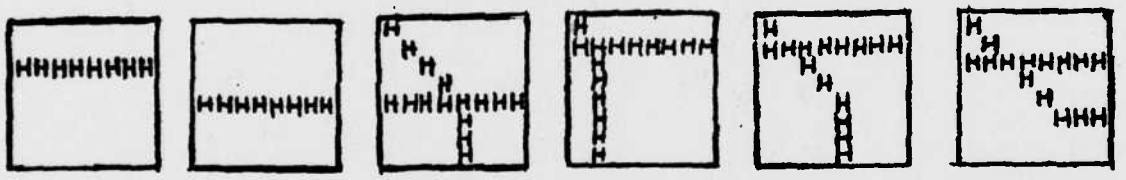

h
F3 →
H3 F1 H3
h
F4 →
H4 F2 H4
h
F5 →
H5 F3 H5
h
F6 →
H6 F4 H6
h
F7 →
H7 F5 H7
b
M1 →
H1 H0 H1
b
M2 →
H2 M1 H2
b
M3 →
H3 M2 H3
b
M4 →
H4 M3 H4



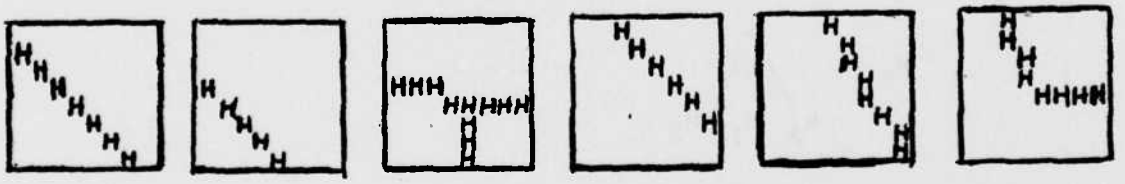
(a) Group H_0



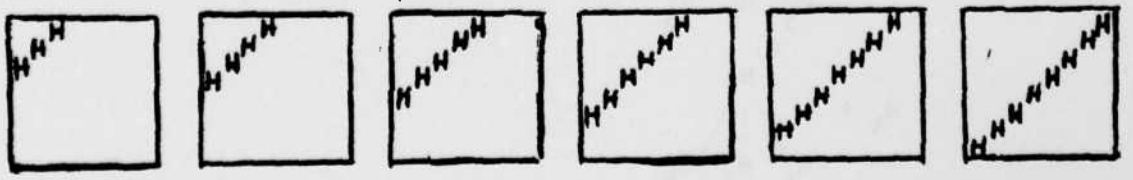
(b) Group A



(c) Group B

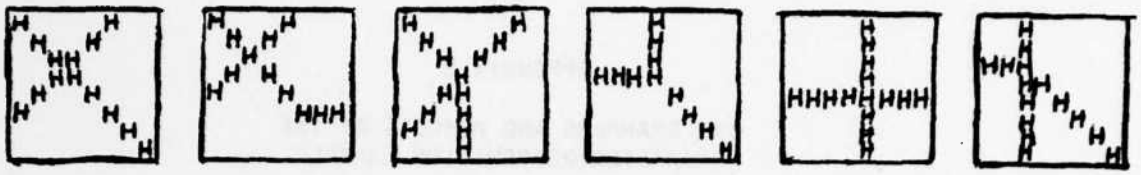


(d) Group C, D



(e) Group E

Figure C-1 Typical patterns generated from G_H



(f) Group F, M

Figure C-1 Continued

APPENDIX D

THE GRAMMARS AND RESULTS OF THE CHARACTER RECOGNITION EXAMPLE

D.1 Character Grammars

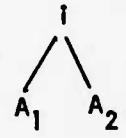
(a) Character A

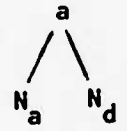
$G_A = (V_A, r_A, P_A, A)$ where

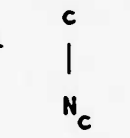
$V_A = \{A, A_1, A_2, N_a, N_d, N_c\}$, $\Sigma_A = \{i, a, c, d\}$

$r_A(i) = \{2\}$, $r_A(a) = \{0,2\}$, $r_A(c) = \{0,1\}$, $r_A(d) = \{0\}$

P_A:

$A \rightarrow$ 

$A_1 \rightarrow$ 

$A_2 \rightarrow$ 

$N_a \rightarrow a$, $N_d \rightarrow d$, $N_c \rightarrow c$

(b) Character C

$G_C = (V_C, r_C, P_C, C)$ where

$V_C = \{C, C_1, C_2, C_3, N_d, N_f\}$, $\Sigma_C = \{a, b, d, f\}$

$r_C(i) = \{2\}$, $r_C(a) = \{1\}$, $r_C(b) = \{1\}$, $r_C(d) = \{0,1\}$, $r_C(f) = \{0\}$

P_C:

$$C \rightarrow \overset{1}{\diagup\ \diagdown}\ C_1\ N_d$$

$$C_1 \rightarrow \overset{a}{|}\ C_2$$

$$C_2 \rightarrow \overset{b}{|}\ C_3$$

$$C_3 \rightarrow \overset{d}{|}\ N_f$$

$$N_d \rightarrow d, \quad N_f \rightarrow f$$

(c) Character D

$G_D = (V_D, r_D, P_D, D)$

$V_D = \{D, D_1, D_2, D_3, D_4, N_b, N_d\}, \quad \Sigma_D = \{i, b, d\}$

$r_D(i) = \{2\}, \ r_D(b) = \{0,1\}, \ r_D(d) = \{0,1\}$

P_D:

$$D \rightarrow \overset{i}{\diagup\ \diagdown}\ D_1\ D_2$$

$$D_1 \rightarrow \overset{b}{|}\ D_3$$

$$D_3 \rightarrow \overset{b}{|}\ N_d$$

$$D_2 \rightarrow \begin{matrix} b \\ | \\ D_4 \end{matrix}$$

$$D_4 \rightarrow \begin{matrix} b \\ | \\ N_b \end{matrix}$$

$$N_d \rightarrow d, \quad N_b \rightarrow b$$

(d) Character E

$G_E = (V_E, r_E, P_E, E)$

$V_E = \{E, E_1, E_2, E_3, N_d\}$, $\Sigma_E = \{1, b, d\}$

$r_E(1) = \{2\}$, $r_E(b) = \{1,2\}$, $r_E(d) = \{0,1\}$

P_E:

$$E \rightarrow \begin{matrix} 1 \\ E_1 \quad N_d \end{matrix}$$

$$E_1 \rightarrow \begin{matrix} b \\ E_2 \quad N_d \end{matrix}$$

$$E_2 \rightarrow \begin{matrix} b \\ | \\ E_3 \end{matrix}$$

$$E_3 \rightarrow \begin{matrix} d \\ | \\ N_d \end{matrix}$$

$$N_d \rightarrow d$$

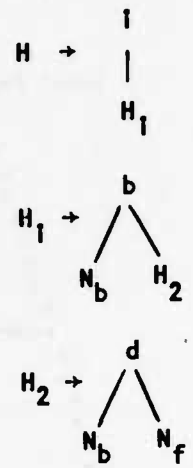
(e) Character H

$G_H \;= (V_H, r_H, P_H, H)$

$V_H \;= \{H, H_1, H_2, N_b, N_f\}, \quad \Sigma_H = \{l, b, d, f\}$

$r_H(l) = \{2\},\; r_H(b) = \{0,2\},\; r_H(d) = \{2\},\; r_H(f) = \{0\}$

P_H:

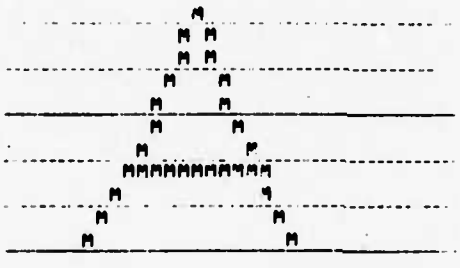


$N_b \rightarrow b$

$N_f \rightarrow f$

Appendix D.2. Classification Results of 26 Test Patterns

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	C	1
111	A	2
1111	*	1
1112	D	0
11111	A	0
121	C	0

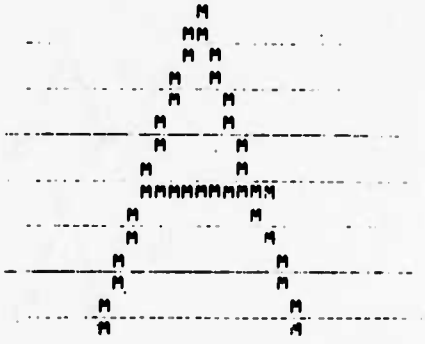
TIME USED FOR LINKING A TREE
.189 SEC

INPUT CHARACTER IS A

DISTANCE FORM NORMAL A
IS 0

TIME USED FOR PARSING
4.911 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	C	1
111	B	2
1111	*	1
1112	D	0
11111	B	0
121	C	0

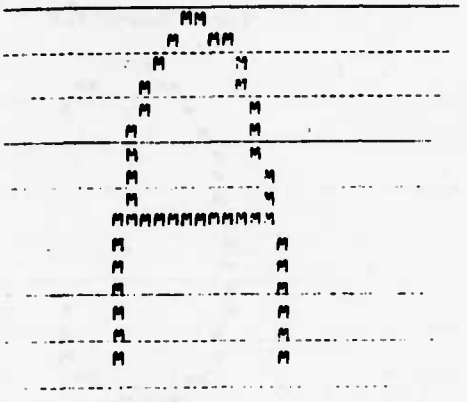
TIME USED FOR LINKING A TREE
.188 SEC

INPUT CHARACTER IS A

DISTANCE FORM NORMAL A
IS 2

TIME USED FOR PARSING
5.134 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	C	1
111	B	2
1111	*	1
1112	D	0
11111	B	0
121	B	0

TIME USED FOR LINKING A TREE
.188 SEC

INPUT CHARACTER IS A

DISTANCE FORM NORMAL A
IS 3

TIME USED FOR PARSING
5.161 SEC

INPUT CHARACTER

TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	B	0
111	A	2
1111	*	1
1112	D	1
11111	A	0
11121	B	0

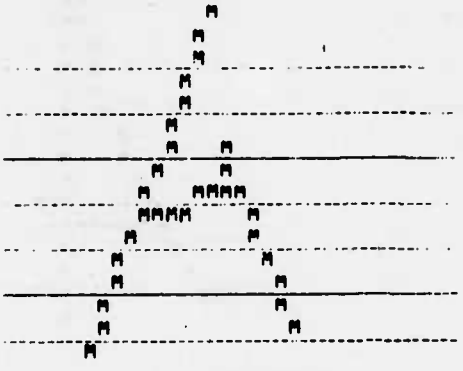
TIME USED FOR LINKING A TREE
.192 SEC

INPUT CHARACTER IS A

DISTANCE FORM NORMAL A
IS 3

TIME USED FOR PARSING
4.924 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	A	2
111	*	1
112	0	1
1111	A	0
1121	C	0

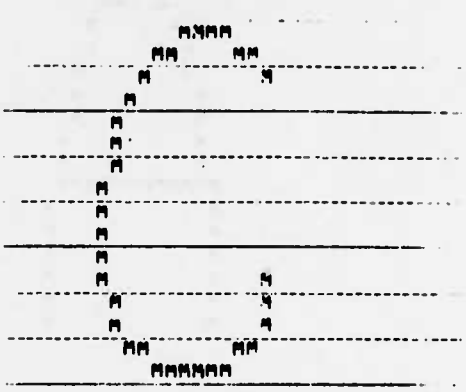
TIME USED FOR LINKING A TREE
.185 SEC

INPUT CHARACTER IS A

DISTANCE FORM NORMAL A
IS 3

TIME USED FOR PARSING
2.669 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	D	0
111	A	1
1111	B	1
11111	D	1
111111	F	0

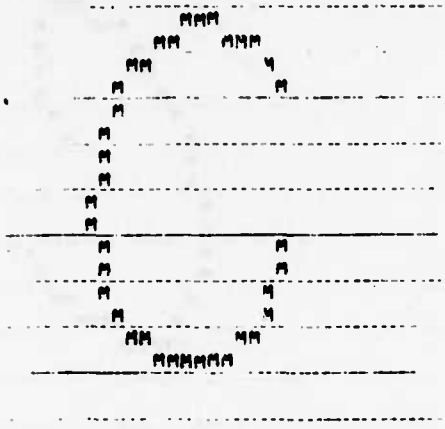
TIME USED FOR LINKING A TREE
.185 SEC

INPUT CHARACTER IS C

DISTANCE FORM NORMAL C
IS 0

TIME USED FOR PARSING
2.977 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	D	0
111	B	1
1111	C	1
11111	D	1
111111	F	0

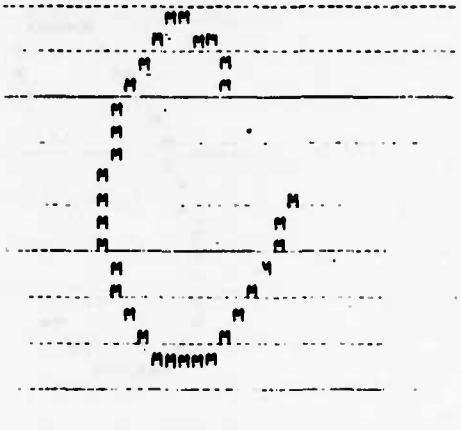
TIME USED FOR LINKING A TREE
.186 SEC

INPUT CHARACTER IS C

DISTANCE FORM NORMAL C
IS 1

TIME USED FOR PARSING
2.967 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	C	0
111	B	1
1111	C	1
11111	E	0

TIME USED FOR LINKING A TREE
.193 SEC

INPUT CHARACTER IS C

DISTANCE FORM NORMAL C
IS 3

TIME USED FOR PARSING
2.861 SEC

INPUT CHARACTER

TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	B	0
111	A	1
1111	B	1
11111	D	0

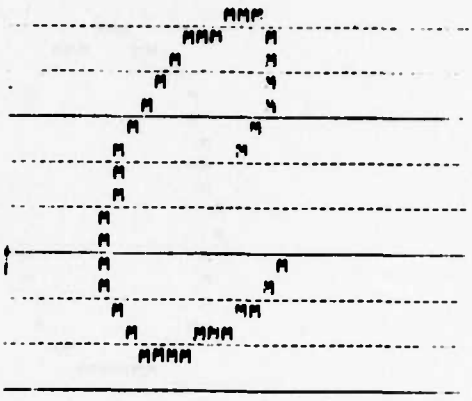
TIME USED FOR LINKING A TREE
.163 SEC

INPUT CHARACTER IS C

DISTANCE FORM NORMAL C
IS 1

TIME USED FOR PARSING
2.824 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	B	0
111	A	1
1111	B	1
11111	D	1
111111	E	0

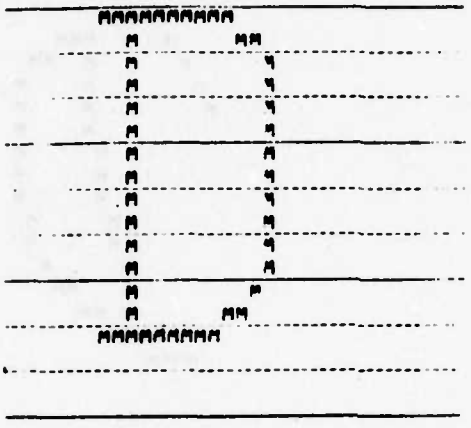
TIME USED FOR LINKING A TREE
.187 SEC

INPUT CHARACTER IS C

DISTANCE FORM NORMAL C
IS 2

TIME USED FOR PARSING
2.964 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	•	1
12	0	1
111	3	1
1111	3	1
121	3	1
11111	0	0
1211	3	0

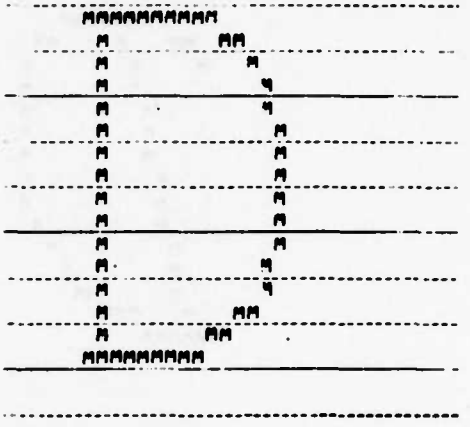
TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS 0

DISTANCE FORM NORMAL 0
IS 0

TIME USED FOR PARSING
3.138 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	•	1
12	0	1
111	3	1
1111	3	1
121	3	1
11111	0	0
1211	3	0

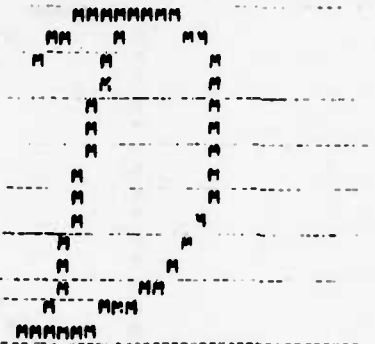
TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS 0

DISTANCE FORM NORMAL 0
IS 0

TIME USED FOR PARSING
3.139 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	•	2
12	D	1
111	•	1
112	B	1
1111	M	0
1121	B	1
121	A	0
11211	D	0

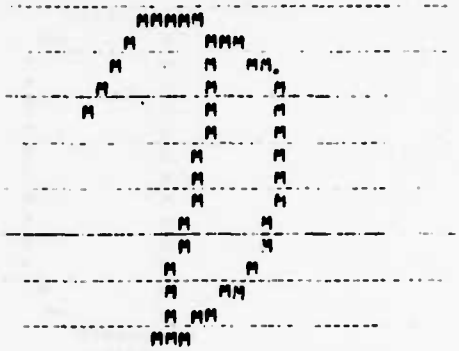
TIME USED FOR LINKING A TREE
.194 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
IS 2

TIME USED FOR PARSING
4.335 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	•	1
12	D	2
111	A	0
121	•	1
122	B	1
1211	B	1
12111	B	1
1221	A	0
121111	M	0

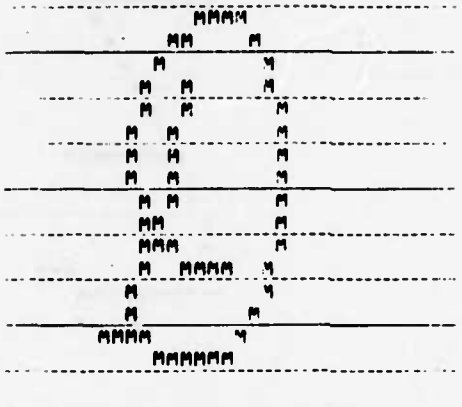
TIME USED FOR LINKING A TREE
.194 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
IS 5

TIME USED FOR PARSING
4.137 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	C	1
111	B	2
1111	*	2
1112	F	0
11111	*	1
11112	D	0
111111	B	1
121	B	0
1111111	D	0

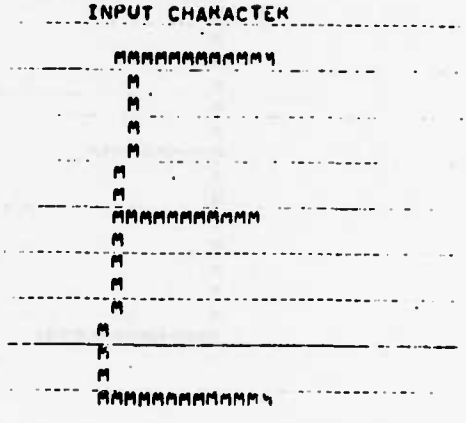
TIME USED FOR LINKING A TREE
.197 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
IS 4

TIME USED FOR PARSING
6.666 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	D	0
111	B	2
1111	*	1
1112	D	0
11111	B	1
111111	D	1
1111111	D	0

TIME USED FOR LINKING A TREE
.193 SEC

INPUT CHARACTER IS E

DISTANCE FORM NORMAL E
IS 0

TIME USED FOR PARSING
5.301 SEC

INPUT CHARACTER

TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	0	1
111	B	2
1111	*	1
1112	0	0
11111	B	1
121	0	0
111111	0	1
1111111	0	0

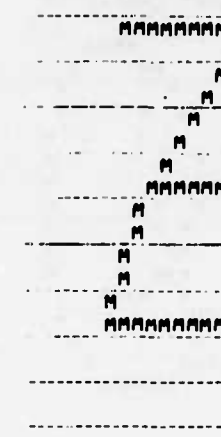
TIME USED FOR LINKING A TREE
.192 SEC

INPUT CHARACTER IS E

DISTANCE FORM NORMAL E
IS 1

TIME USED FOR PARSING
5.447 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	2
12	0	0
111	*	1
112	H	0
1111	A	2
11111	*	1
11112	0	0
111111	B	1
1111111	0	1
11111111	0	0

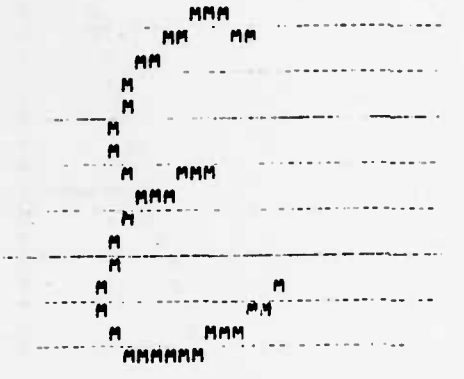
TIME USED FOR LINKING A TREE
.195 SEC

INPUT CHARACTER IS E

DISTANCE FORM NORMAL E
IS 2

TIME USED FOR PARSING
6.463 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	2
11	*	1
12	D	0
111	A	2
1111	*	1
1112	D	0
11111	D	1
111111	D	0

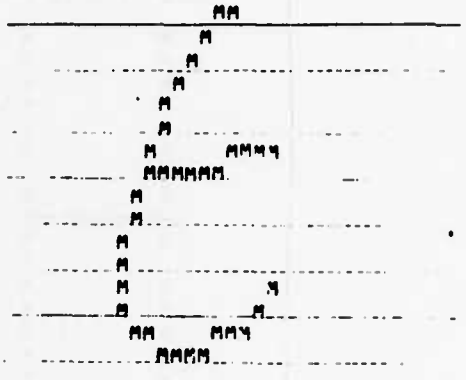
TIME USED FOR LINKING A TREE
.191 SEC

INPUT CHARACTER IS E

DISTANCE FORM NORMAL E
IS 1

TIME USED FOR PARSING
4.917 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	A	2
111	*	1
112	D	0
1111	B	1
11111	D	0

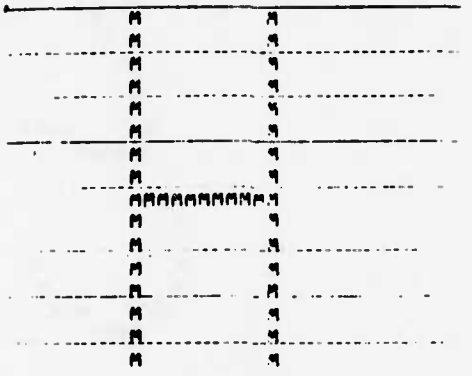
TIME USED FOR LINKING A TREE
.183 SEC

INPUT CHARACTER IS E

DISTANCE FORM NORMAL E
IS 3

TIME USED FOR PARSING
2.682 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	1
112	J	2
1111	B	0
1121	*	1
1122	F	0
11211	B	0

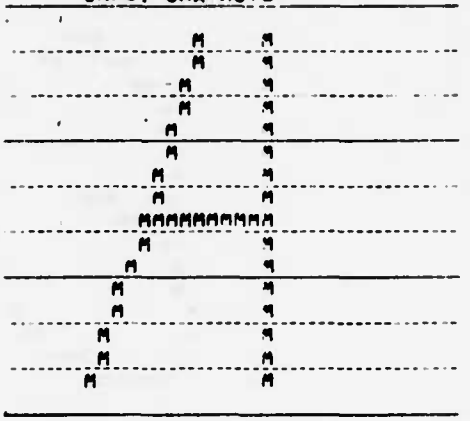
TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 0

TIME USED FOR PARSING
3.768 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	1
112	J	2
1111	A	0
1121	*	1
1122	F	0
11211	B	0

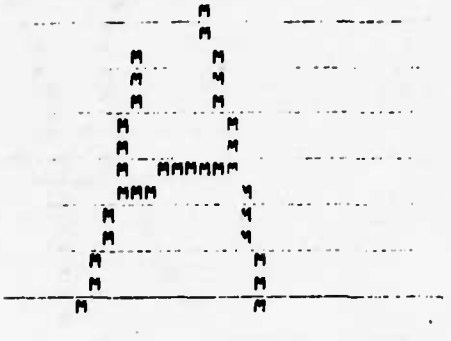
TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 1

TIME USED FOR PARSING
3.761 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	1
112	D	2
1111	A	0
1121	*	1
1122	F	0
11211	B	0

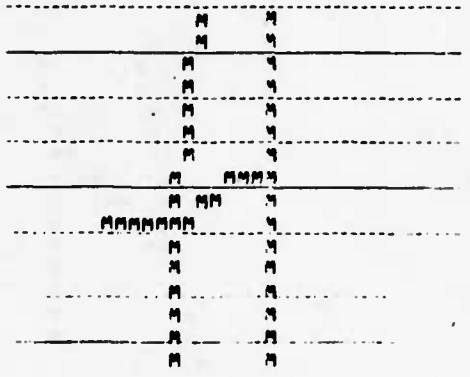
TIME USED FOR LINKING A TREE
.187 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 1

TIME USED FOR PARSING
5.795 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	2
112	D	2
1111	*	1
1112	H	0
11111	B	0
1121	*	1
1122	F	0
11211	B	0

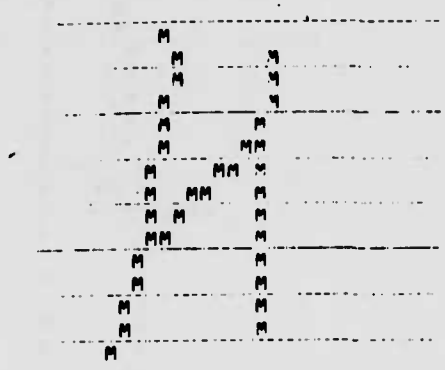
TIME USED FOR LINKING A TREE
.193 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 1

TIME USED FOR PARSING
5.117 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	1
112	E	2
1111	A	0
1121	*	1
1122	F	0
11211	B	0

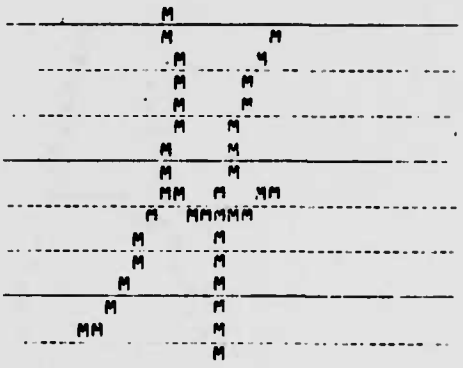
TIME USED FOR LINKING A TREE
.188 SEC

INPUT CHARACTER IS M

DISTANCE FORM NORMAL M
IS 2

TIME USED FOR PARSING
3.099 SEC

INPUT CHARACTER



TREE REPRESENTATION

TREE DOMAIN	PRIMITIVE	RANK
1	I	1
11	B	2
111	*	1
112	D	2
1111	A	0
1121	*	2
1122	E	0
11211	*	1
11212	D	0
112111	B	0

TIME USED FOR LINKING A TREE
.192 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 3

TIME USED FOR PARSING
5.121 SEC

APPENDIX E

SYNTHESIS GRAMMARS FOR NETTING AND REPTILE SKIN

E.1. Grammar for Netting (pattern D34)

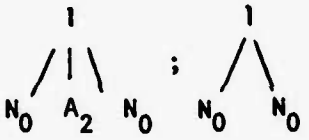
$G_4 = (V_4, r, P_4, S_4)$ over $\langle \Sigma, r \rangle$ and $G_{S_4} = (V_4, r, P_{S_4}, S_4)$ over $\langle \Sigma, r \rangle$

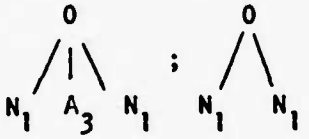
$V_4 = \{A_{0,1,\ldots,9}, B_{0,1,\ldots,9}, C_{0,1,\ldots,9}, D_{0,1,\ldots,9}, E_{0,1,\ldots,9}, F_{0,1,\ldots,9},$

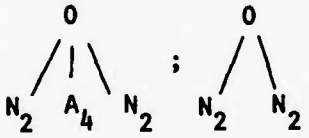
$N_{0,1,\ldots,6}, V_{0,1,\ldots,3}\}$ Σ

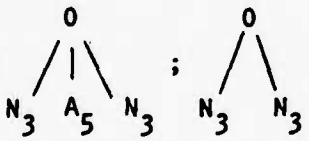
$S_4 = \{A_1, B_0, B_9, C_8, A_7, B_6, C_5, A_4, B_3, C_2, D_1, E_0, E_9, F_8, D_7, E_6, F_5, D_4, E_3, F_2\}$

P_4:

$A_1 \rightarrow$  ;

$A_2 \rightarrow$  ;

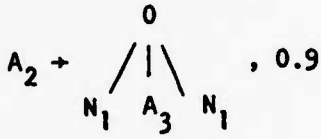
$A_3 \rightarrow$  ;

$A_4 \rightarrow$  ;

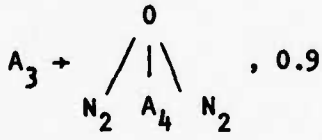
P_{S_4}:

$A_1 \rightarrow$  , 0.9 ;

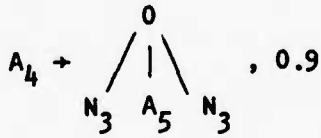
, 0.1

$A_2 \rightarrow$  , 0.9 ;

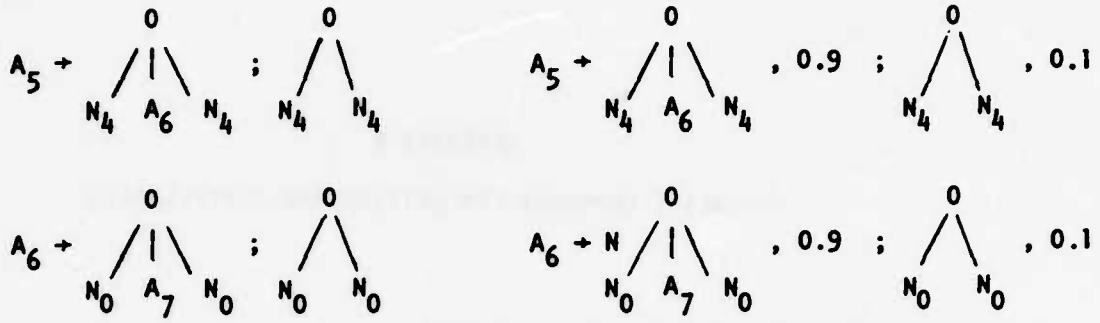
, 0.1

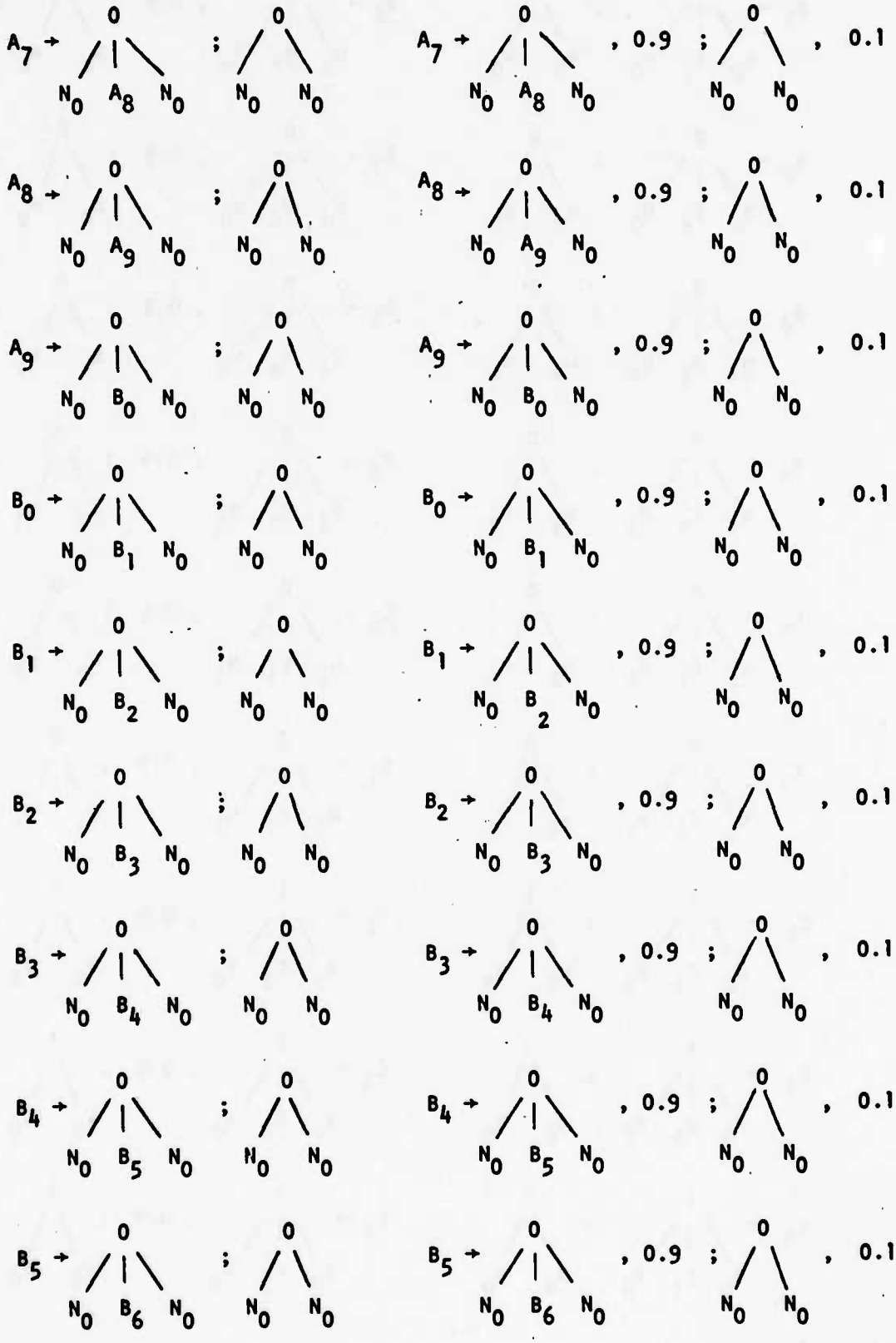
$A_3 \rightarrow$  , 0.9 ;

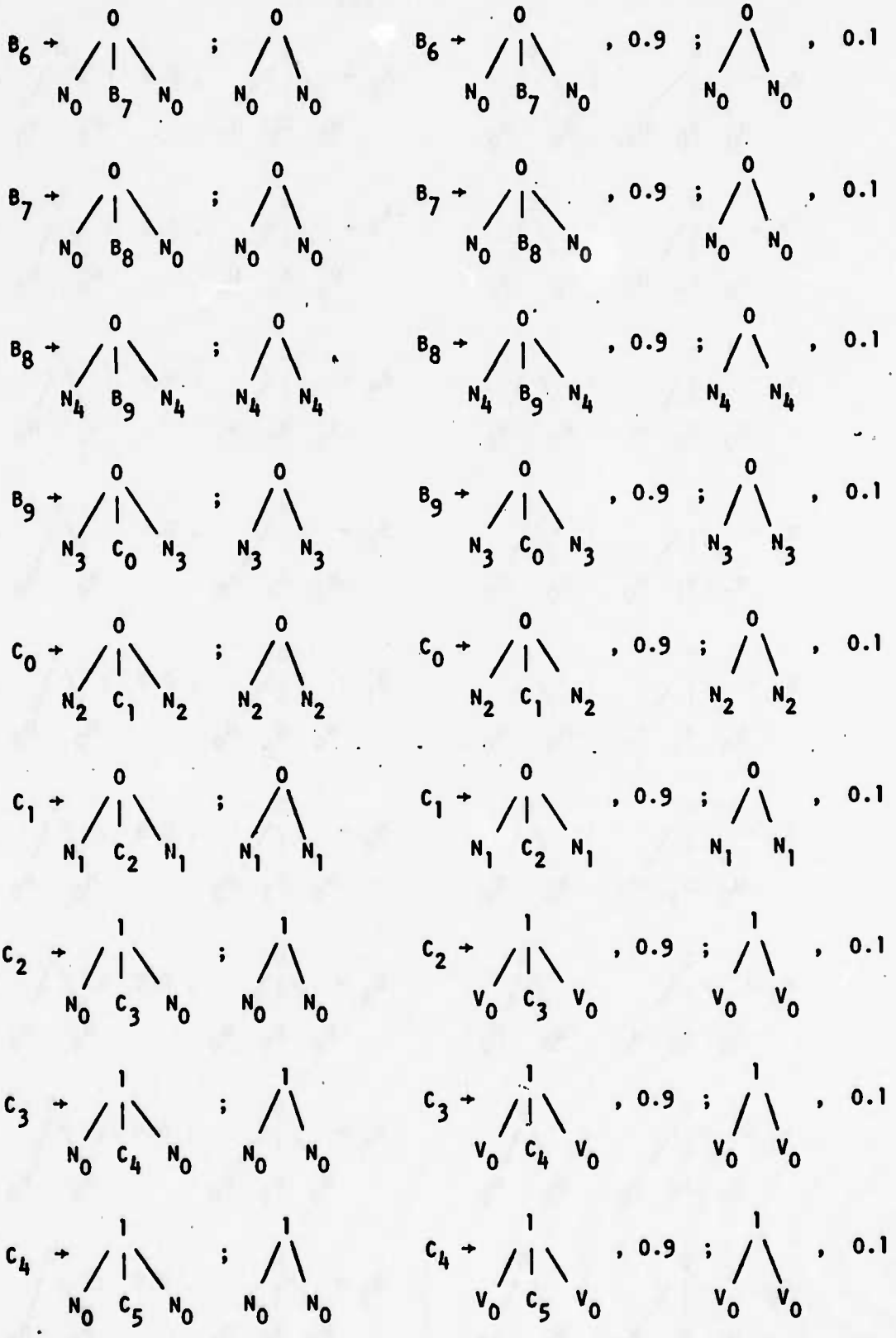
, 0.1

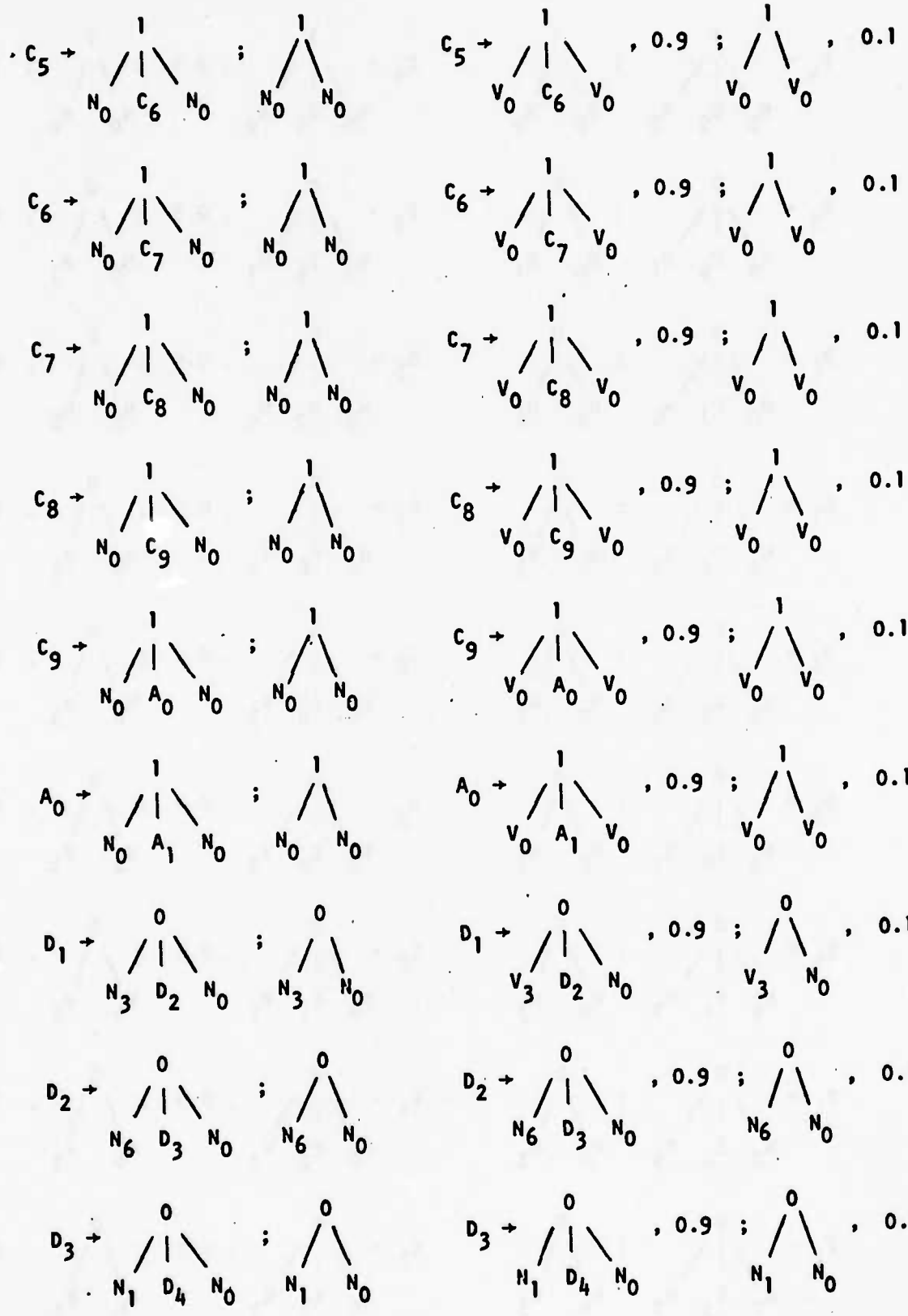
$A_4 \rightarrow$  , 0.9 ;

, 0.1

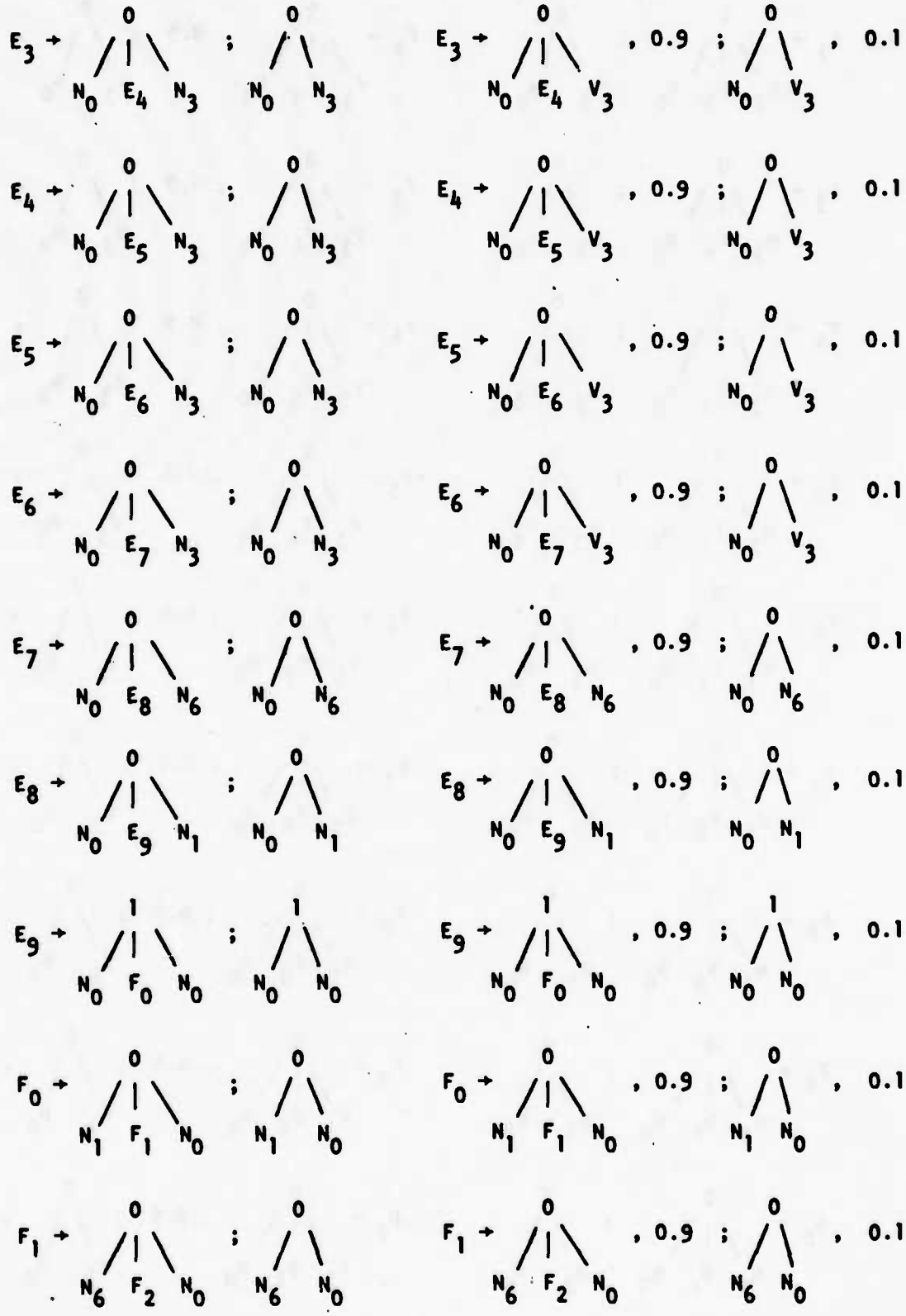

A5 → O (N4 A6 N4) ; O (N4 N4)
A5 → O (N4 A6 N4) , 0.9 ; O (N4 N4) , 0.1
A6 → O (N0 A7 N0) ; O (N0 N0)
A6 → N O (N0 A7 N0) , 0.9 ; O (N0 N0) , 0.1

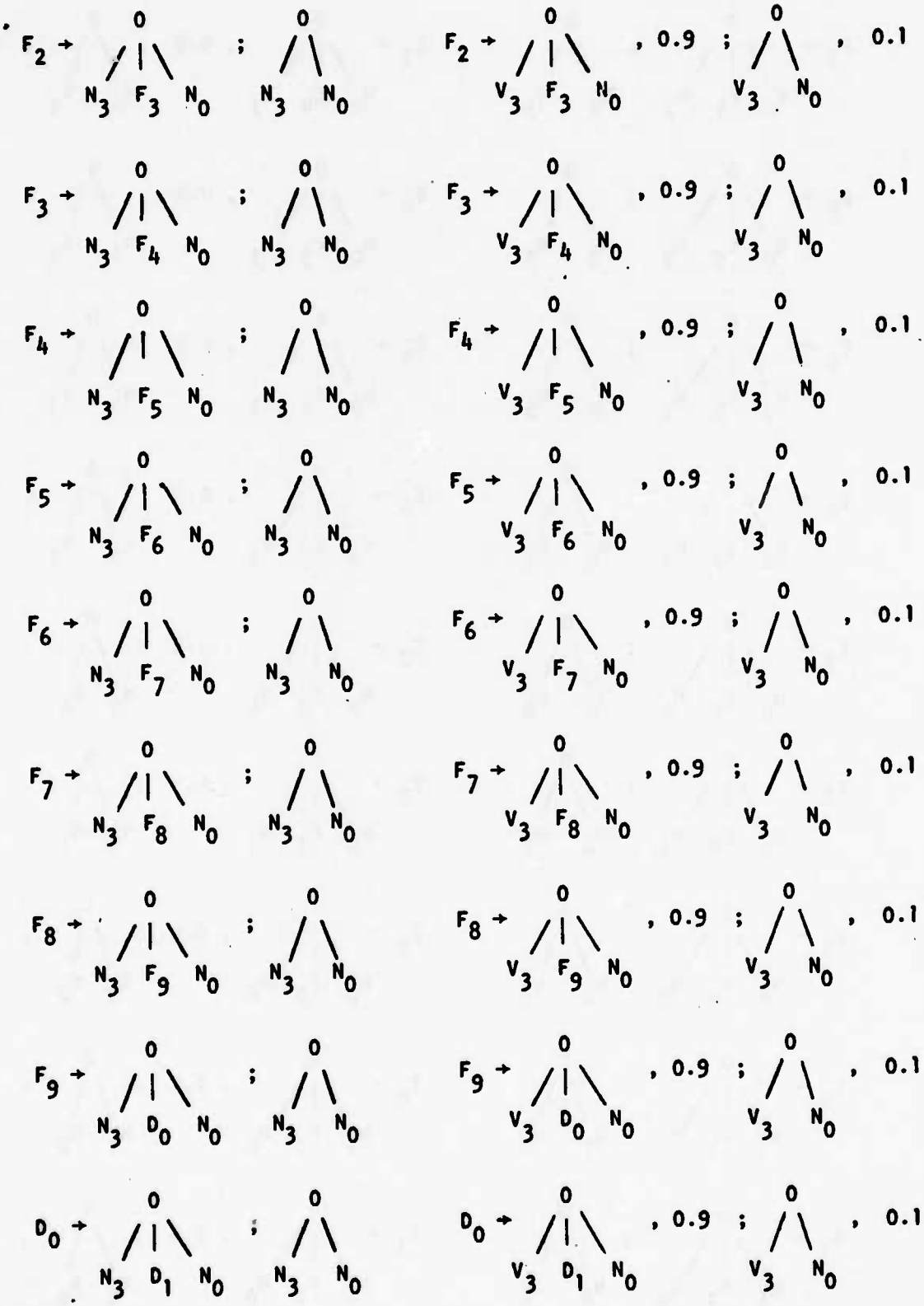

A7 → 0 N0 A8 N0 ; 0 N0 N0
A7 → 0 N0 A8 N0 , 0.9 ; 0 N0 N0 , 0.1
A8 → 0 N0 A9 N0 ; 0 N0 N0
A8 → 0 N0 A9 N0 , 0.9 ; 0 N0 N0 , 0.1
A9 → 0 N0 B0 N0 ; 0 N0 N0
A9 → 0 N0 B0 N0 , 0.9 ; 0 N0 N0 , 0.1
B0 → 0 N0 B1 N0 ; 0 N0 N0
B0 → 0 N0 B1 N0 , 0.9 ; 0 N0 N0 , 0.1
B1 → 0 N0 B2 N0 ; 0 N0 N0
B1 → 0 N0 B2 N0 , 0.9 ; 0 N0 N0 , 0.1
B2 → 0 N0 B3 N0 ; 0 N0 N0
B2 → 0 N0 B3 N0 , 0.9 ; 0 N0 N0 , 0.1
B3 → 0 N0 B4 N0 ; 0 N0 N0
B3 → 0 N0 B4 N0 , 0.9 ; 0 N0 N0 , 0.1
B4 → 0 N0 B5 N0 ; 0 N0 N0
B4 → 0 N0 B5 N0 , 0.9 ; 0 N0 N0 , 0.1
B5 → 0 N0 B6 N0 ; 0 N0 N0
B5 → 0 N0 B6 N0 , 0.9 ; 0 N0 N0 , 0.1


B6 → 0 N0 B7 N0 ; 0 N0 N0
B6 → 0 N0 B7 N0 , 0.9 ; 0 N0 N0 , 0.1
B7 → 0 N0 B8 N0 ; 0 N0 N0
B7 → 0 N0 B8 N0 , 0.9 ; 0 N0 N0 , 0.1
B8 → 0 N4 B9 N4 ; 0 N4 N4
B8 → 0 N4 B9 N4 , 0.9 ; 0 N4 N4 , 0.1
B9 → 0 N3 C0 N3 ; 0 N3 N3
B9 → 0 N3 C0 N3 , 0.9 ; 0 N3 N3 , 0.1
C0 → 0 N2 C1 N2 ; 0 N2 N2
C0 → 0 N2 C1 N2 , 0.9 ; 0 N2 N2 , 0.1
C1 → 0 N1 C2 N1 ; 0 N1 N1
C1 → 0 N1 C2 N1 , 0.9 ; 0 N1 N1 , 0.1
C2 → 1 N0 C3 N0 ; 1 N0 N0
C2 → 1 V0 C3 V0 , 0.9 ; 1 V0 V0 , 0.1
C3 → 1 N0 C4 N0 ; 1 N0 N0
C3 → 1 V0 C4 V0 , 0.9 ; 1 V0 V0 , 0.1
C4 → 1 N0 C5 N0 ; 1 N0 N0
C4 → 1 V0 C5 V0 , 0.9 ; 1 V0 V0 , 0.1


C5 → 1 N0 C6 N0 ; 1 N0 N0
C5 → 1 V0 C6 V0 , 0.9 ; 1 V0 V0 , 0.1
C6 → 1 N0 C7 N0 ; 1 N0 N0
C6 → 1 V0 C7 V0 , 0.9 ; 1 V0 V0 , 0.1
C7 → 1 N0 C8 N0 ; 1 N0 N0
C7 → 1 V0 C8 V0 , 0.9 ; 1 V0 V0 , 0.1
C8 → 1 N0 C9 N0 ; 1 N0 N0
C8 → 1 V0 C9 V0 , 0.9 ; 1 V0 V0 , 0.1
C9 → 1 N0 A0 N0 ; 1 N0 N0
C9 → 1 V0 A0 V0 , 0.9 ; 1 V0 V0 , 0.1
A0 → 1 N0 A1 N0 ; 1 N0 N0
A0 → 1 V0 A1 V0 , 0.9 ; 1 V0 V0 , 0.1
D1 → 0 N3 D2 N0 ; 0 N3 N0
D1 → 0 V3 D2 N0 , 0.9 ; 0 V3 N0 , 0.1
D2 → 0 N6 D3 N0 ; 0 N6 N0
D2 → 0 N6 D3 N0 , 0.9 ; 0 N6 N0 , 0.1
D3 → 0 N1 D4 N0 ; 0 N1 N0
D3 → 0 N1 D4 N0 , 0.9 ; 0 N1 N0 , 0.1

```
          1              1                       1                   1
D4 →    / | \     ;     / \         D4 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  D5  N0      N0   N0                N0  D5  N0           N0   N0

          0              0                       0                   0
D5 →    / | \     ;     / \         D5 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  D6  N1      N0   N1                N0  D6  N1           N0   N1

          0              0                       0                   0
D6 →    / | \     ;     / \         D6 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  D7  N6      N0   N6                N0  D7  N6           N0   N6

          0              0                       0                   0
D7 →    / | \     ;     / \         D7 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  D8  N3      N0   N3                N0  D8  V3           N0   V3

          0              0                       0                   0
D8 →    / | \     ;     / \         D8 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  D9  N3      N0   N3                N0  D9  V3           N0   V3

          0              0                       0                   0
D9 →    / | \     ;     / \         D9 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  E0  N3      N0   N3                N0  E0  V3           N0   V3

          0              0                       0                   0
E0 →    / | \     ;     / \         E0 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  E1  N3      N0   N3                N0  E1  V3           N0   V3

          0              0                       0                   0
E1 →    / | \     ;     / \         E1 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  E2  N3      N0   N3                N0  E2  V3           N0   V3

          0              0                       0                   0
E2 →    / | \     ;     / \         E2 →       / | \    , 0.9  ;    / \    ,  0.1
      N0  E3  N3      N0   N3                N0  E3  V3           N0   V3
```


E3 → 0 : N0 E4 N3 ; 0 : N0 N3 , 0.9 ; , 0.1 : N0 E4 V3 ; N0 V3
E4 → 0 : N0 E5 N3 ; 0 : N0 N3 , 0.9 ; , 0.1 : N0 E5 V3 ; N0 V3
E5 → 0 : N0 E6 N3 ; 0 : N0 N3 , 0.9 ; , 0.1 : N0 E6 V3 ; N0 V3
E6 → 0 : N0 E7 N3 ; 0 : N0 N3 , 0.9 ; , 0.1 : N0 E7 V3 ; N0 V3
E7 → 0 : N0 E8 N6 ; 0 : N0 N6 , 0.9 ; , 0.1 : N0 E8 N6 ; N0 N6
E8 → 0 : N0 E9 N1 ; 0 : N0 N1 , 0.9 ; , 0.1 : N0 E9 N1 ; N0 N1
E9 → 1 : N0 F0 N0 ; 1 : N0 N0 , 0.9 ; , 0.1 : N0 F0 N0 ; N0 N0
F0 → 0 : N1 F1 N0 ; 0 : N1 N0 , 0.9 ; , 0.1 : N1 F1 N0 ; N1 N0
F1 → 0 : N6 F2 N0 ; 0 : N6 N0 , 0.9 ; , 0.1 : N6 F2 N0 ; N6 N0


F2 → O N3 F3 N0 ; O N3 N0
F2 → O V3 F3 N0 , 0.9 ; O V3 N0 , 0.1
F3 → O N3 F4 N0 ; O N3 N0
F3 → O V3 F4 N0 , 0.9 ; O V3 N0 , 0.1
F4 → O N3 F5 N0 ; O N3 N0
F4 → O V3 F5 N0 , 0.9 ; O V3 N0 , 0.1
F5 → O N3 F6 N0 ; O N3 N0
F5 → O V3 F6 N0 , 0.9 ; O V3 N0 , 0.1
F6 → O N3 F7 N0 ; O N3 N0
F6 → O V3 F7 N0 , 0.9 ; O V3 N0 , 0.1
F7 → O N3 F8 N0 ; O N3 N0
F7 → O V3 F8 N0 , 0.9 ; O V3 N0 , 0.1
F8 → O N3 F9 N0 ; O N3 N0
F8 → O V3 F9 N0 , 0.9 ; O V3 N0 , 0.1
F9 → O N3 D0 N0 ; O N3 N0
F9 → O V3 D0 N0 , 0.9 ; O V3 N0 , 0.1
D0 → O N3 D1 N0 ; O N3 N0
D0 → O V3 D1 N0 , 0.9 ; O V3 N0 , 0.1

$$N_0 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \; ; \; 0$$

$$N_1 \rightarrow \begin{matrix} 1 \\ | \\ N_0 \end{matrix} \; ; \; 1$$

$$N_2 \rightarrow \begin{matrix} 0 \\ | \\ N_1 \end{matrix}$$

$$N_3 \rightarrow \begin{matrix} 0 \\ | \\ N_2 \end{matrix}$$

$$N_4 \rightarrow \begin{matrix} 0 \\ | \\ N_3 \end{matrix}$$

$$N_5 \rightarrow \begin{matrix} 1 \\ | \\ N_2 \end{matrix}$$

$$N_6 \rightarrow \begin{matrix} 0 \\ | \\ N_5 \end{matrix}$$

$$N_0 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \; , 0.75 \; ; \; 0 \; , 0.25$$

$$V_0 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \; , 0.6 \; ; \; \begin{matrix} 1 \\ | \\ N_0 \end{matrix} \; , 0.4$$

$$N_1 \rightarrow \begin{matrix} 1 \\ | \\ N_0 \end{matrix} \; , 0.5 \; ; \; \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \; , 0.2$$

$$1 \; , 0.2 \; ; \; 0 \; , 0.1$$

$$N_2 \rightarrow \begin{matrix} 0 \\ | \\ N_1 \end{matrix} \; , 1.0$$

$$N_3 \rightarrow \begin{matrix} 0 \\ | \\ N_2 \end{matrix} \; , 1.0$$

$$N_4 \rightarrow \begin{matrix} 0 \\ | \\ N_3 \end{matrix} \; , 1.0$$

$$N_5 \rightarrow \begin{matrix} 1 \\ | \\ N_2 \end{matrix} \; , 0.7 \; ; \; \begin{matrix} 0 \\ | \\ N_2 \end{matrix} \; , 0.3$$

$$N_6 \rightarrow \begin{matrix} 0 \\ | \\ N_5 \end{matrix} \; , 1.0$$

$V_2 \rightarrow \begin{matrix} 1 \\ | \\ N_1 \end{matrix}$, 1.0

$V_3 \rightarrow \begin{matrix} 0 \\ | \\ V_3 \end{matrix}$, 1.0

$G_4' = (V_4', r_4', P_4', X_1)$ over $<\Sigma_4', r_4'>$

$V_4' = \{X_{1,2,\ldots,9,0}, Y_{1,2,\ldots 9,0}\} \cup \Sigma_4'$

$\Sigma_4' = \{A_1, B_0, B_9, C_8, A_7, B_6, C_5, A_4, B_3, C_2, D_1, E_0, E_9, F_8, D_7, E_6, F_5, D_4, E_3, F_2\}$

$r_4' = \{0, 1, 2\}$

P_4':

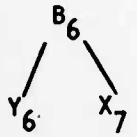
$X_1 \rightarrow \begin{matrix} A_1 \\ / \quad \backslash \\ Y_1 \quad X_2 \end{matrix}$; $\begin{matrix} A_1 \\ | \\ X_2 \end{matrix}$; A_1

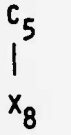
$X_2 \rightarrow \begin{matrix} B_0 \\ | \\ X_3 \end{matrix}$; B_0

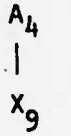
$X_3 \rightarrow \begin{matrix} B_9 \\ | \\ X_4 \end{matrix}$; B_9

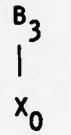
$X_4 \rightarrow \begin{matrix} C_8 \\ | \\ X_5 \end{matrix}$; C_8

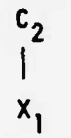
$X_5 \rightarrow \begin{matrix} A_7 \\ | \\ X_6 \end{matrix}$; A_7

$X_6 \rightarrow$  ; ; B_6

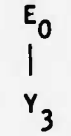
$X_7 \rightarrow$  ; C_5

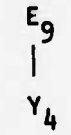
$X_8 \rightarrow$  ; A_4

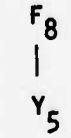
$X_9 \rightarrow$  ; B_3

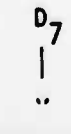
$X_0 \rightarrow$  ; C_2

$Y_1 \rightarrow$  ; ; D_1

$Y_2 \rightarrow$  ; E_0

$Y_3 \rightarrow$  ; E_9

$Y_4 \rightarrow$  ; F_8

$Y_5 \rightarrow$  ; D_7

$$Y_6 \rightarrow \overset{E_6}{\underset{X_1 \quad Y_7}{/\ \backslash}} \;;\; \overset{E_6}{|}\ \underset{Y_7}{} \;;\; E_6$$

$$Y_7 \rightarrow \overset{F_5}{|}\ \underset{Y_8}{} \;;\; F_5$$

$$Y_8 \rightarrow \overset{D_4}{|}\ \underset{Y_9}{} \;;\; D_4 \qquad Y_0 \rightarrow \overset{F_2}{|}\ \underset{Y_1}{} \;;\; F_2$$

$$Y_9 \rightarrow \overset{E_3}{|}\ \underset{Y_0}{} \;;\; E_3$$

E.2. Grammar for Reptile Skin (pattern D22)

$G_5 = (V_5, r, P_5, S_5)$ over $<\Sigma, r>$

$V_5 = S_5 \cup \Sigma \cup \{N_{0,1,\ldots,9}, V_{0,1,\ldots,5}\}$

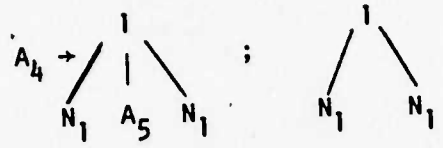
$S_5 = \{X_I \mid X \in \{A,B,C,D,E,F,G,H,I\},\ I \in \{1,2,\ldots,9\}\}$

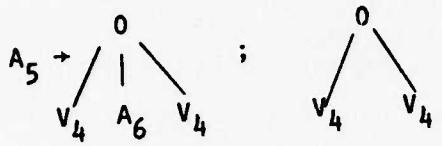
P_5:

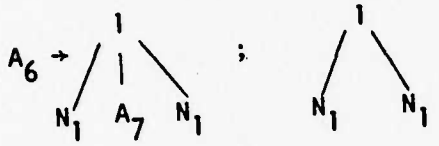
$$A_1 \rightarrow \overset{1}{\underset{V_0 \quad A_2 \quad V_0}{/\ |\ \backslash}} \;;\; \overset{1}{\underset{V_0 \quad V_0}{/\ \backslash}}$$

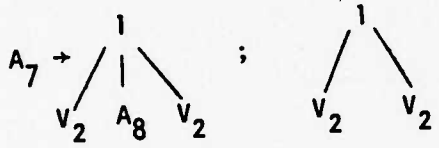
$$A_2 \rightarrow \overset{1}{\underset{V_0 \quad A_3 \quad V_0}{/\ |\ \backslash}} \;;\; \overset{1}{\underset{V_0 \quad V_0}{/\ \backslash}}$$

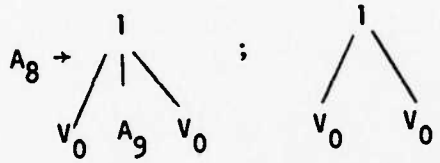
$$A_3 \rightarrow \overset{1}{\underset{V_2 \quad A_4 \quad V_2}{/\ |\ \backslash}} \;;\; \overset{1}{\underset{V_2 \quad V_2}{/\ \backslash}}$$

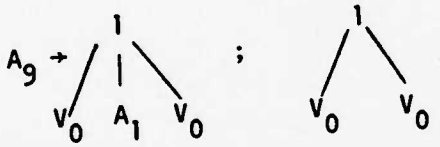

A4 → 1 N1 A5 N1 ; 1 N1 N1

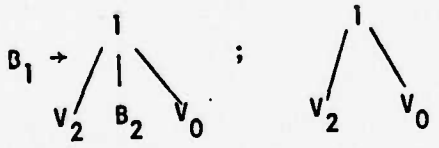

A5 → 0 V4 A6 V4 ; 0 V4 V4

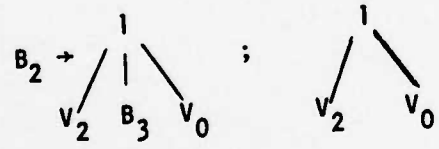

A6 → 1 N1 A7 N1 ; 1 N1 N1

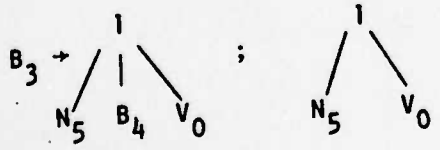

A7 → 1 V2 A8 V2 ; 1 V2 V2

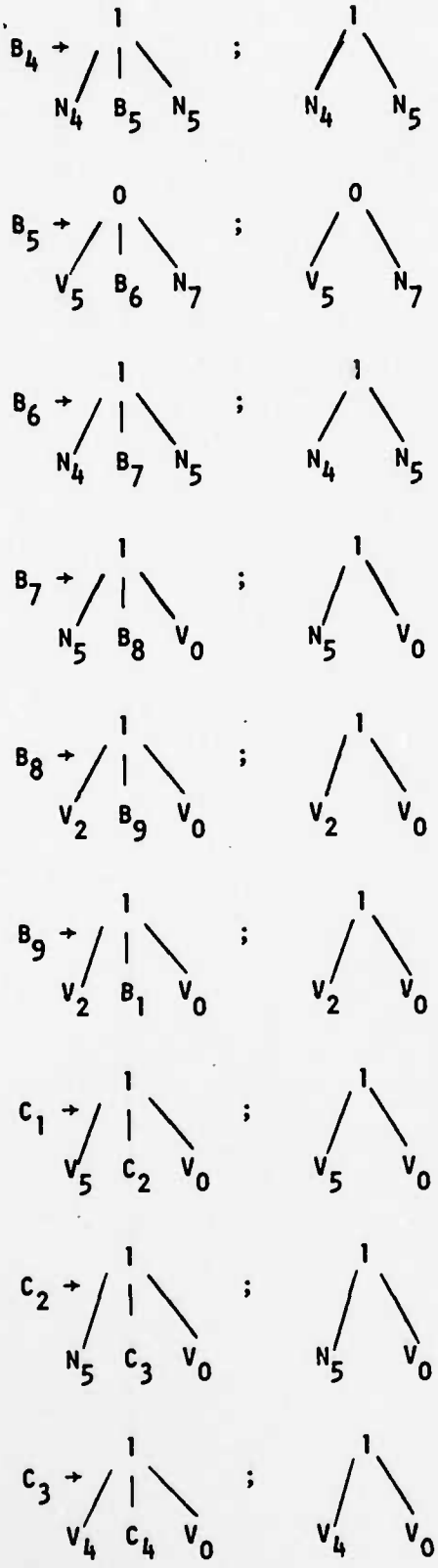

A8 → 1 V0 A9 V0 ; 1 V0 V0

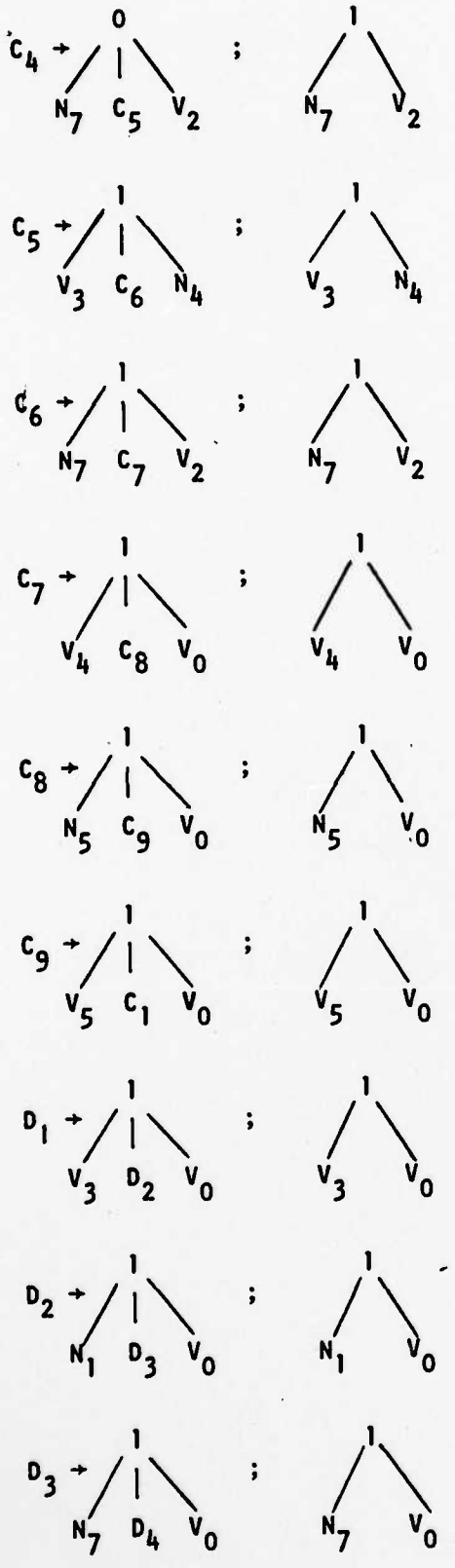

A9 → 1 V0 A1 V0 ; 1 V0 V0

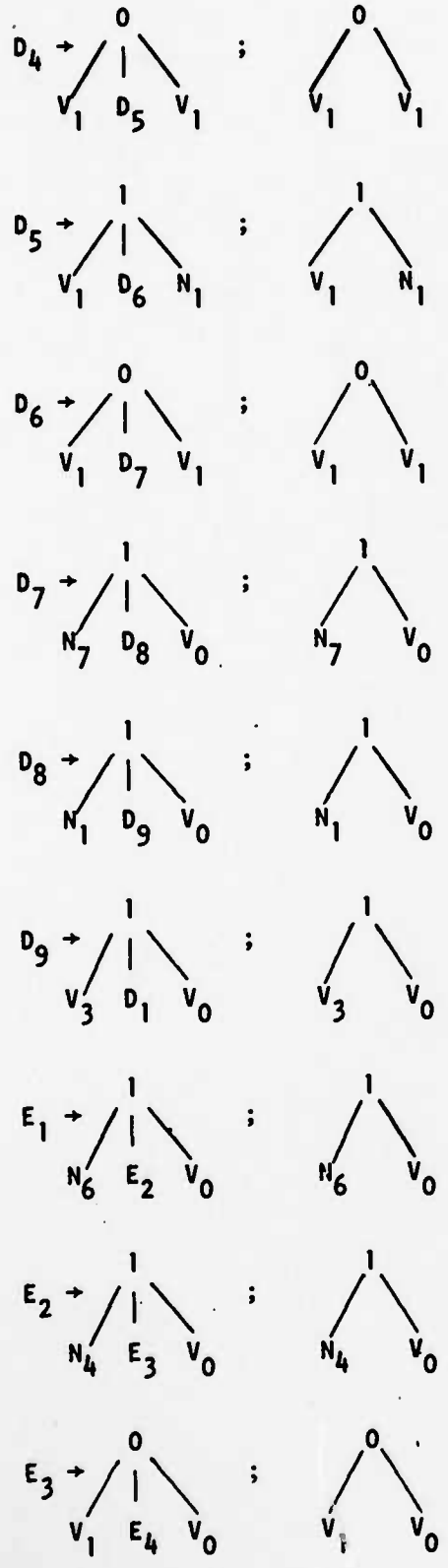

B1 → 1 V2 B2 V0 ; 1 V2 V0

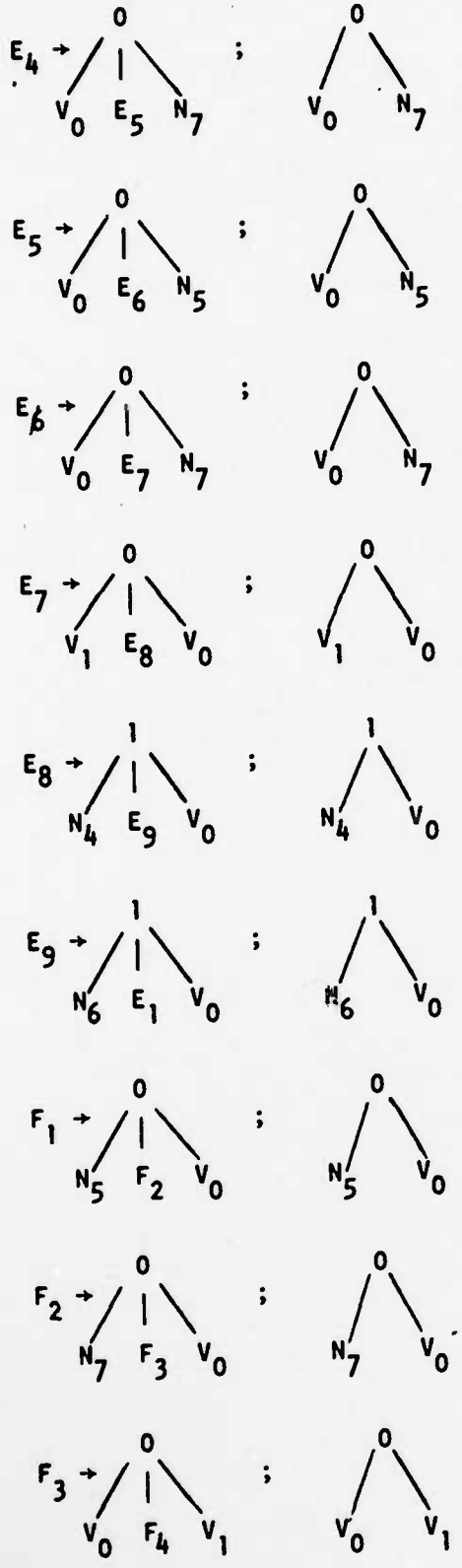

B2 → 1 V2 B3 V0 ; 1 V2 V0

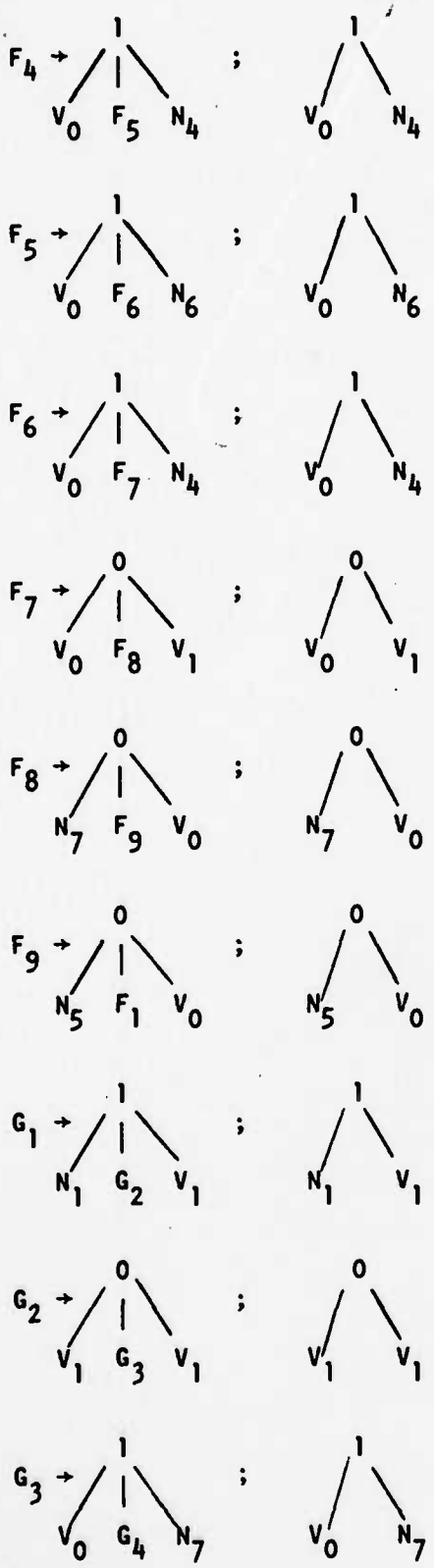

B3 → 1 N5 B4 V0 ; 1 N5 V0

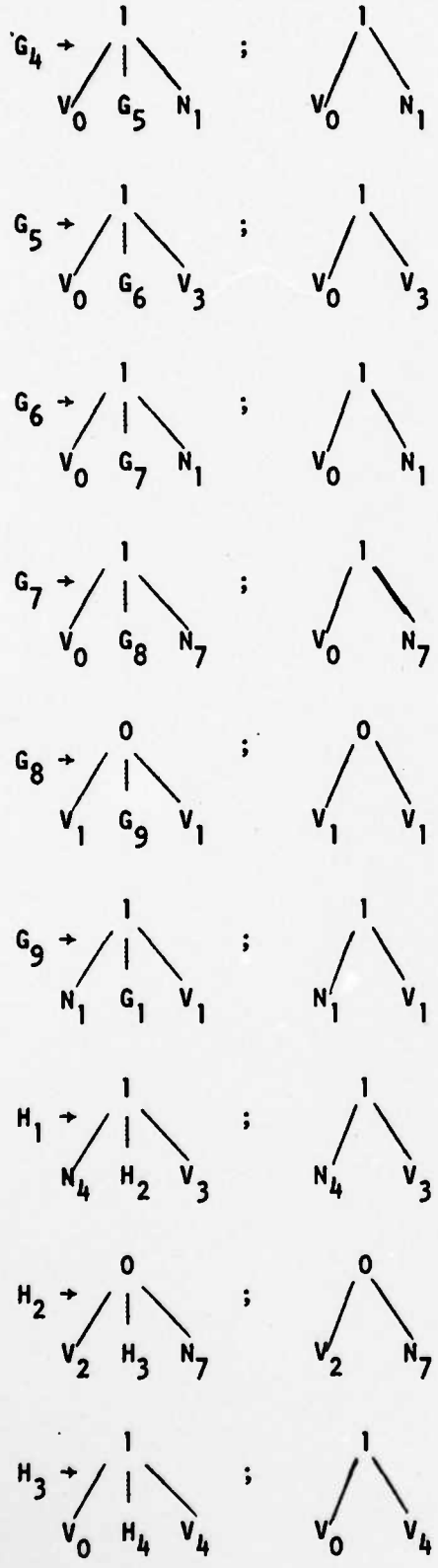

B4 → 1 N4 B5 N5 ; 1 N4 N5
B5 → 0 V5 B6 N7 ; 0 V5 N7
B6 → 1 N4 B7 N5 ; 1 N4 N5
B7 → 1 N5 B8 V0 ; 1 N5 V0
B8 → 1 V2 B9 V0 ; 1 V2 V0
B9 → 1 V2 B1 V0 ; 1 V2 V0
C1 → 1 V5 C2 V0 ; 1 V5 V0
C2 → 1 N5 C3 V0 ; 1 N5 V0
C3 → 1 V4 C4 V0 ; 1 V4 V0

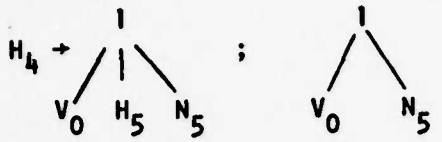

C4 → 0 N7 C5 V2 ; 1 N7 V2
C5 → 1 V3 C6 N4 ; 1 V3 N4
C6 → 1 N7 C7 V2 ; 1 N7 V2
C7 → 1 V4 C8 V0 ; 1 V4 V0
C8 → 1 N5 C9 V0 ; 1 N5 V0
C9 → 1 V5 C1 V0 ; 1 V5 V0
D1 → 1 V3 D2 V0 ; 1 V3 V0
D2 → 1 N1 D3 V0 ; 1 N1 V0
D3 → 1 N7 D4 V0 ; 1 N7 V0

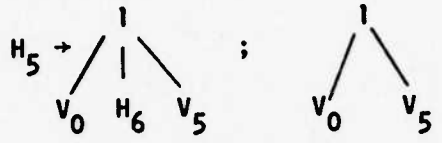

D4 → 0 V1 D5 V1 ; 0 V1 V1
D5 → 1 V1 D6 N1 ; 1 V1 N1
D6 → 0 V1 D7 V1 ; 0 V1 V1
D7 → 1 N7 D8 V0 ; 1 N7 V0
D8 → 1 N1 D9 V0 ; 1 N1 V0
D9 → 1 V3 D1 V0 ; 1 V3 V0
E1 → 1 N6 E2 V0 ; 1 N6 V0
E2 → 1 N4 E3 V0 ; 1 N4 V0
E3 → 0 V1 E4 V0 ; 0 V1 V0

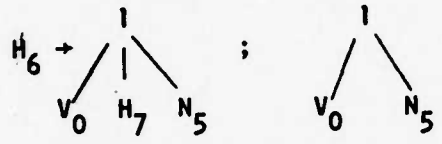

E4 → 0 V0 E5 N7 ; 0 V0 N7
E5 → 0 V0 E6 N5 ; 0 V0 N5
E6 → 0 V0 E7 N7 ; 0 V0 N7
E7 → 0 V1 E8 V0 ; 0 V1 V0
E8 → 1 N4 E9 V0 ; 1 N4 V0
E9 → 1 N6 E1 V0 ; 1 N6 V0
F1 → 0 N5 F2 V0 ; 0 N5 V0
F2 → 0 N7 F3 V0 ; 0 N7 V0
F3 → 0 V0 F4 V1 ; 0 V0 V1

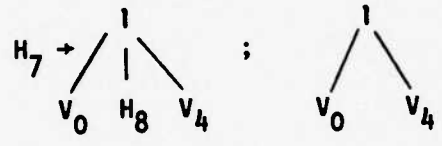

F4 → 1 V0 F5 N4 ; 1 V0 N4
F5 → 1 V0 F6 N6 ; 1 V0 N6
F6 → 1 V0 F7 N4 ; 1 V0 N4
F7 → 0 V0 F8 V1 ; 0 V0 V1
F8 → 0 N7 F9 V0 ; 0 N7 V0
F9 → 0 N5 F1 V0 ; 0 N5 V0
G1 → 1 N1 G2 V1 ; 1 N1 V1
G2 → 0 V1 G3 V1 ; 0 V1 V1
G3 → 1 V0 G4 N7 ; 1 V0 N7

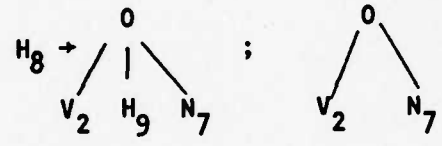

G4 → 1 V0 G5 N1 ; 1 V0 N1
G5 → 1 V0 G6 V3 ; 1 V0 V3
G6 → 1 V0 G7 N1 ; 1 V0 N1
G7 → 1 V0 G8 N7 ; 1 V0 N7
G8 → 0 V1 G9 V1 ; 0 V1 V1
G9 → 1 N1 G1 V1 ; 1 N1 V1
H1 → 1 N4 H2 V3 ; 1 N4 V3
H2 → 0 V2 H3 N7 ; 0 V2 N7
H3 → 1 V0 H4 V4 ; 1 V0 V4

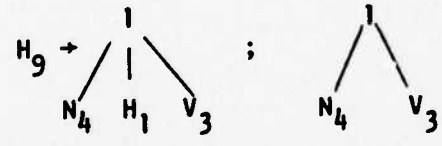

H4 → 1 V0 H5 N5 ; 1 V0 N5

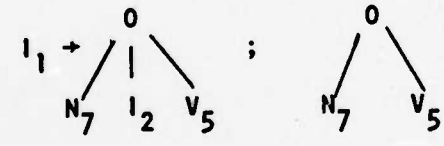

H5 → 1 V0 H6 V5 ; 1 V0 V5

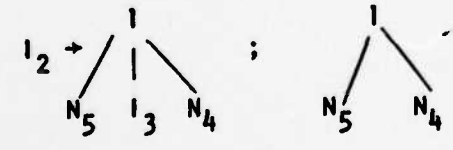

H6 → 1 V0 H7 N5 ; 1 V0 N5

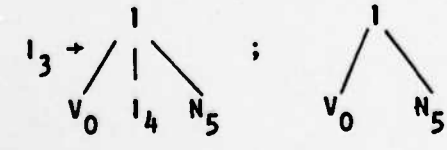

H7 → 1 V0 H8 V4 ; 1 V0 V4

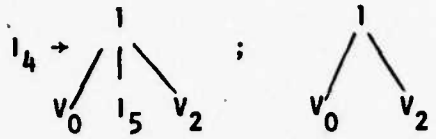

H8 → 0 V2 H9 N7 ; 0 V2 N7

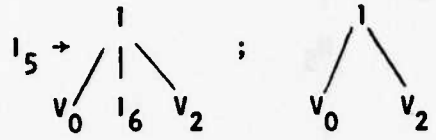

H9 → 1 N4 H1 V3 ; 1 N4 V3

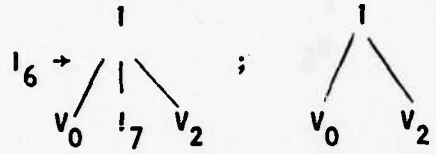

I1 → 0 N7 I2 V5 ; 0 N7 V5

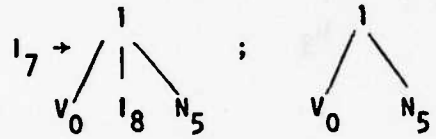

I2 → 1 N5 I3 N4 ; 1 N5 N4

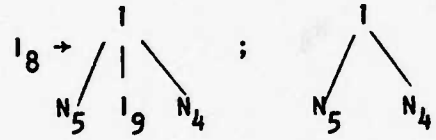

I3 → 1 V0 I4 N5 ; 1 V0 N5

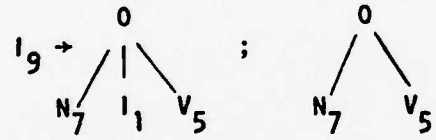

I4 →
1
V0 I5 V2
;
1
V0 V2

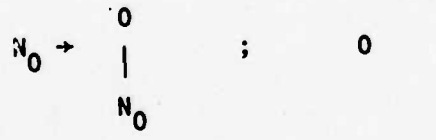

I5 →
1
V0 I6 V2
;
1
V0 V2


I6 →
1
V0 I7 V2
;
1
V0 V2


I7 →
1
V0 I8 N5
;
1
V0 N5


I8 →
1
N5 I9 N4
;
1
N5 N4

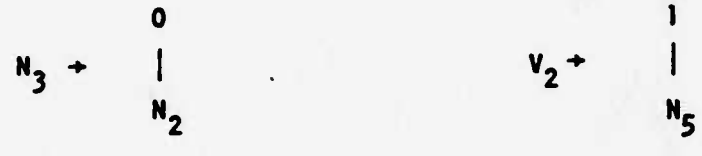

I9 →
0
N7 I1 V5
;
0
N7 V5


N0 →
0
N0
;
0


N1 →
1
N0
;
1


N2 →
0
N1


V1 → 0 | V0

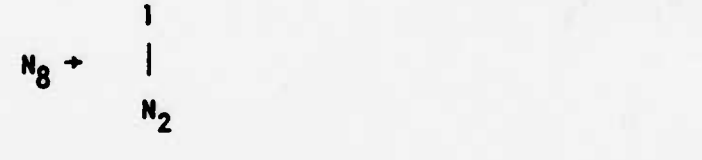

N3 → 0 | N2
V2 → 1 | N5


N4 → 0 | N3
V3 → 1 | N6

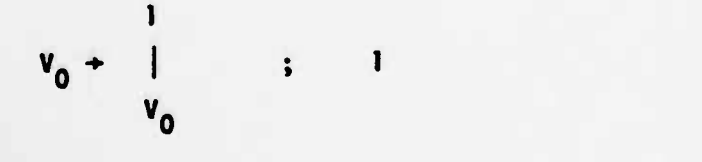

N5 → 1 | N1
V4 → 1 | N3


N6 → 0 | N5
V5 → 1 | N8


N7 → 0 | N6


N8 → 1 | N2


N9 → 0 | N8


V0 → 1 | V0
;
1

APPENDIX F

DISCRIMINATION GRAMMARS FOR PATTERN D22, D34, D38, and D68

F.1. Grammar for Synthesis and Discrimination of Water Pattern (D38)

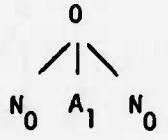
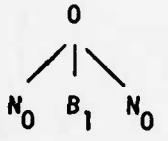
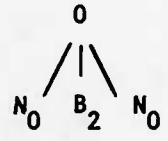
$G_6 = (V_6, r, P_6, S_6)$ over $\langle \Sigma, r \rangle$

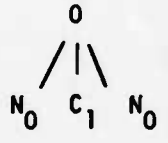
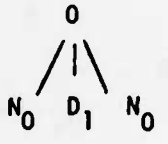
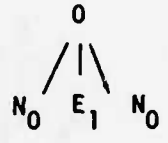
$V_6 = S_6 \cup \{N_{0,1,\ldots,9}, V_{0,1,\ldots,5}\} \cup \Sigma$

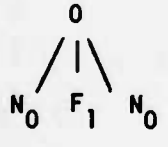
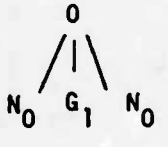
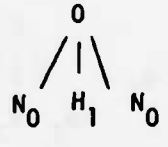
$S_6 = \{A_{0,1}, B_{1,2,\ldots,7}, C_{1,2,\ldots,6}, D_{1,2,\ldots,7}, E_{1,2,\ldots,5}, F_{1,2,\ldots,5}, G_{1,2,3}$

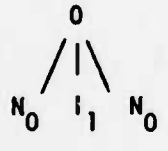
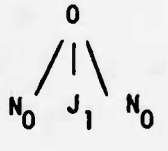
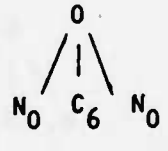
$H_{1,2,3}, I_1, J_1\}$

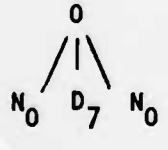
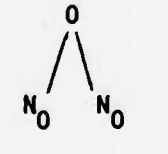
P_6:

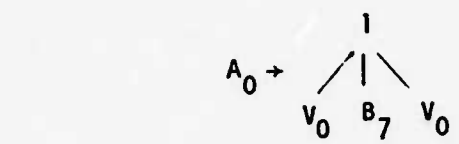
$A_1 \rightarrow$  , 0.1 ;
 , 0 ;
 , 0.1 ;

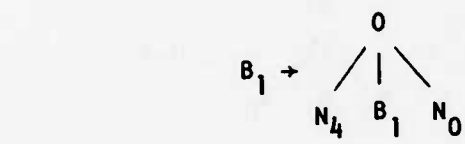
 , 0.1 ;
 , 0.1 ;
 , 0.1 ;

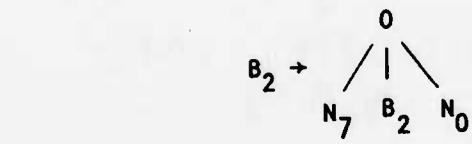
 , 0.1 ;
 , 0.1 ;
 , 0.1 ;

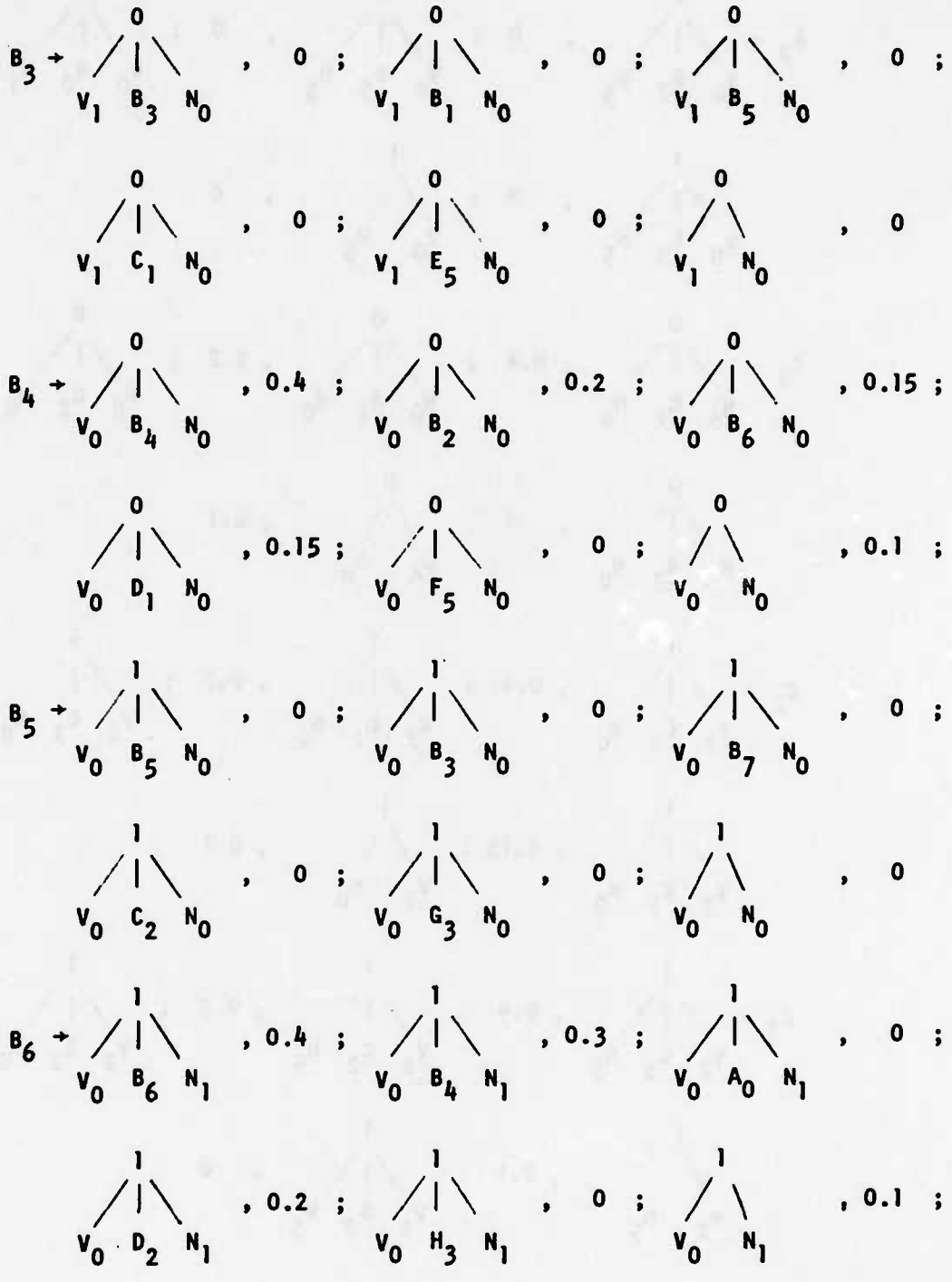
 , 0.1 ;
 , 0 ;
 , 0 ;

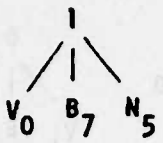
 , 0 ;
 , 0.1

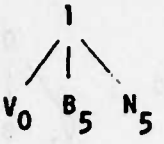
$A_0 \rightarrow$ 

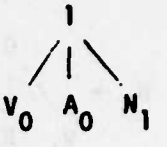
$B_1 \rightarrow$ 

$B_2 \rightarrow$ 

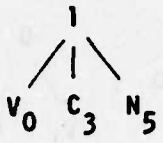

B3 → 0 V1 B3 N0 , 0 ; 0 V1 B1 N0 , 0 ; 0 V1 B5 N0 , 0 ;
0 V1 C1 N0 , 0 ; 0 V1 E5 N0 , 0 ; 0 V1 N0 , 0
B4 → 0 V0 B4 N0 , 0.4 ; 0 V0 B2 N0 , 0.2 ; 0 V0 B6 N0 , 0.15 ;
0 V0 D1 N0 , 0.15 ; 0 V0 F5 N0 , 0 ; 0 V0 N0 , 0.1 ;
B5 → 1 V0 B5 N0 , 0 ; 1 V0 B3 N0 , 0 ; 1 V0 B7 N0 , 0 ;
1 V0 C2 N0 , 0 ; 1 V0 G3 N0 , 0 ; 1 V0 N0 , 0
B6 → 1 V0 B6 N1 , 0.4 ; 1 V0 B4 N1 , 0.3 ; 1 V0 A0 N1 , 0 ;
1 V0 D2 N1 , 0.2 ; 1 V0 H3 N1 , 0 ; 1 V0 N1 , 0.1 ;

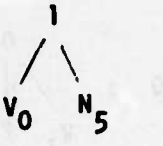
$B_7 \rightarrow$ 

, 0 ; 

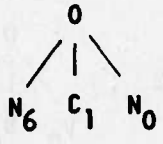
, 0 ; 

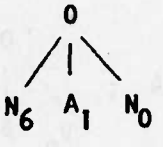
, 0 ;

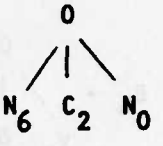


, 0 ; 

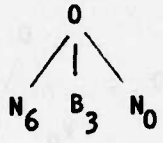
, 0

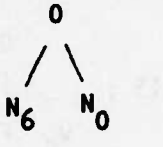
$C_1 \rightarrow$ 

, 0.4 ; 

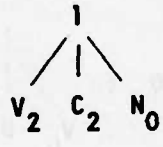
, 0.2 ; 

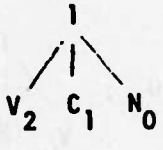
, 0.3 ;

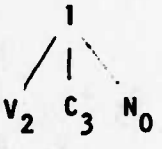


, 0 ; 

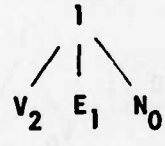
, 0.1

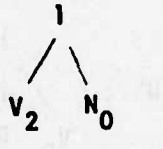
$C_2 \rightarrow$ 

, 0.4 ; 

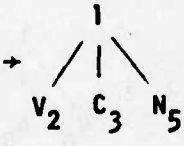
, 0.2 ; 

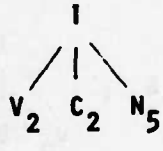
, 0.15 ;

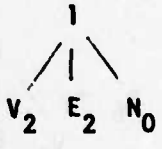


, 0.15 ; 

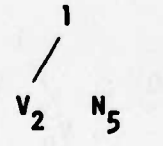
, 0.1

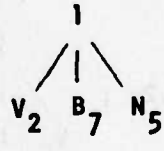
$C_3 \rightarrow$ 

, 0.4 ; 

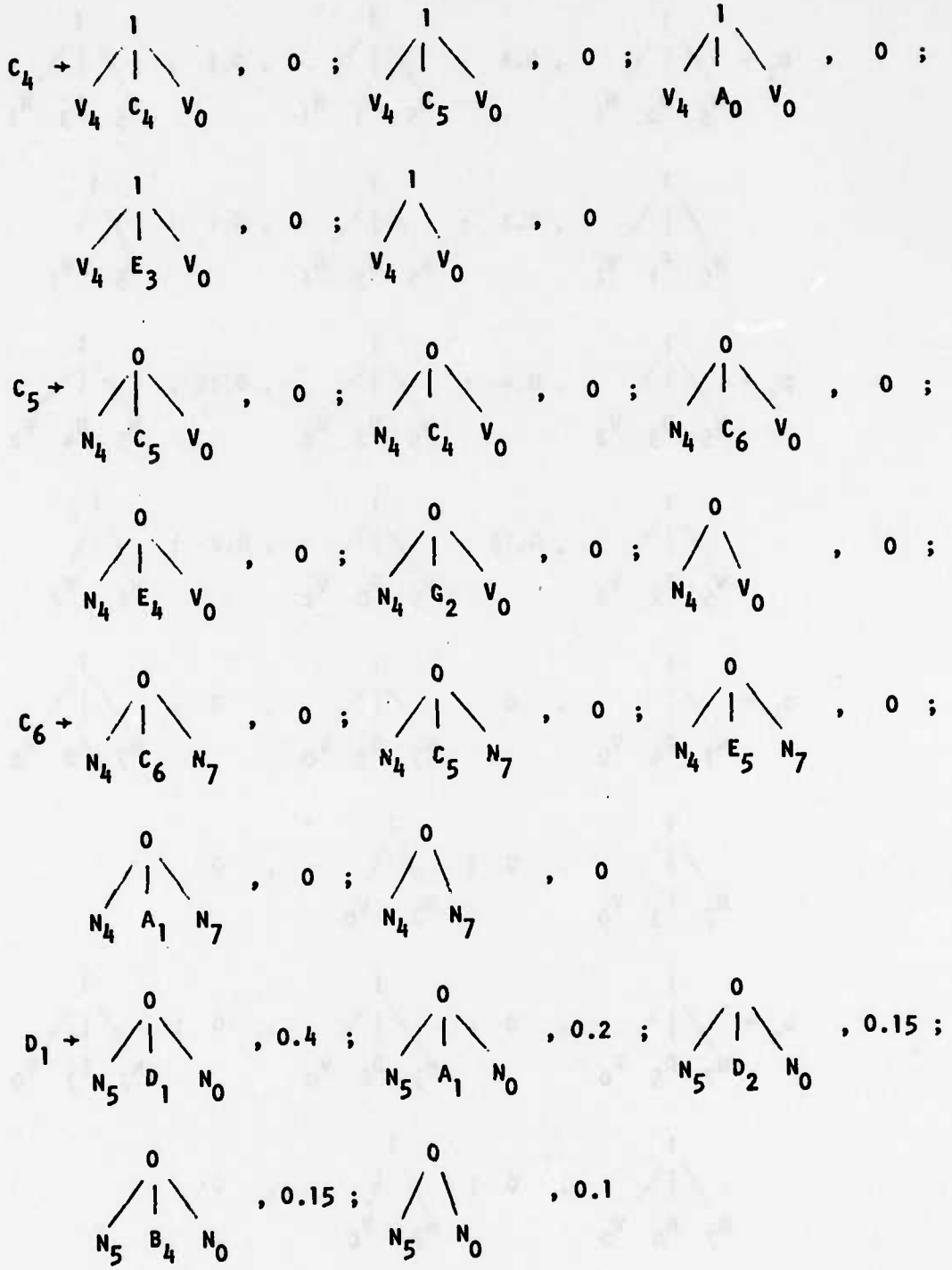
, 0.3 ; 

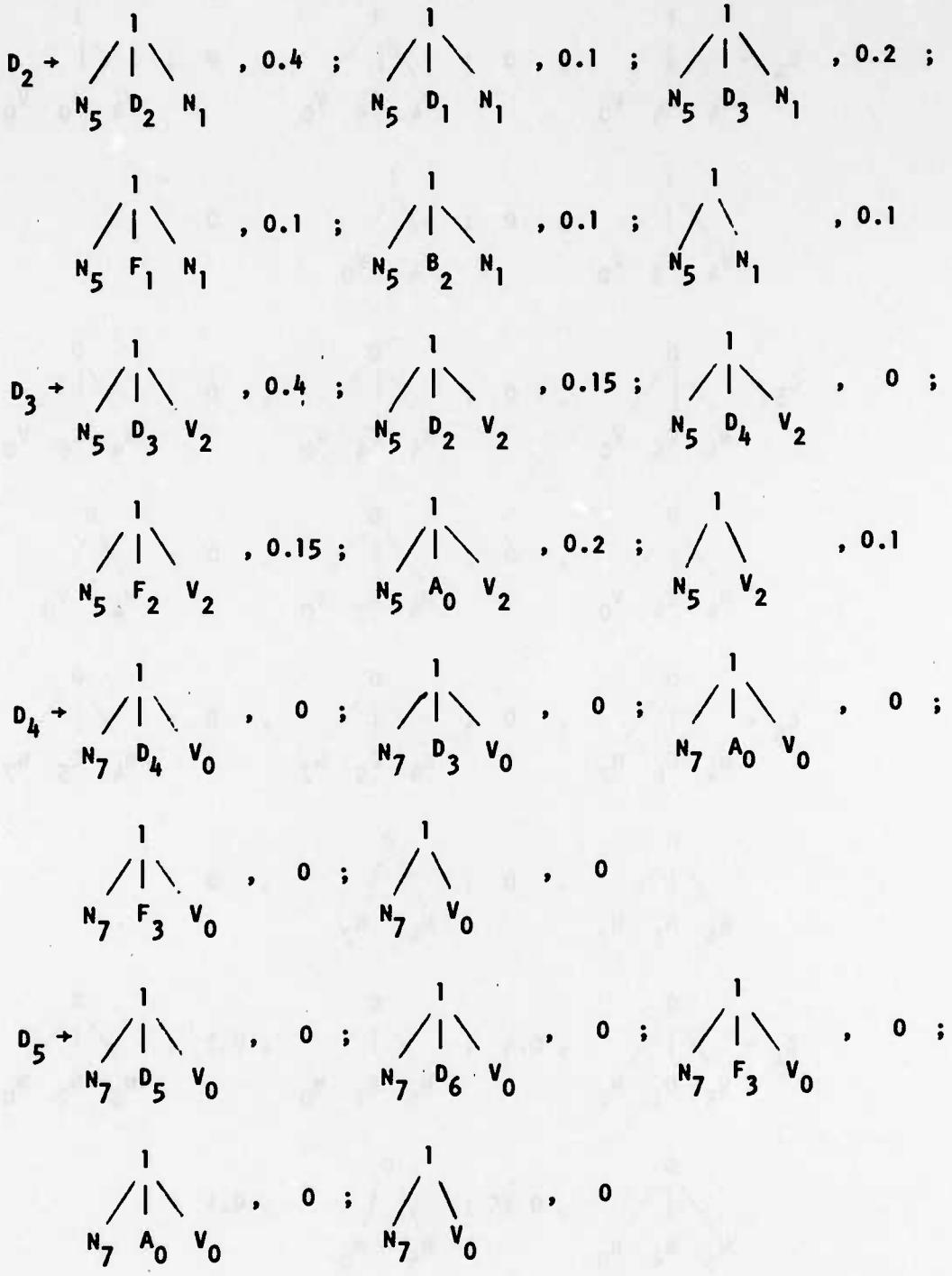
, 0.3 ;

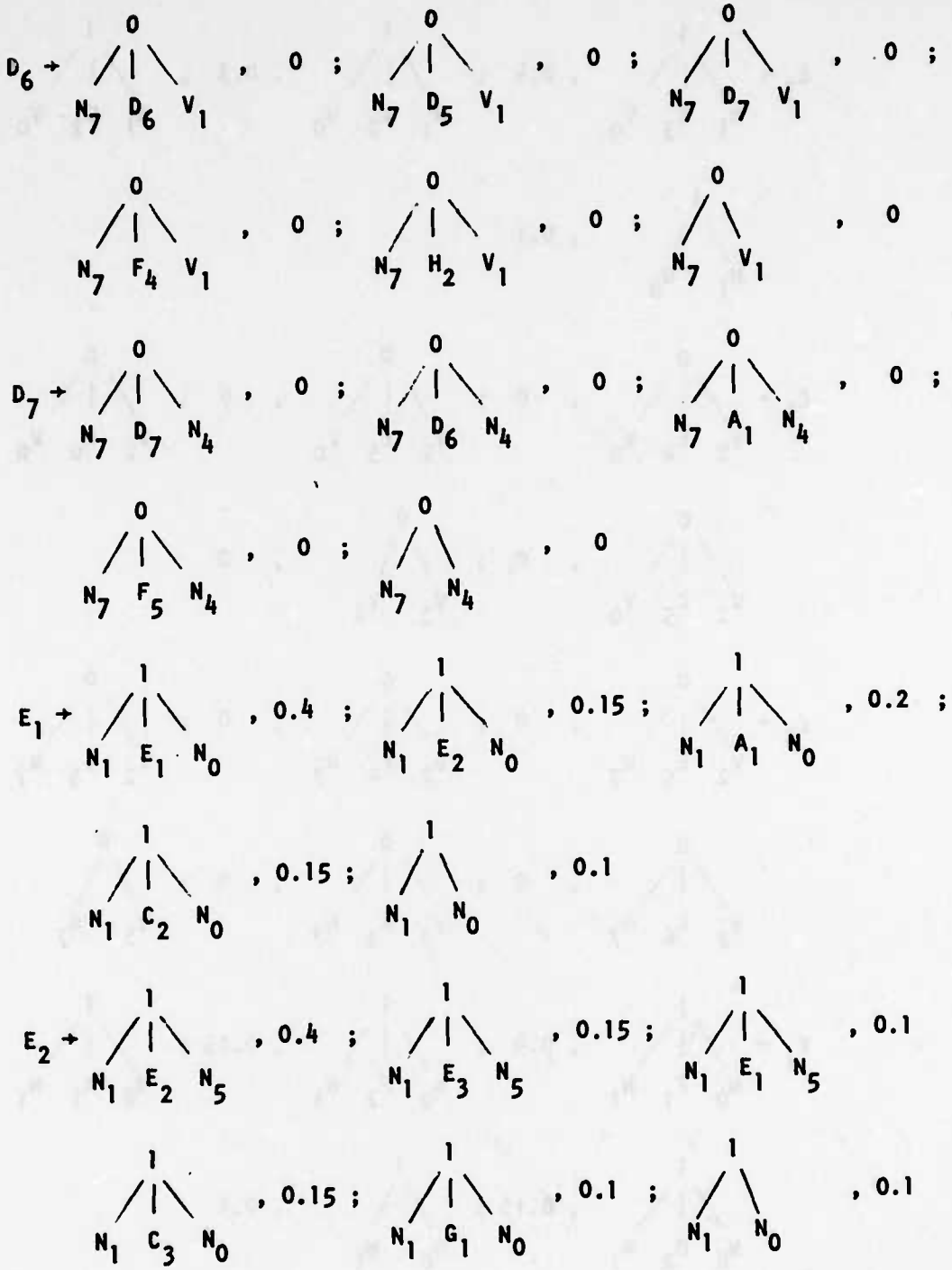


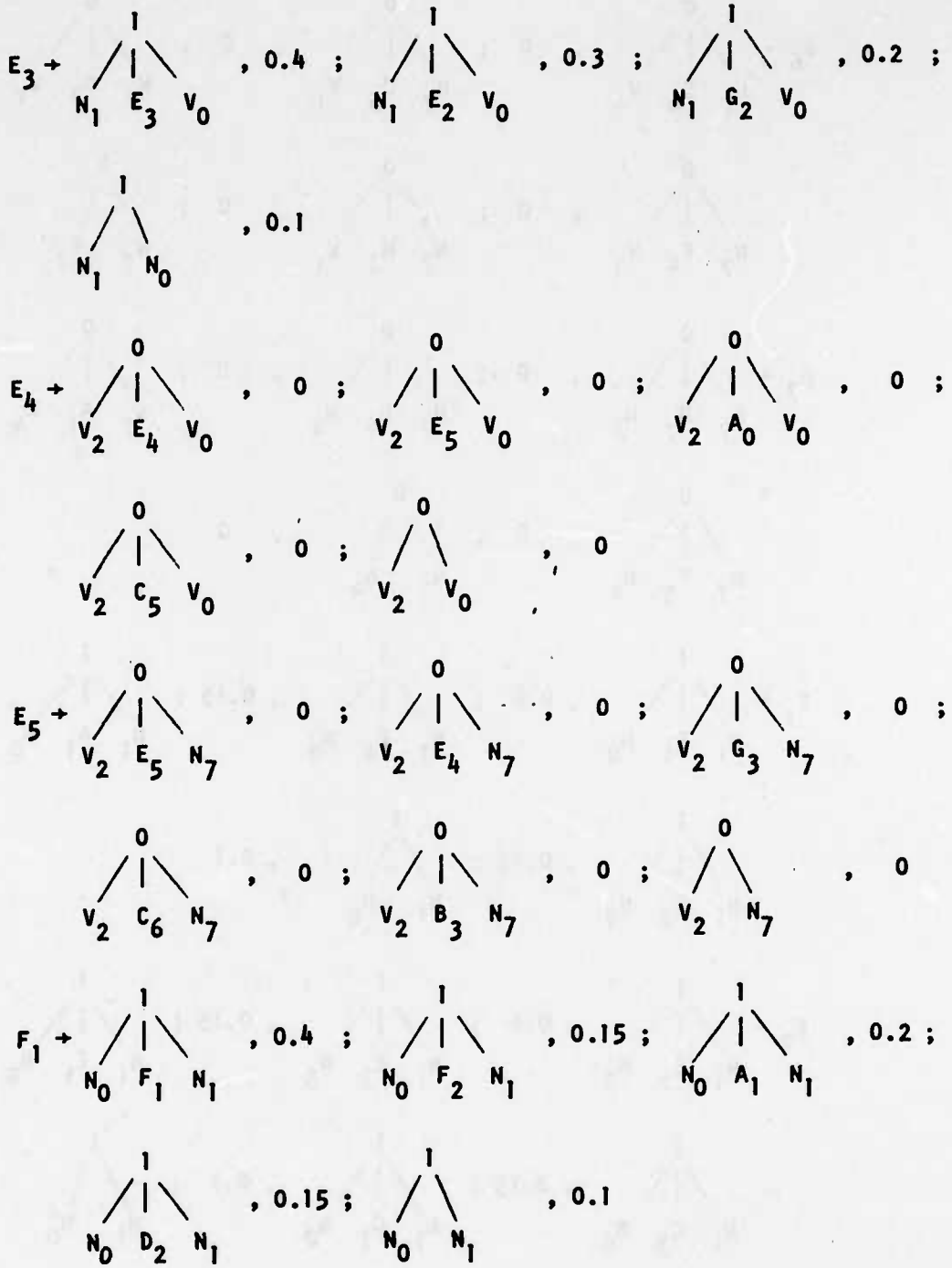
, 0.1 ; 

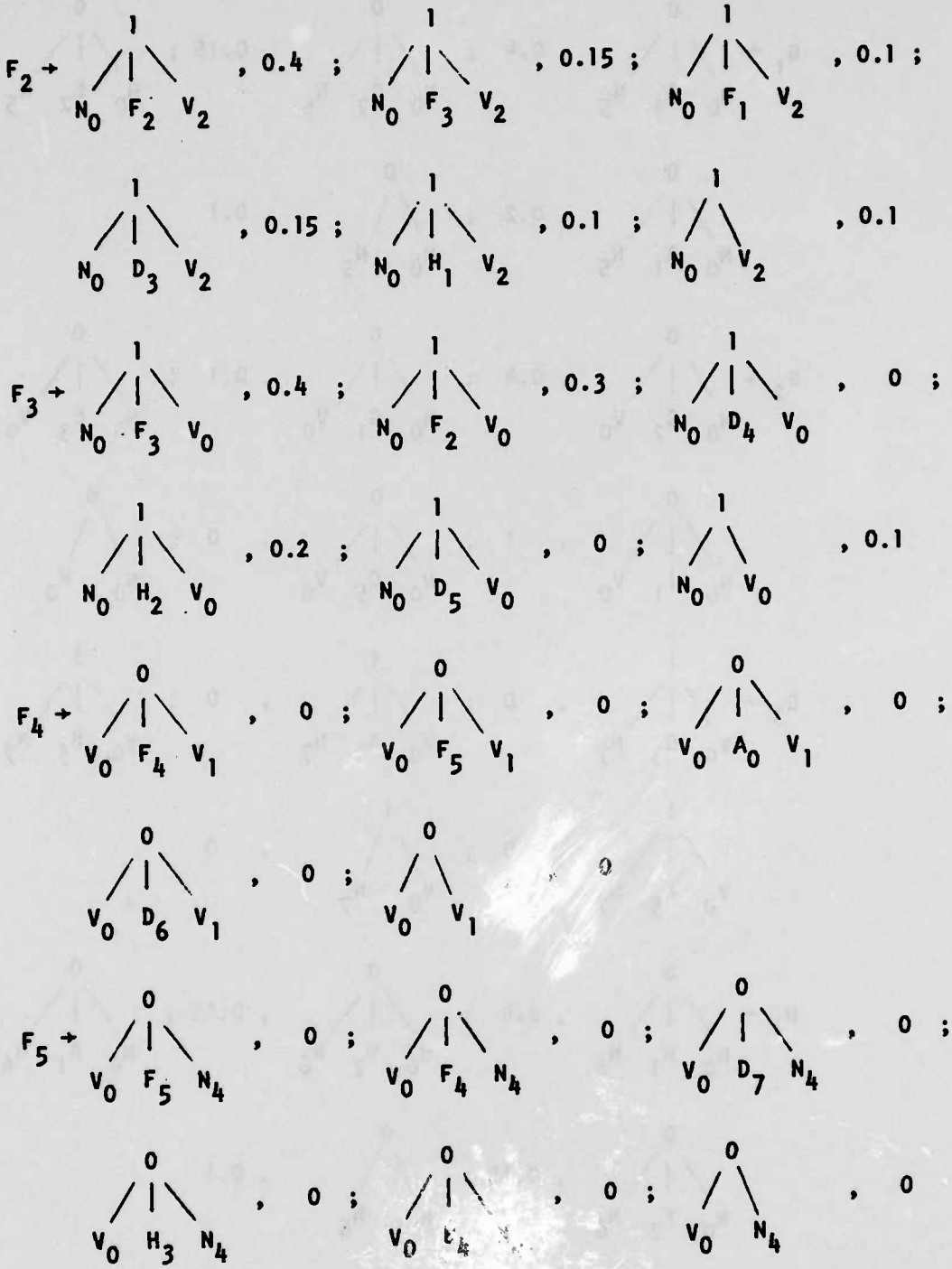
, 0

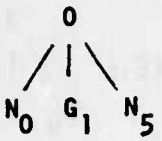
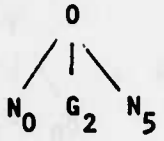
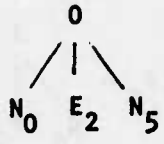

C4 → 1 (V4 C4 V0) , 0 ; 1 (V4 C5 V0) , 0 ; 1 (V4 A0 V0) , 0 ;
1 (V4 E3 V0) , 0 ; 1 (V4 V0) , 0
C5 → 0 (N4 C5 V0) , 0 ; 0 (N4 C4 V0) , 0 ; 0 (N4 C6 V0) , 0 ;
0 (N4 E4 V0) , 0 ; 0 (N4 G2 V0) , 0 ; 0 (N4 V0) , 0 ;
C6 → 0 (N4 C6 N7) , 0 ; 0 (N4 C5 N7) , 0 ; 0 (N4 E5 N7) , 0 ;
0 (N4 A1 N7) , 0 ; 0 (N4 N7) , 0
D1 → 0 (N5 D1 N0) , 0.4 ; 0 (N5 A1 N0) , 0.2 ; 0 (N5 D2 N0) , 0.15 ;
0 (N5 B4 N0) , 0.15 ; 0 (N5 N0) , 0.1

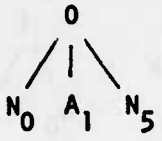
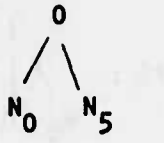

D2 → 1 N5 D2 N1 , 0.4 ; 1 N5 D1 N1 , 0.1 ; 1 N5 D3 N1 , 0.2 ;
1 N5 F1 N1 , 0.1 ; 1 N5 B2 N1 , 0.1 ; 1 N5 N1 , 0.1
D3 → 1 N5 D3 V2 , 0.4 ; 1 N5 D2 V2 , 0.15 ; 1 N5 D4 V2 , 0 ;
1 N5 F2 V2 , 0.15 ; 1 N5 A0 V2 , 0.2 ; 1 N5 V2 , 0.1
D4 → 1 N7 D4 V0 , 0 ; 1 N7 D3 V0 , 0 ; 1 N7 A0 V0 , 0 ;
1 N7 F3 V0 , 0 ; 1 N7 V0 , 0
D5 → 1 N7 D5 V0 , 0 ; 1 N7 D6 V0 , 0 ; 1 N7 F3 V0 , 0 ;
1 N7 A0 V0 , 0 ; 1 N7 V0 , 0

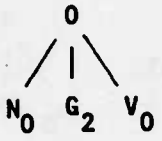
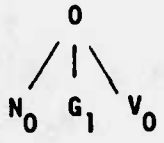
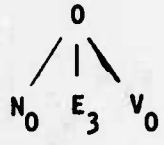

D6 → 0 N7 D6 V1 , 0 ; 0 N7 D5 V1 , 0 ; 0 N7 D7 V1 , 0 ;
0 N7 F4 V1 , 0 ; 0 N7 H2 V1 , 0 ; 0 N7 V1 , 0
D7 → 0 N7 D7 N4 , 0 ; 0 N7 D6 N4 , 0 ; 0 N7 A1 N4 , 0 ;
0 N7 F5 N4 , 0 ; 0 N7 N4 , 0
E1 → 1 N1 E1 N0 , 0.4 ; 1 N1 E2 N0 , 0.15 ; 1 N1 A1 N0 , 0.2 ;
1 N1 C2 N0 , 0.15 ; 1 N1 N0 , 0.1
E2 → 1 N1 E2 N5 , 0.4 ; 1 N1 E3 N5 , 0.15 ; 1 N1 E1 N5 , 0.1
1 N1 C3 N0 , 0.15 ; 1 N1 G1 N0 , 0.1 ; 1 N1 N0 , 0.1

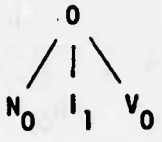
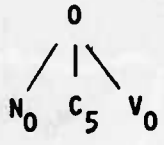
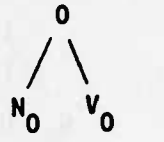

E3 → (1 N1 E3 V0) , 0.4 ; (1 N1 E2 V0) , 0.3 ; (1 N1 G2 V0) , 0.2 ;
(1 N1 N0) , 0.1
E4 → (0 V2 E4 V0) , 0 ; (0 V2 E5 V0) , 0 ; (0 V2 A0 V0) , 0 ;
(0 V2 C5 V0) , 0 ; (0 V2 V0) , 0
E5 → (0 V2 E5 N7) , 0 ; (0 V2 E4 N7) , 0 ; (0 V2 G3 N7) , 0 ;
(0 V2 C6 N7) , 0 ; (0 V2 B3 N7) , 0 ; (0 V2 N7) , 0
F1 → (1 N0 F1 N1) , 0.4 ; (1 N0 F2 N1) , 0.15 ; (1 N0 A1 N1) , 0.2 ;
(1 N0 D2 N1) , 0.15 ; (1 N0 N1) , 0.1

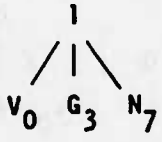
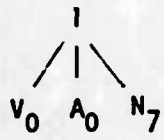

F2 → 1 N0 F2 V2 , 0.4 ; 1 N0 F3 V2 , 0.15 ; 1 N0 F1 V2 , 0.1 ;
1 N0 D3 V2 , 0.15 ; 1 N0 H1 V2 , 0.1 ; 1 N0 V2 , 0.1
F3 → 1 N0 F3 V0 , 0.4 ; 1 N0 F2 V0 , 0.3 ; 1 N0 D4 V0 , 0 ;
1 N0 H2 V0 , 0.2 ; 1 N0 D5 V0 , 0 ; 1 N0 V0 , 0.1
F4 → 0 V0 F4 V1 , 0 ; 0 V0 F5 V1 , 0 ; 0 V0 A0 V1 , 0 ;
0 V0 D6 V1 , 0 ; 0 V0 V1
F5 → 0 V0 F5 N4 , 0 ; 0 V0 F4 N4 , 0 ; 0 V0 D7 N4 , 0 ;
0 V0 H3 N4 , 0 ; 0 V0 [illegible] , 0 ; 0 V0 N4 , 0

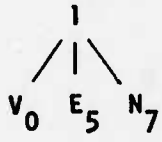
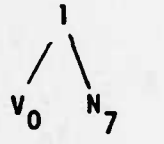
$G_1 \rightarrow$  , 0.4 ;  , 0.15 ;  , 0.15 ;

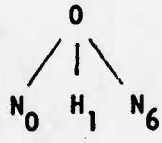
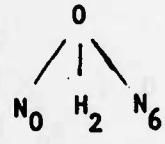
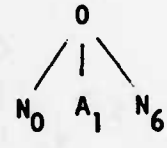
 , 0.2 ;  , 0.1

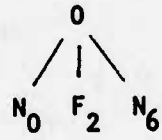
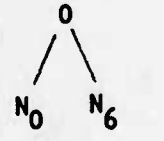
$G_2 \rightarrow$  , 0.4 ;  , 0.1 ;  , 0.3 ;

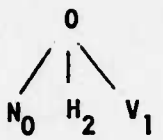
 , 1 ;  , 0 ;  , 0.1

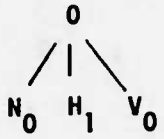
$G_3 \rightarrow$  , 0 ;  , 0 ;  , 0 ;

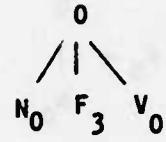
 , 0 ;  , 0

$H_1 \rightarrow$  , 0.4 ;  , 0.15 ;  , 0.2 ;

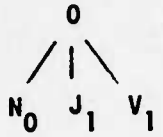
 , 0.15 ;  , 0.1

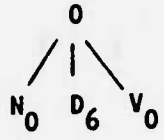
$H_2 \rightarrow$ 

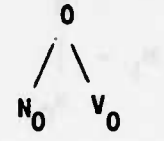
, 0.4 ; 

, 0.1 ; 

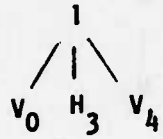
, 0.3 ;

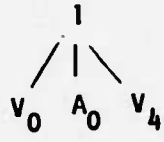


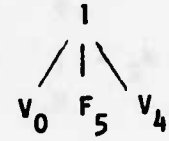
, 0.1 ; 

, 0 ; 

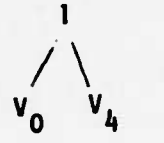
, 0.1

$H_3 \rightarrow$ 

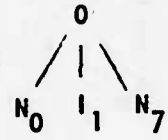
, 0 ; 

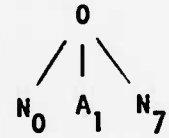
, 0 ; 

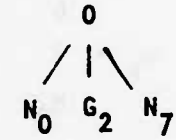
, 0 ;

, 0 ; 

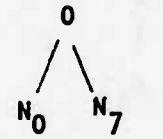
, 0

$I_1 \rightarrow$ 

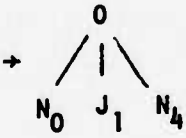
, 0.4 ; 

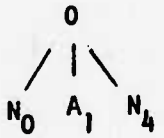
, 0.2 ; 

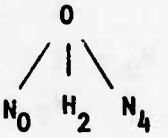
, 0.3 ;



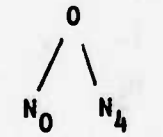
, 0.1

$J_1 \rightarrow$ 

, 0.4 ; 

, 0.2 ; 

, 0.3 ;



, 0.1

$$N_0 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \quad , \quad 0.9 \quad ; \quad 0 \quad , \quad 0.1$$

$$N_1 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \quad , \quad 0.9 \quad ; \quad 1 \quad , \quad 0.1$$

$$N_2 \rightarrow \begin{matrix} 0 \\ | \\ N_1 \end{matrix} \quad , \quad 1 \qquad\qquad N_3 \rightarrow \begin{matrix} 0 \\ | \\ N_2 \end{matrix} \quad , \quad 1$$

$$N_4 \rightarrow \begin{matrix} 0 \\ | \\ N_3 \end{matrix} \quad , \quad 1 \qquad\qquad N_5 \rightarrow \begin{matrix} 1 \\ | \\ N_1 \end{matrix} \quad , \quad 1$$

$$N_6 \rightarrow \begin{matrix} 0 \\ | \\ N_5 \end{matrix} \quad , \quad 1 \qquad\qquad N_7 \rightarrow \begin{matrix} 0 \\ | \\ N_6 \end{matrix} \quad , \quad 1$$

$$N_8 \rightarrow \begin{matrix} 1 \\ | \\ N_2 \end{matrix} \quad , \quad 1 \qquad\qquad N_9 \rightarrow \begin{matrix} 0 \\ | \\ N_8 \end{matrix} \quad , \quad 1$$

$$V_0 \rightarrow \begin{matrix} 1 \\ | \\ V_0 \end{matrix} \quad , \quad 0.9 \quad ; \quad 1 \quad , \quad 0.1$$

$$V_1 \rightarrow \begin{matrix} 0 \\ | \\ V_0 \end{matrix} \quad , \quad 1 \qquad\qquad V_2 \rightarrow \begin{matrix} 1 \\ | \\ N_5 \end{matrix} \quad , \quad 1$$

$$V_3 \rightarrow \begin{array}{|c} 1 \\ | \\ N_6 \end{array} , \quad 1 \qquad\qquad V_4 \rightarrow \begin{array}{|c} 1 \\ | \\ N_3 \end{array} , \quad 1$$

$$V_5 \rightarrow \begin{array}{|c} 1 \\ | \\ N_8 \end{array} , \quad 1$$

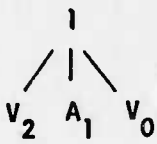
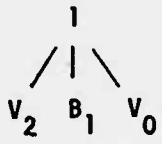
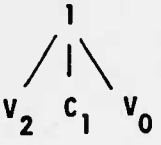
F.2. Discrimination Grammar for Pattern D22

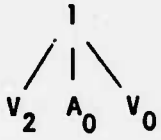
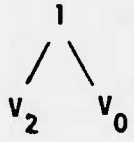
$G_{22} = (V_{22}, r, P_{22}, S_{22})$ over $<\Sigma, r>$

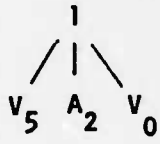
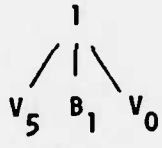
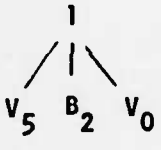
$V_{22} = S_{22} \cup \{N_{0,1,\ldots,9}, V_{0,1,\ldots,5}\} \cup \Sigma$

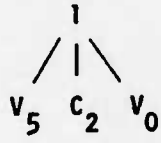
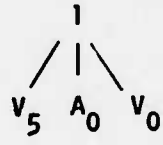
$S_{22} = A_{0,1,\ldots,9}, B_{1,2,\ldots,8}, C_{1,2,\ldots,7}, D_{1,2,\ldots,6}, E_{1,2,\ldots,6}, F_{1,2,\ldots,6}, G_{1,2,\ldots,6}, H_{1,2,\ldots,6}, I_{1,2,3}$.

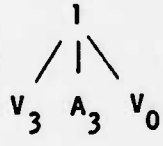
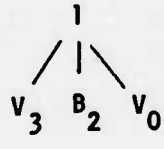
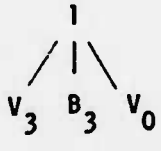
P_{22}:

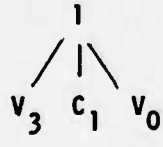
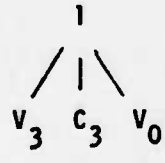
$A_1 \rightarrow$  ;  ;  ;

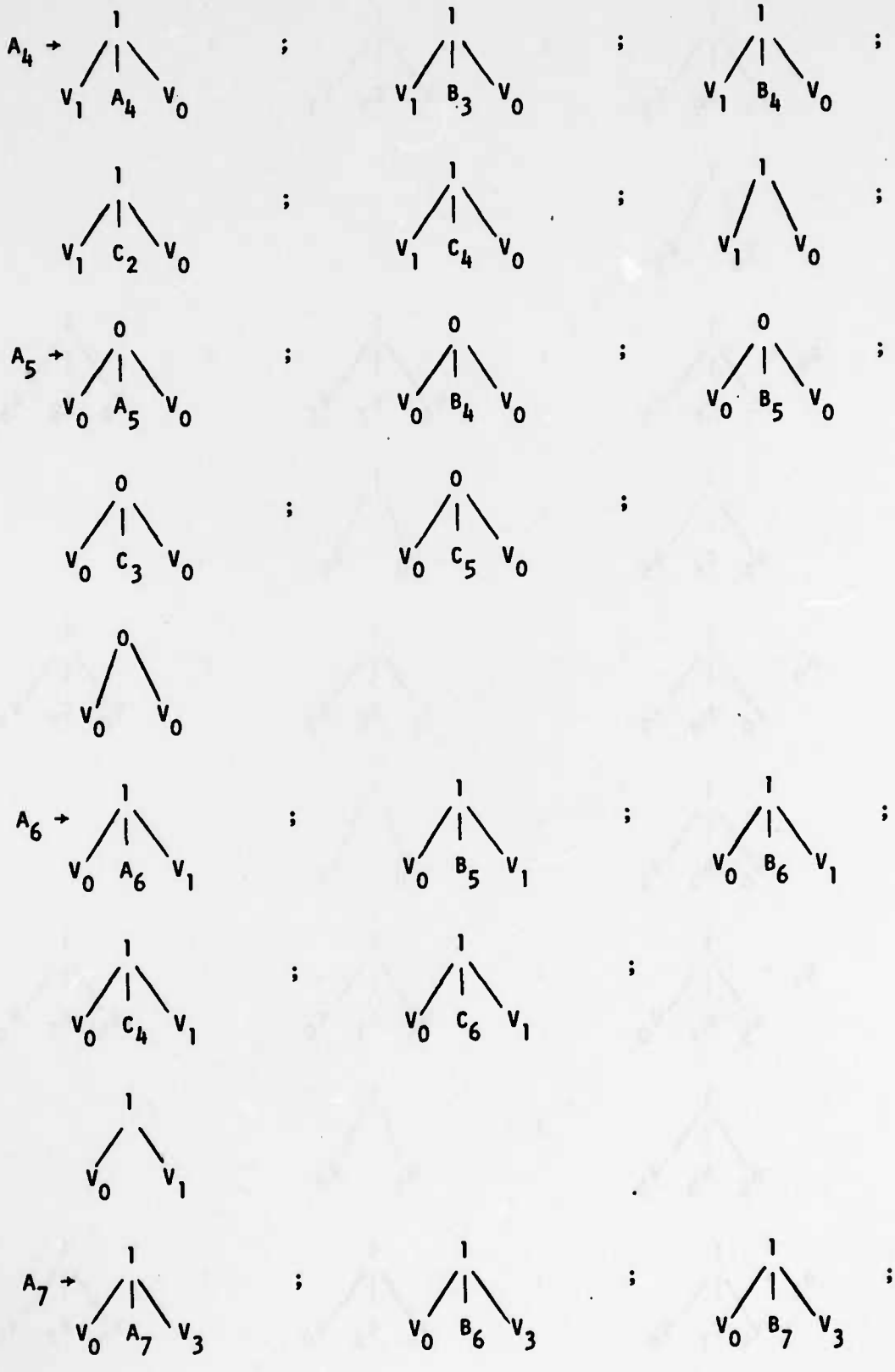
 ; 

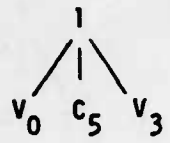
$A_2 \rightarrow$  ;  ;  ;

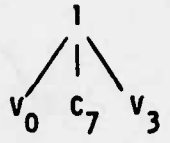
 ;  ; 

$A_3 \rightarrow$  ;  ;  ;

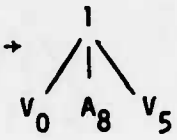
 ;  ; 

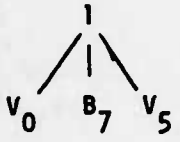

A4 → 1 V1 A4 V0 ; 1 V1 B3 V0 ; 1 V1 B4 V0 ;
1 V1 C2 V0 ; 1 V1 C4 V0 ; 1 V1 V0 ;
A5 → 0 V0 A5 V0 ; 0 V0 B4 V0 ; 0 V0 B5 V0 ;
0 V0 C3 V0 ; 0 V0 C5 V0 ;
0 V0 V0
A6 → 1 V0 A6 V1 ; 1 V0 B5 V1 ; 1 V0 B6 V1 ;
1 V0 C4 V1 ; 1 V0 C6 V1 ;
1 V0 V1
A7 → 1 V0 A7 V3 ; 1 V0 B6 V3 ; 1 V0 B7 V3 ;

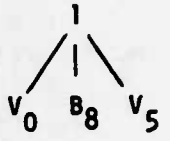


;


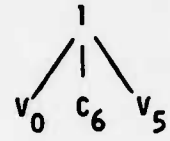
;

$A_8 \rightarrow$ 

;


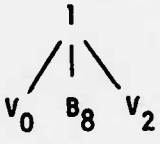
;


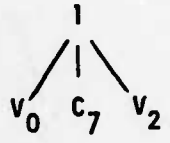
;



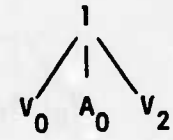
;

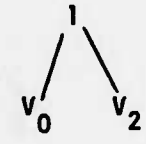

$A_9 \rightarrow$ 

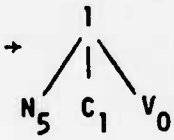
;


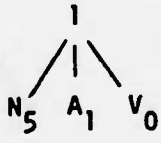
;


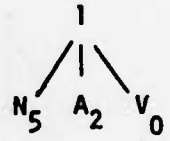
;



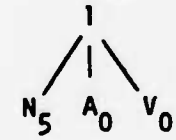
;


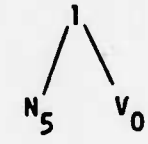
$B_1 \rightarrow$ 

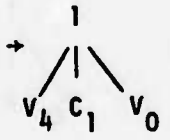
;


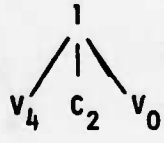
;


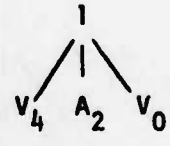
;



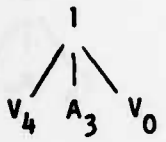
;


$B_2 \rightarrow$ 

;


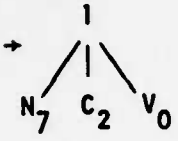
;


;

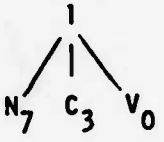


;

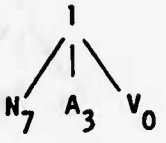
$B_3 \rightarrow$



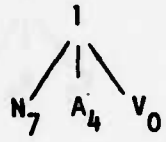
;



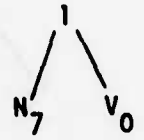
;



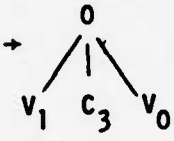
;



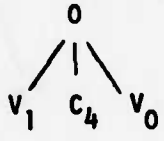
;



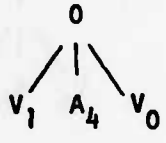
$B_4 \rightarrow$



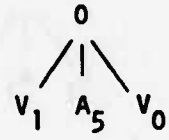
;



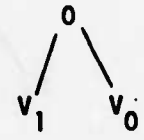
;



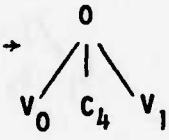
;



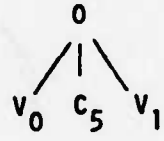
;



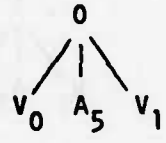
$B_5 \rightarrow$



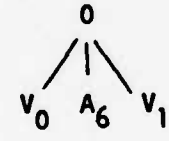
;



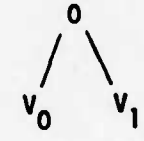
;



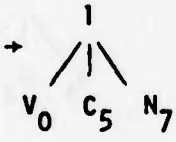
;



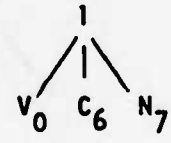
;



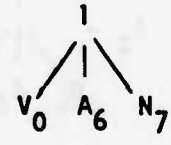
$B_6 \rightarrow$



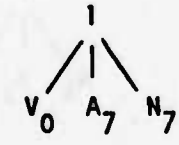
;



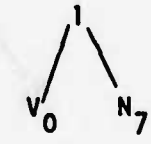
;

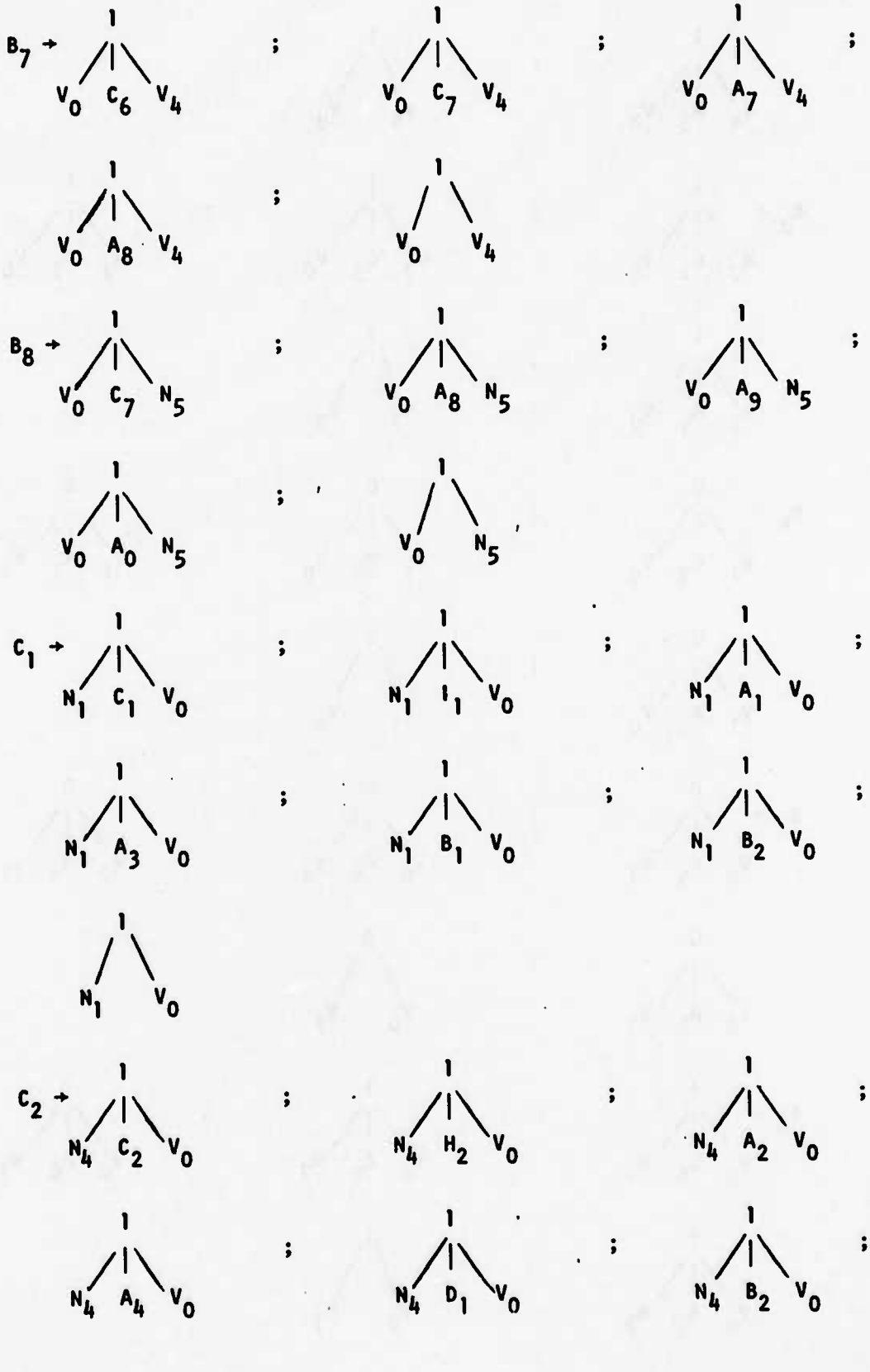


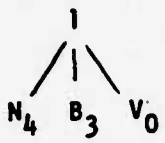
;



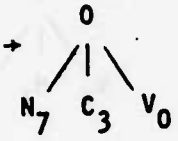
;

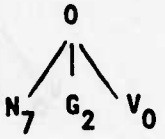


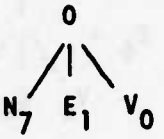

B7 →
1
V0 C6 V4
;
1
V0 C7 V4
;
1
V0 A7 V4
;
1
V0 A8 V4
;
1
V0 V4
;
B8 →
1
V0 C7 N5
;
1
V0 A8 N5
;
1
V0 A9 N5
;
1
V0 A0 N5
;
1
V0 N5
;
C1 →
1
N1 C1 V0
;
1
N1 I1 V0
;
1
N1 A1 V0
;
1
N1 A3 V0
;
1
N1 B1 V0
;
1
N1 B2 V0
;
1
N1 V0
C2 →
1
N4 C2 V0
;
1
N4 H2 V0
;
1
N4 A2 V0
;
1
N4 A4 V0
;
1
N4 D1 V0
;
1
N4 B2 V0
;



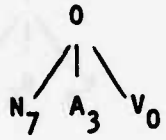
;

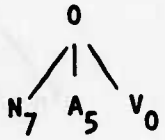

$C_3 \rightarrow$


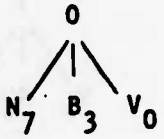
;


;


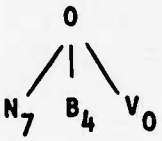
;

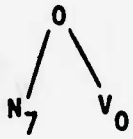


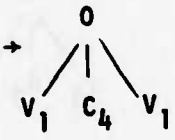
;


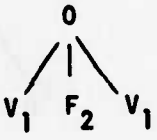
;


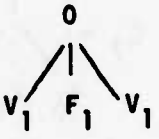
;



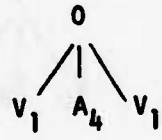
;


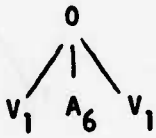
$C_4 \rightarrow$


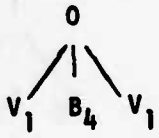
;


;


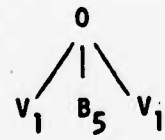
;

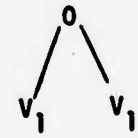


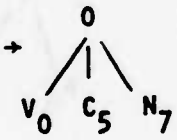
;


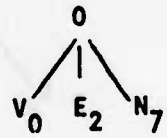
;


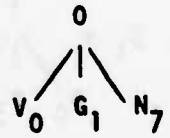
;



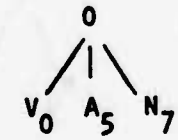
;


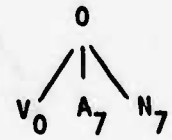
$C_5 \rightarrow$


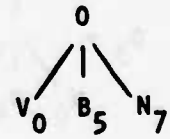
;


;


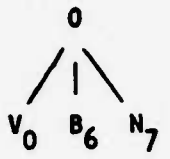
;



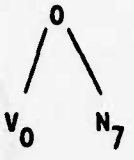
;


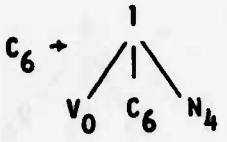
;


;

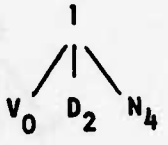

0
V0 B6 N7

;

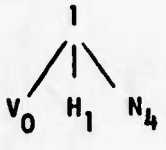

0
V0 N7


C6 →
1
V0 C6 N4

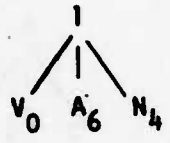
;


1
V0 D2 N4

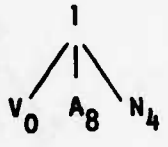
;


1
V0 H1 N4

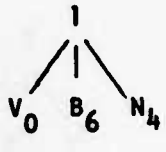
;


1
V0 A6 N4

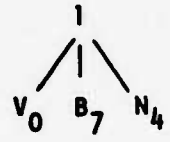
;


1
V0 A8 N4

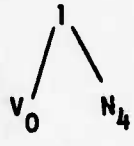
;

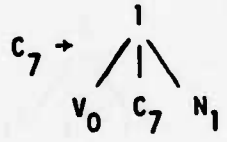

1
V0 B6 N4

;


1
V0 B7 N4

;

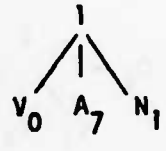

1
V0 N4


C7 →
1
V0 C7 N1

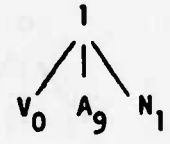
;


1
V0 I1 N1

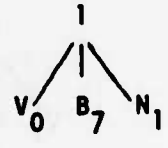
;


1
V0 A7 N1

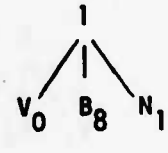
;


1
V0 A9 N1

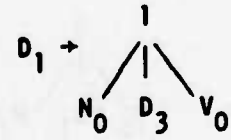
;


1
V0 B7 N1

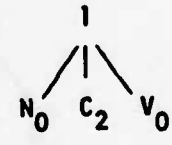
;


1
V0 B8 N1

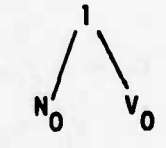

1
V0 N1

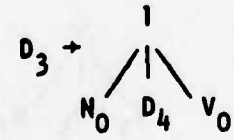

D1 →
1
N0 D3 V0

;

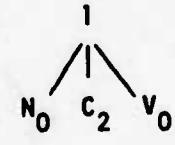

1
N0 C2 V0

;

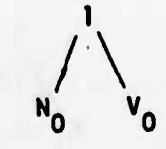

1
N0 V0

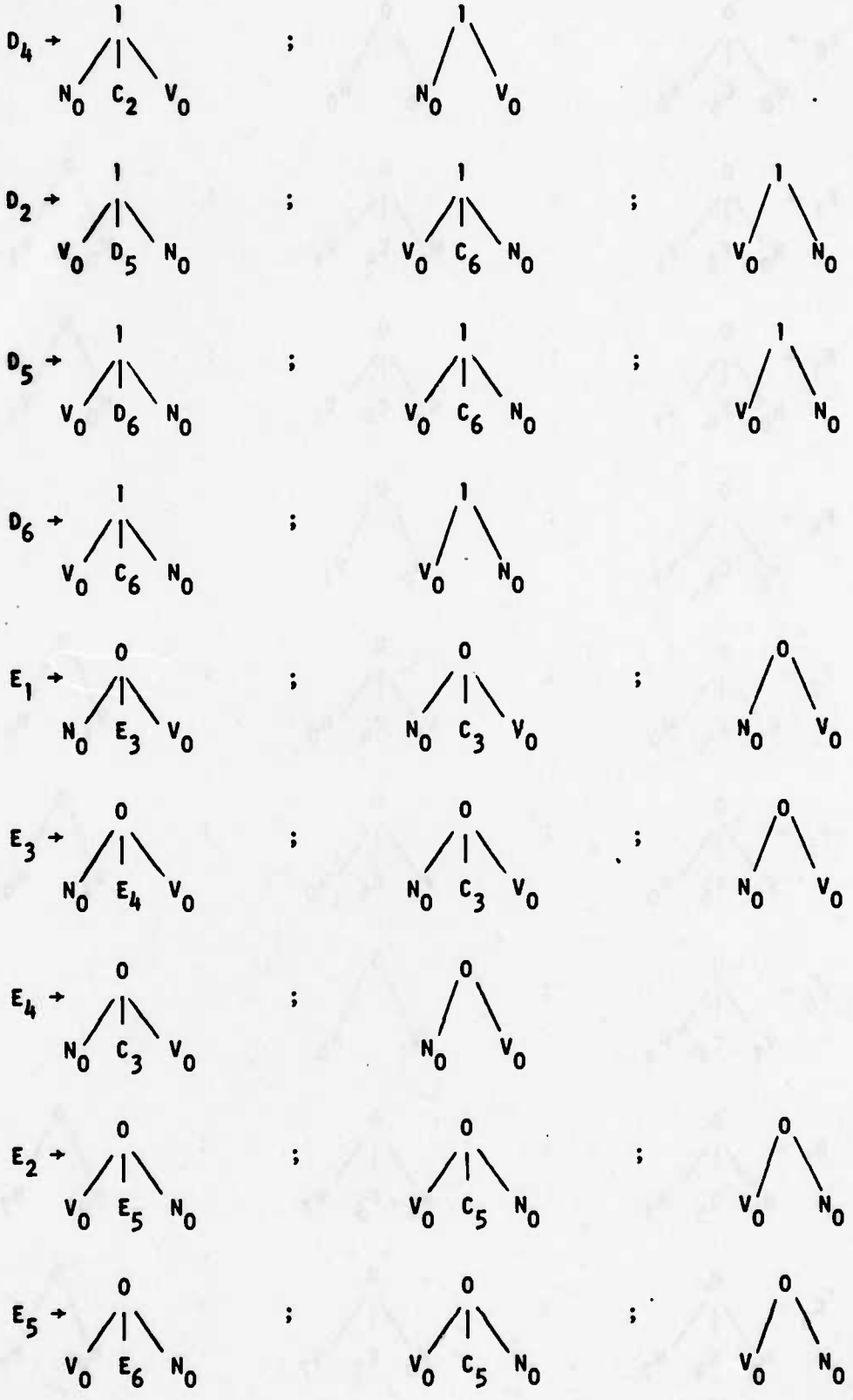

D3 →
1
N0 D4 V0

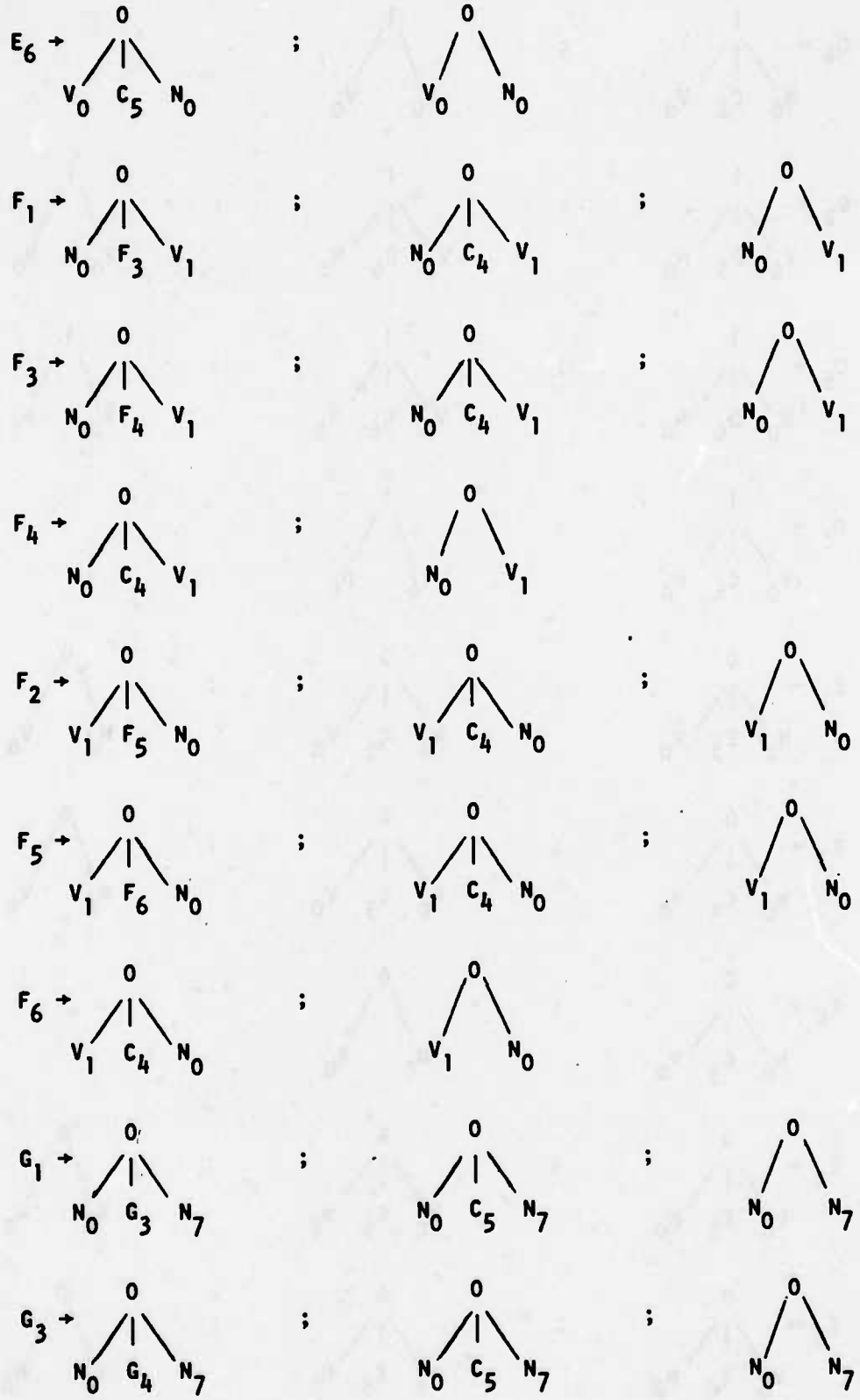
;

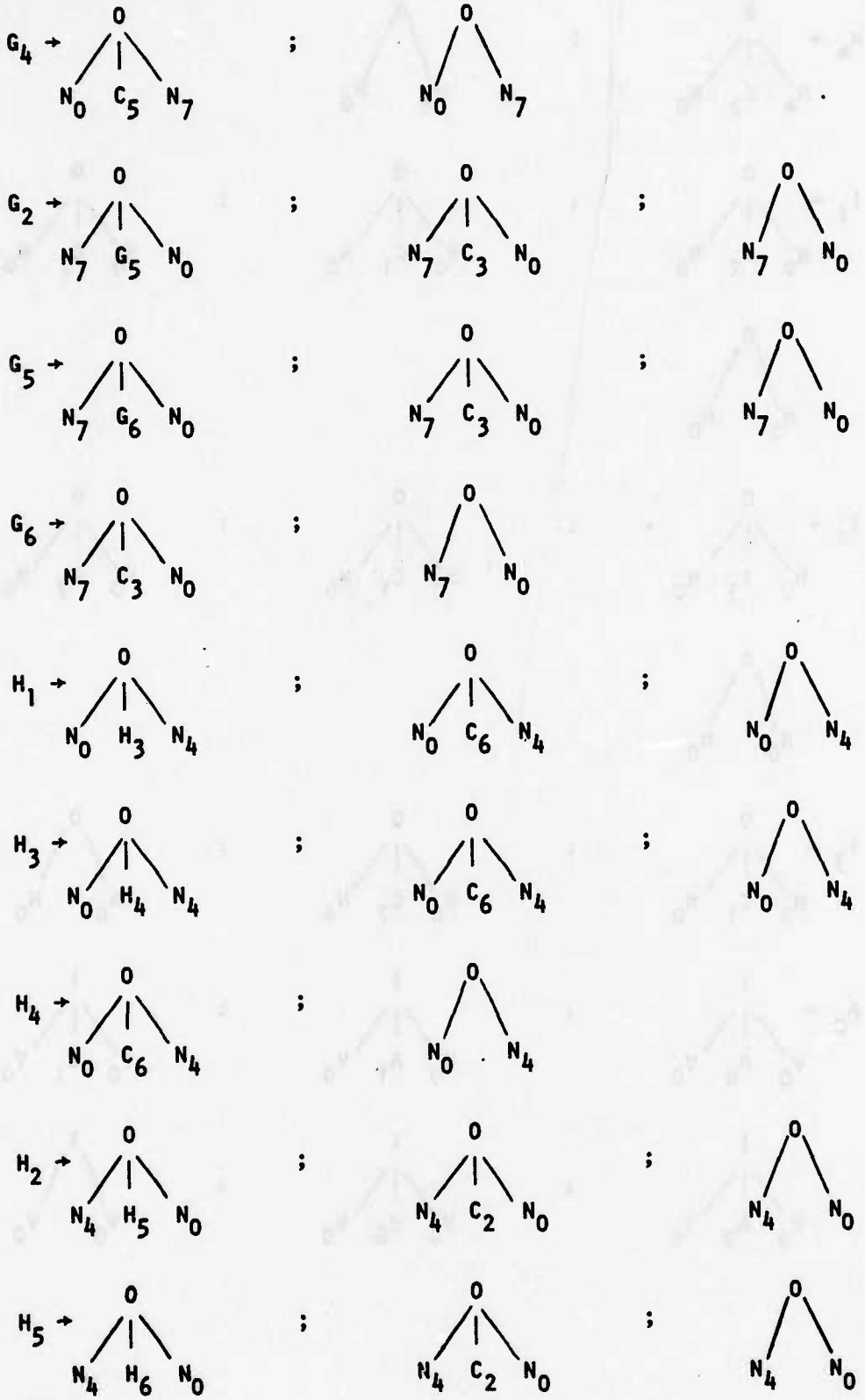

1
N0 C2 V0

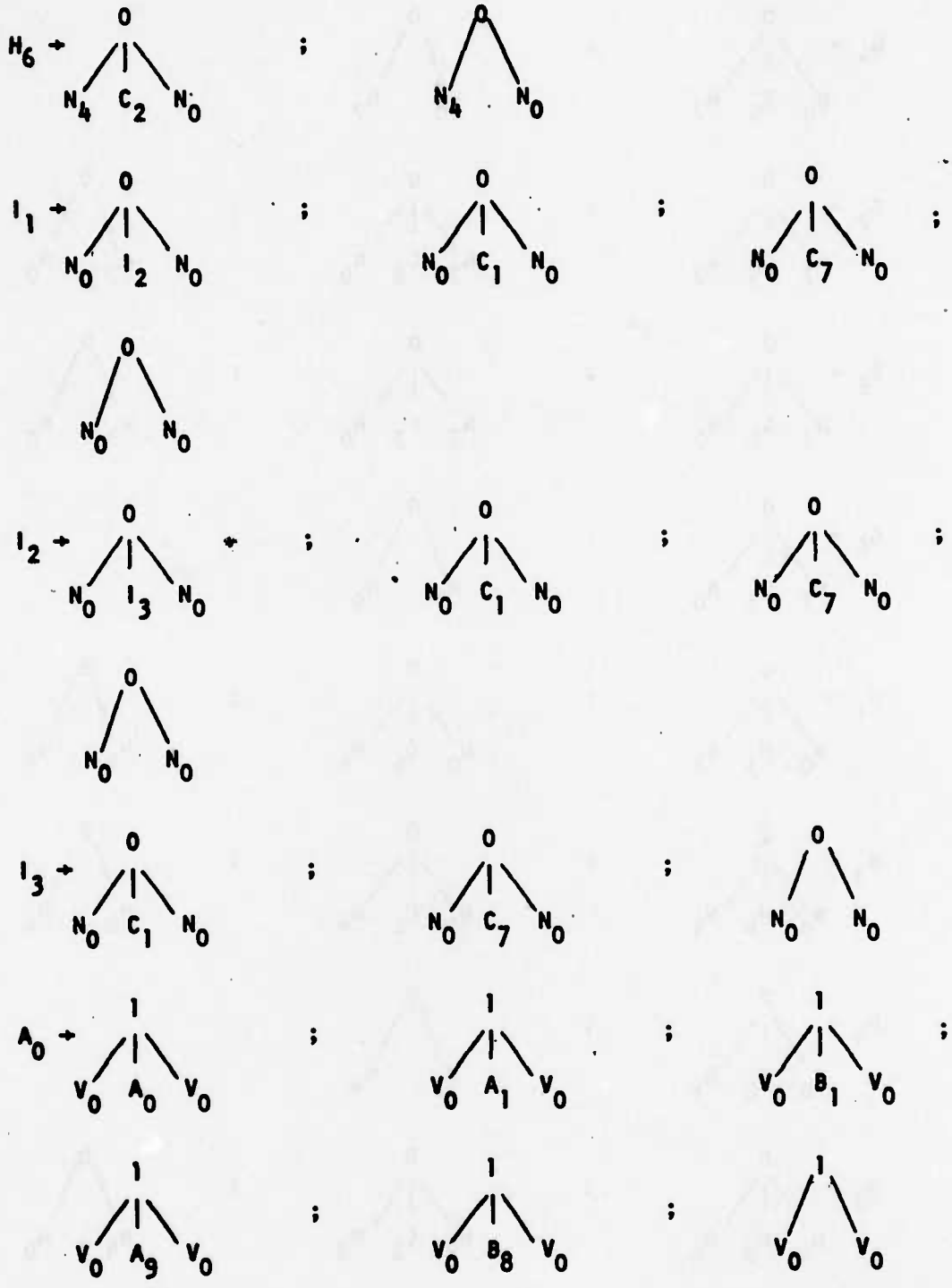
;


1
N0 V0


D4 → 1 N0 C2 V0 ; 1 N0 V0
D2 → 1 V0 D5 N0 ; 1 V0 C6 N0 ; 1 V0 N0
D5 → 1 V0 D6 N0 ; 1 V0 C6 N0 ; 1 V0 N0
D6 → 1 V0 C6 N0 ; 1 V0 N0
E1 → 0 N0 E3 V0 ; 0 N0 C3 V0 ; 0 N0 V0
E3 → 0 N0 E4 V0 ; 0 N0 C3 V0 ; 0 N0 V0
E4 → 0 N0 C3 V0 ; 0 N0 V0
E2 → 0 V0 E5 N0 ; 0 V0 C5 N0 ; 0 V0 N0
E5 → 0 V0 E6 N0 ; 0 V0 C5 N0 ; 0 V0 N0


E6 → O V0 C5 N0 ; O V0 N0
F1 → O N0 F3 V1 ; O N0 C4 V1 ; O N0 V1
F3 → O N0 F4 V1 ; O N0 C4 V1 ; O N0 V1
F4 → O N0 C4 V1 ; O N0 V1
F2 → O V1 F5 N0 ; O V1 C4 N0 ; O V1 N0
F5 → O V1 F6 N0 ; O V1 C4 N0 ; O V1 N0
F6 → O V1 C4 N0 ; O V1 N0
G1 → O N0 G3 N7 ; O N0 C5 N7 ; O N0 N7
G3 → O N0 G4 N7 ; O N0 C5 N7 ; O N0 N7


G4 → O N0 C5 N7 ; O N0 N7
G2 → O N7 G5 N0 ; O N7 C3 N0 ; O N7 N0
G5 → O N7 G6 N0 ; O N7 C3 N0 ; O N7 N0
G6 → O N7 C3 N0 ; O N7 N0
H1 → O N0 H3 N4 ; O N0 C6 N4 ; O N0 N4
H3 → O N0 H4 N4 ; O N0 C6 N4 ; O N0 N4
H4 → O N0 C6 N4 ; O N0 N4
H2 → O N4 H5 N0 ; O N4 C2 N0 ; O N4 N0
H5 → O N4 H6 N0 ; O N4 C2 N0 ; O N4 N0


H6 → 0 N4 C2 N0 ; 0 N4 N0
I1 → 0 N0 I2 N0 ; 0 N0 C1 N0 ; 0 N0 C7 N0 ; 0 N0 N0
I2 → 0 N0 I3 N0 ; 0 N0 C1 N0 ; 0 N0 C7 N0 ; 0 N0 N0
I3 → 0 N0 C1 N0 ; 0 N0 C7 N0 ; 0 N0 N0
A0 → 1 V0 A0 V0 ; 1 V0 A1 V0 ; 1 V0 B1 V0 ; 1 V0 A9 V0 ; 1 V0 B8 V0 ; 1 V0 V0

$$N_0 \rightarrow \begin{matrix} 0 \\ | \\ N_0 \end{matrix} \quad ; \quad 0$$

$$N_1 \rightarrow \begin{matrix} 1 \\ | \\ N_0 \end{matrix} \quad ; \quad 1$$

$$N_2 \rightarrow \begin{matrix} 0 \\ | \\ N_1 \end{matrix} \qquad N_3 \rightarrow \begin{matrix} 0 \\ | \\ N_2 \end{matrix}$$

$$N_4 \rightarrow \begin{matrix} 0 \\ | \\ N_3 \end{matrix} \qquad N_5 \rightarrow \begin{matrix} 1 \\ | \\ N_1 \end{matrix}$$

$$N_6 \rightarrow \begin{matrix} 0 \\ | \\ N_5 \end{matrix} \qquad N_7 \rightarrow \begin{matrix} 0 \\ | \\ N_6 \end{matrix}$$

$$N_8 \rightarrow \begin{matrix} 1 \\ | \\ N_2 \end{matrix} \qquad N_9 \rightarrow \begin{matrix} 0 \\ | \\ N_8 \end{matrix}$$

$$V_0 \rightarrow \begin{matrix} 1 \\ | \\ V_0 \end{matrix} \quad ; \quad 1$$

$$V_1 \rightarrow \begin{matrix} 0 \\ | \\ V_0 \end{matrix} \qquad V_2 \rightarrow \begin{matrix} 1 \\ | \\ N_5 \end{matrix}$$

$$V_3 \rightarrow \begin{matrix} 1 \\ | \\ N_6 \end{matrix}$$

$$V_4 \rightarrow \begin{matrix} 1 \\ | \\ N_3 \end{matrix}$$

$$V_5 \rightarrow \begin{matrix} 1 \\ | \\ N_8 \end{matrix}$$

F.3. **Discrimination Grammar for Pattern D34**

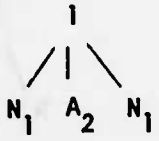
$G_{34} = (V_{34}, r, P_{34}, S_{34})$ over $\langle \Sigma, r \rangle$

$V_{34} = S \cup \Sigma \cup N_{0,1,\ldots,6}$.

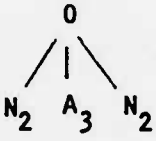
$S_{34} = \{A_{1,\ldots,8}, B_{0,1,\ldots,9}, C_{0,1,\ldots,9}, D_{0,1,\ldots,9}, E_{0,1,\ldots,9}, F_{0,1,\ldots,9}\}$

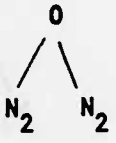
P_{34}:

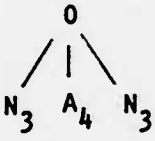
$A_1 \rightarrow$ 

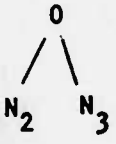
; 

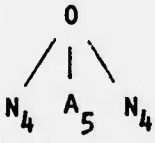
; 

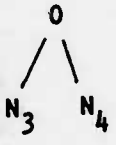
$A_2 \rightarrow$ 

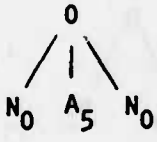
; 

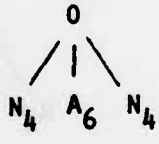
$A_3 \rightarrow$ 

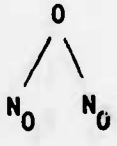
; 

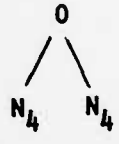
$A_4 \rightarrow$ 

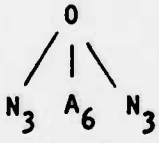
; 

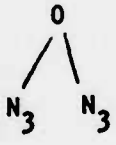
$A_5 \rightarrow$ 

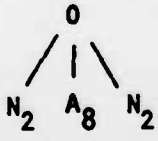
; 

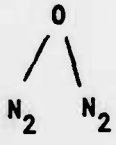
; 

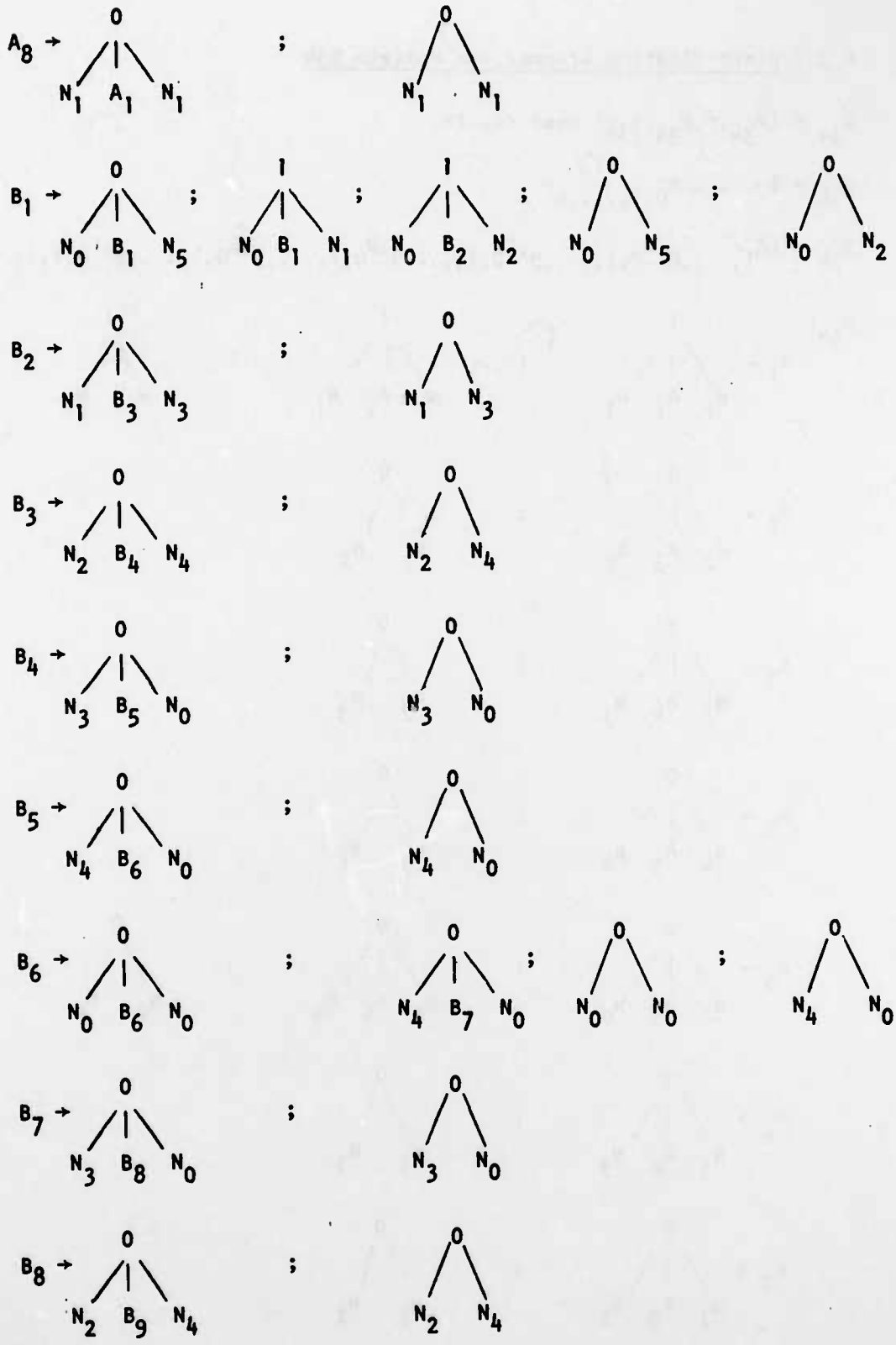
; 

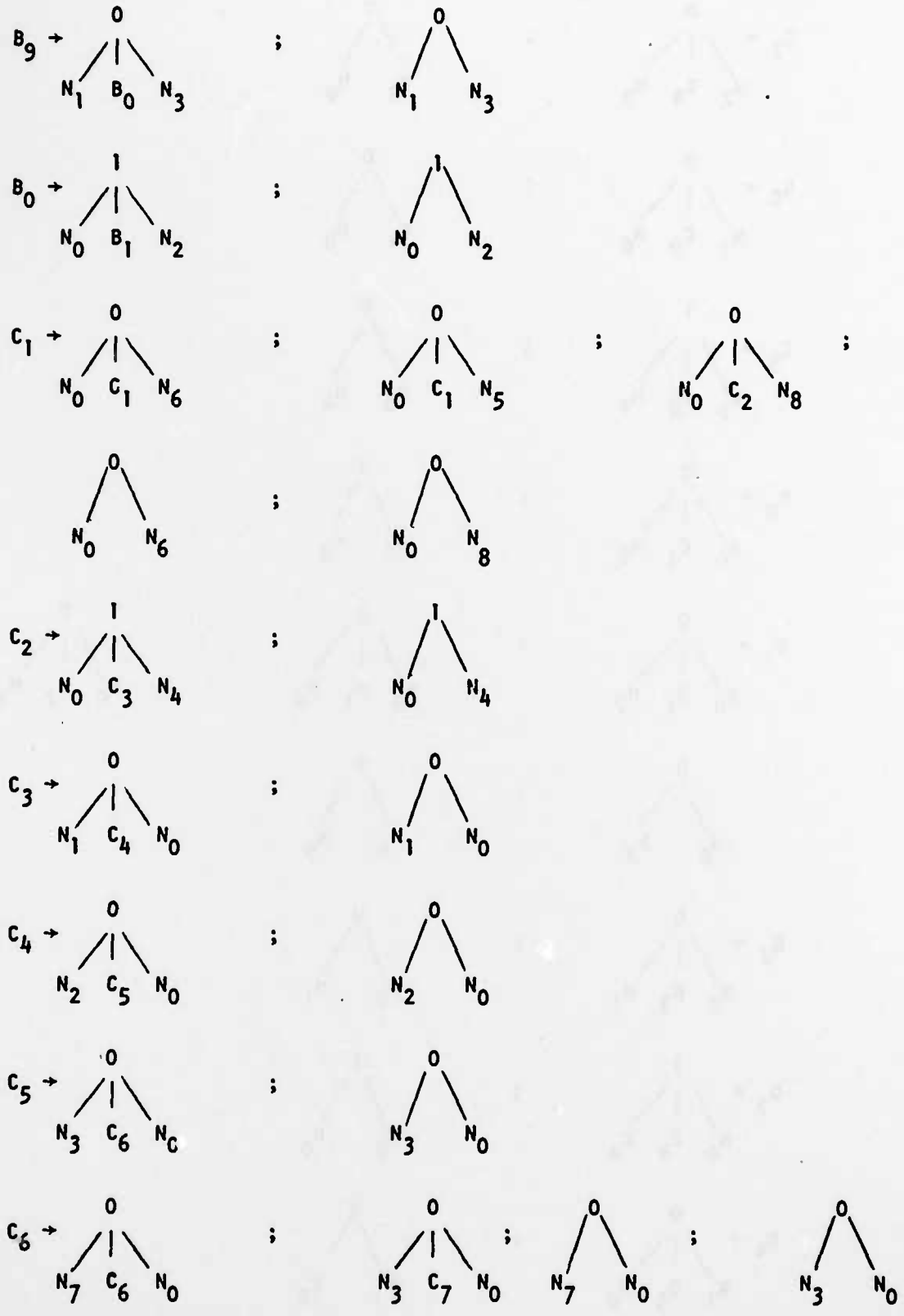
$A_6 \rightarrow$ 

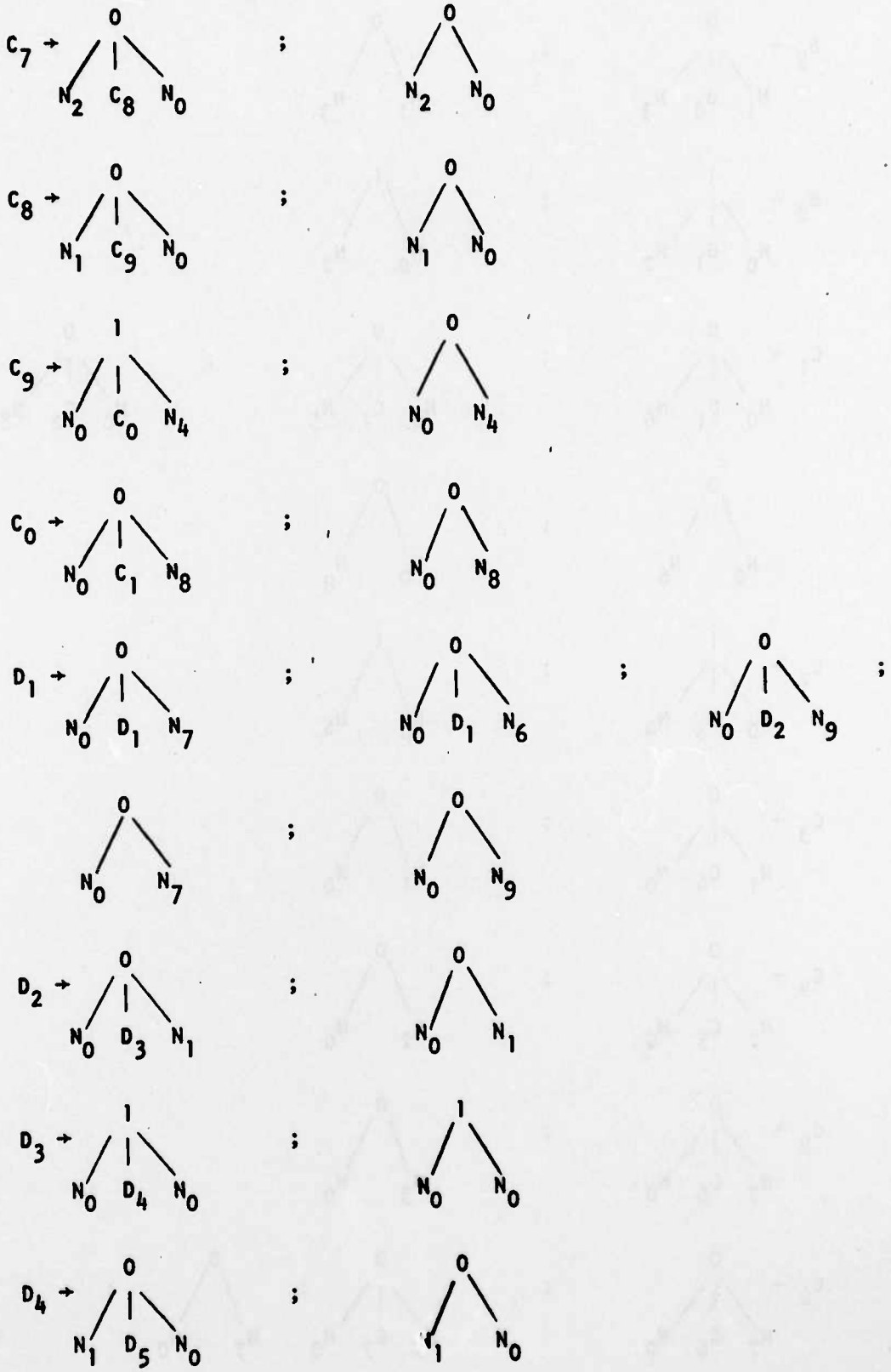
; 

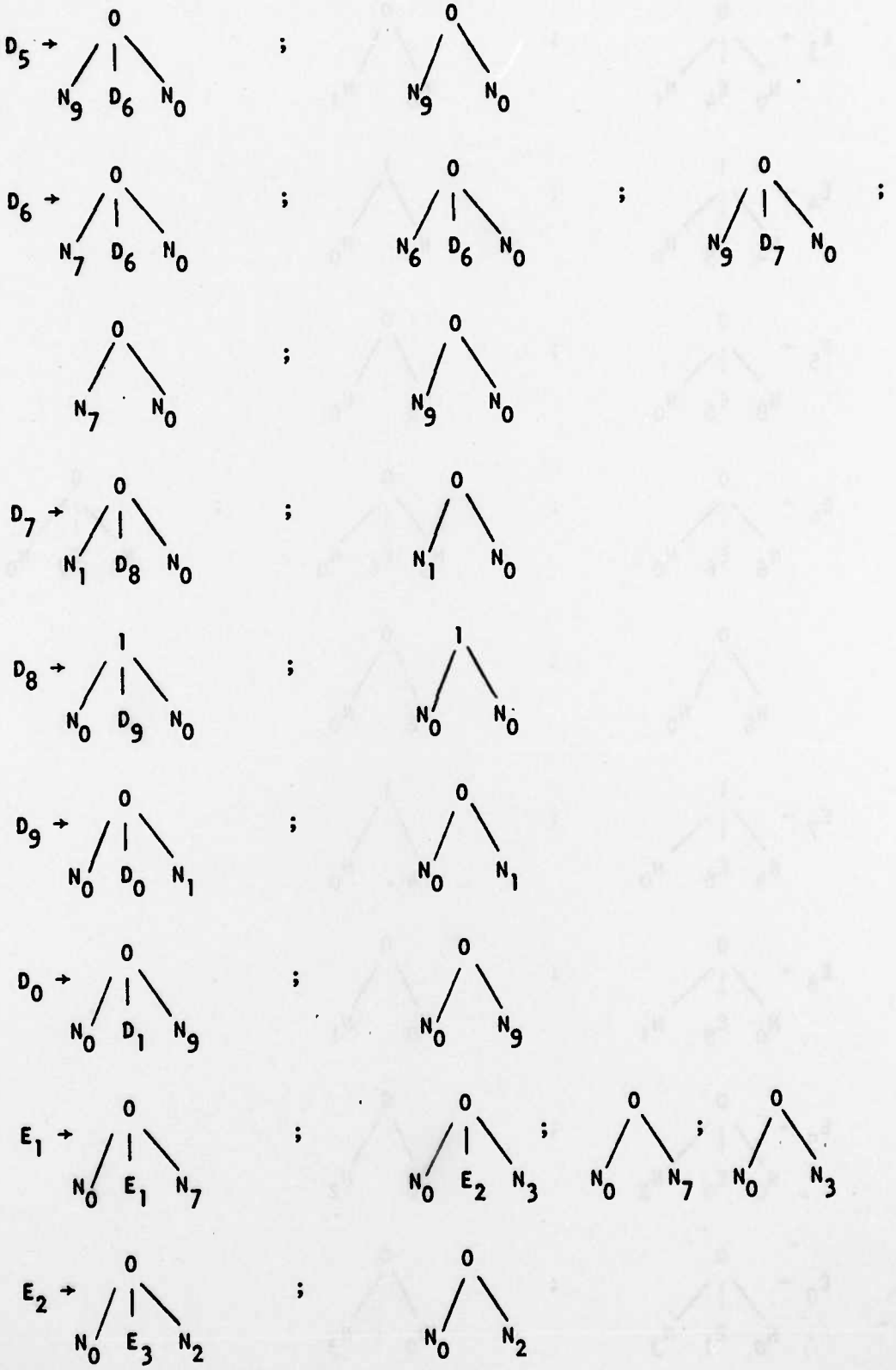
$A_7 \rightarrow$ 

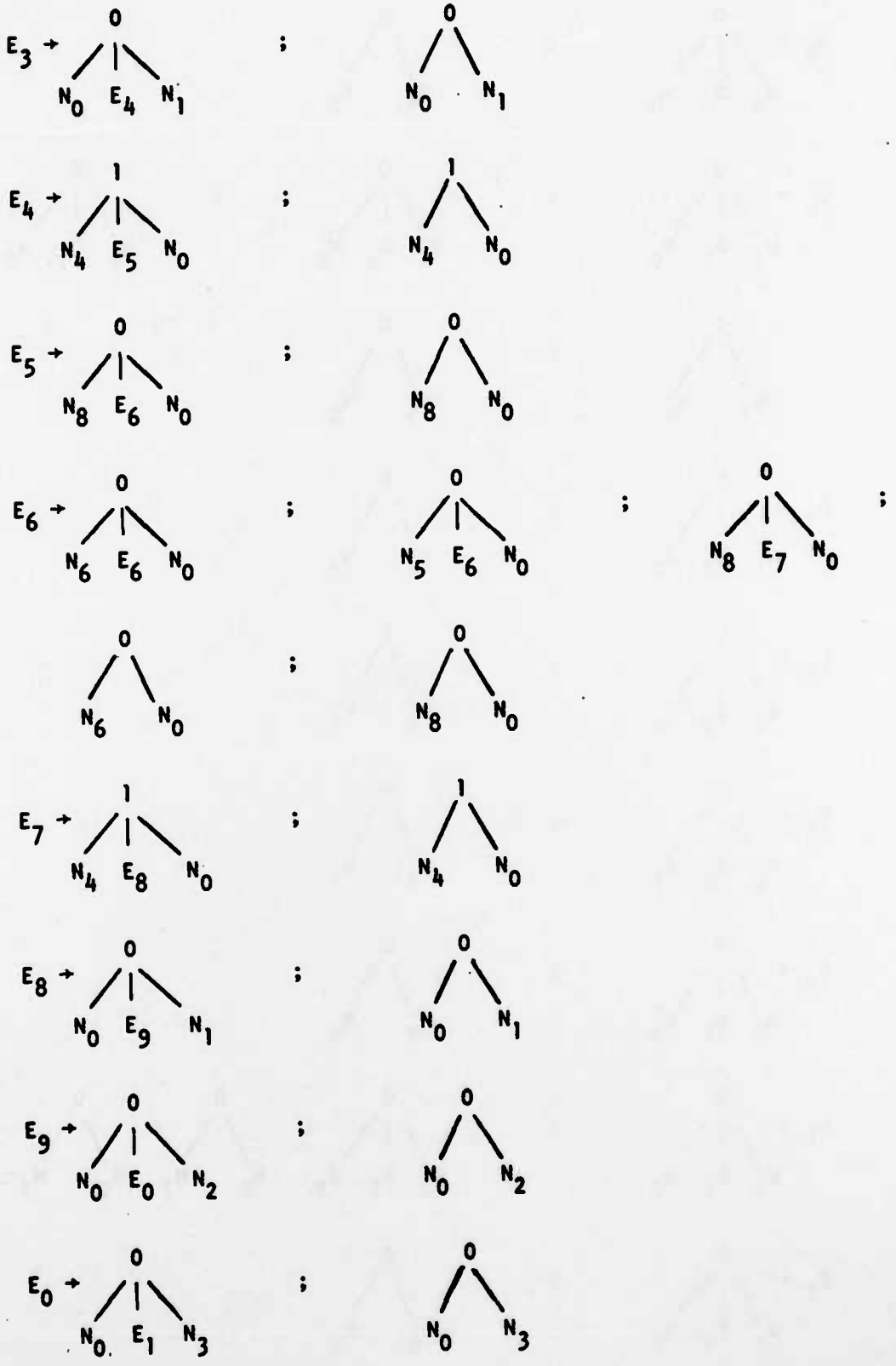
; 

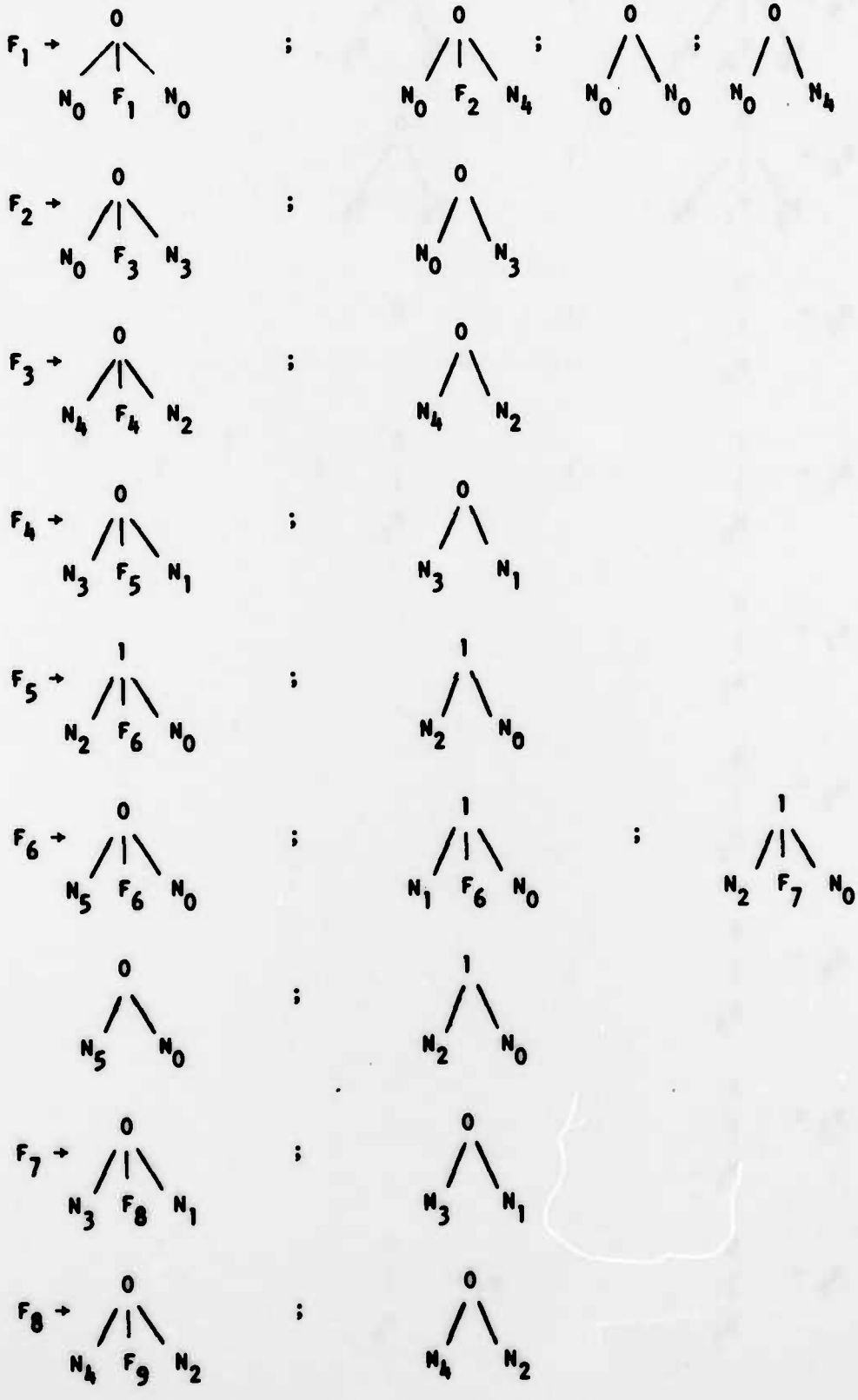

A8 → 0 N1 A1 N1 ; 0 N1 N1
B1 → 0 N0 B1 N5 ; 1 N0 B1 N1 ; 1 N0 B2 N2 ; 0 N0 N5 ; 0 N0 N2
B2 → 0 N1 B3 N3 ; 0 N1 N3
B3 → 0 N2 B4 N4 ; 0 N2 N4
B4 → 0 N3 B5 N0 ; 0 N3 N0
B5 → 0 N4 B6 N0 ; 0 N4 N0
B6 → 0 N0 B6 N0 ; 0 N4 B7 N0 ; 0 N0 N0 ; 0 N4 N0
B7 → 0 N3 B8 N0 ; 0 N3 N0
B8 → 0 N2 B9 N4 ; 0 N2 N4


B9 → 0 N1 B0 N3 ; 0 N1 N3
B0 → 1 N0 B1 N2 ; 1 N0 N2
C1 → 0 N0 C1 N6 ; 0 N0 C1 N5 ; 0 N0 C2 N8 ;
0 N0 N6 ; 0 N0 N8
C2 → 1 N0 C3 N4 ; 1 N0 N4
C3 → 0 N1 C4 N0 ; 0 N1 N0
C4 → 0 N2 C5 N0 ; 0 N2 N0
C5 → 0 N3 C6 N0 ; 0 N3 N0
C6 → 0 N7 C6 N0 ; 0 N3 C7 N0 ; 0 N7 N0 ; 0 N3 N0


C7 → 0 N2 C8 N0 ; 0 N2 N0
C8 → 0 N1 C9 N0 ; 0 N1 N0
C9 → 1 N0 C0 N4 ; 0 N0 N4
C0 → 0 N0 C1 N8 ; 0 N0 N8
D1 → 0 N0 D1 N7 ; 0 N0 D1 N6 ; 0 N0 D2 N9 ;
0 N0 N7 ; 0 N0 N9
D2 → 0 N0 D3 N1 ; 0 N0 N1
D3 → 1 N0 D4 N0 ; 1 N0 N0
D4 → 0 N1 D5 N0 ; 0 1 N0

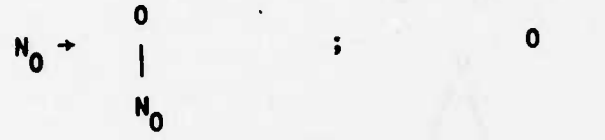

D5 → 0 N9 D6 N0 ; 0 N9 N0
D6 → 0 N7 D6 N0 ; 0 N6 D6 N0 ; 0 N9 D7 N0 ;
0 N7 N0 ; 0 N9 N0
D7 → 0 N1 D8 N0 ; 0 N1 N0
D8 → 1 N0 D9 N0 ; 1 N0 N0
D9 → 0 N0 D0 N1 ; 0 N0 N1
D0 → 0 N0 D1 N9 ; 0 N0 N9
E1 → 0 N0 E1 N7 ; 0 N0 E2 N3 ; 0 N0 N7 ; 0 N0 N3
E2 → 0 N0 E3 N2 ; 0 N0 N2

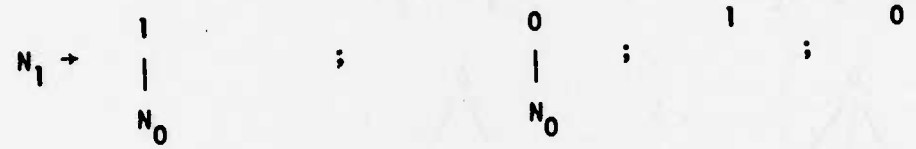

E3 → 0 N0 E4 N1 ; 0 N0 N1
E4 → 1 N4 E5 N0 ; 1 N4 N0
E5 → 0 N8 E6 N0 ; 0 N8 N0
E6 → 0 N6 E6 N0 ; 0 N5 E6 N0 ; 0 N8 E7 N0 ;
0 N6 N0 ; 0 N8 N0
E7 → 1 N4 E8 N0 ; 1 N4 N0
E8 → 0 N0 E9 N1 ; 0 N0 N1
E9 → 0 N0 E0 N2 ; 0 N0 N2
E0 → 0 N0 E1 N3 ; 0 N0 N3

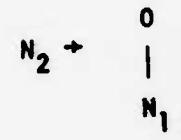

F1 → 0 N0 F1 N0 ; 0 N0 F2 N4 ; 0 N0 N0 ; 0 N0 N4
F2 → 0 N0 F3 N3 ; 0 N0 N3
F3 → 0 N4 F4 N2 ; 0 N4 N2
F4 → 0 N3 F5 N1 ; 0 N3 N1
F5 → 1 N2 F6 N0 ; 1 N2 N0
F6 → 0 N5 F6 N0 ; 1 N1 F6 N0 ; 1 N2 F7 N0 ;
0 N5 N0 ; 1 N2 N0
F7 → 0 N3 F8 N1 ; 0 N3 N1
F8 → 0 N4 F9 N2 ; 0 N4 N2

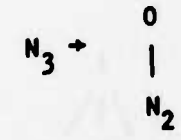

F9 →
0
N0 F0 N3
;
0
N0 N3

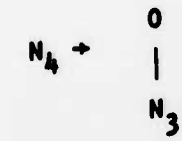

F0 →
0
N0 F1 N4
;
0
N0 N4

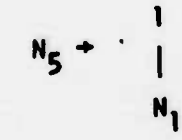

N0 →
0
N0
;
0

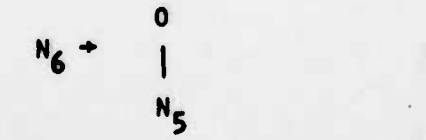

N1 →
1
N0
;
0
N0
;
1
;
0


N2 →
0
N1


N3 →
0
N2


N4 →
0
N3


N5 →
1
N1


N6 →
0
N5
N7 →
0
N6

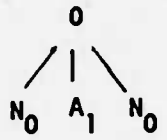
F.4. Discrimination Grammar for Pattern D68

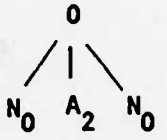
$G_{68} = (V_{68}, r, P_{68}, S_{68})$ over $\langle \Sigma, r \rangle$

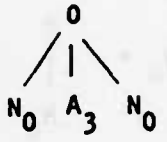
$V_{68} = \{A_1, A_2, A_3, N_0, N_1, V_0, V_1\} \cup \Sigma$

$S_{68} = \{A_1, A_2, A_3\}$

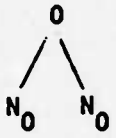
P_{68}:

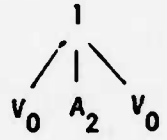
$A_1 \rightarrow$ 

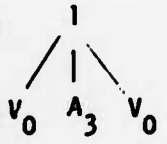
;


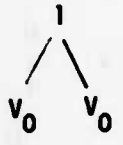
;


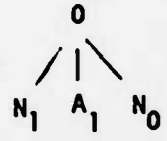
;

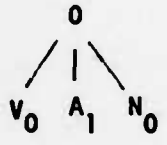


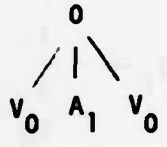
$A_2 \rightarrow$ 

;


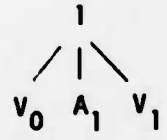
;


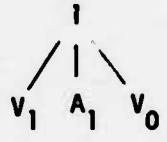
$A_3 \rightarrow$ 

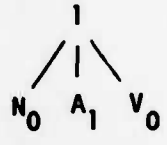
;


;


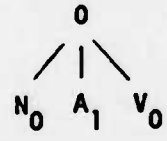
;

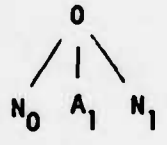


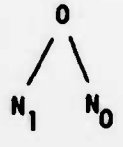
;


;


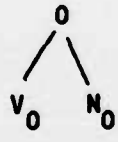
;

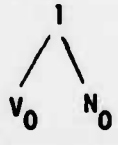


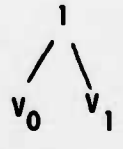
;


;


;



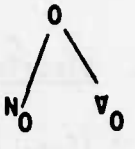
;


;


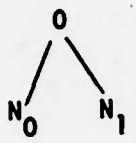
;

;

;



;

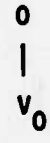


$N_0 \rightarrow$ 

; 0

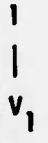
$V_0 \rightarrow$ 

; 1

$N_1 \rightarrow$ 

; 

$V_1 \rightarrow$ 

; 

APPENDIX G

ALGORITHMS FOR THE COMPUTATION OF THE DISTANCE BETWEEN TWO STRINGS, AND THE MINIMUM SPANNING TREE OF A SET OF PATTERNS

G.1. The Distance Between Two Strings

Algorithm G.1

Input: Two strings $x = a_1, a_2, \ldots, a_n$, $y = b_1, b_2, \ldots, b_m$

Output: $d(x,y)$

Method:

Step 1. $D(0,0) = 0$

Step 2.
DO i = 1, n
$D(i,0) = D(i-1,0) + 1$
DO j = 1, m
$D(0,j) = D(0,j) + 1$

Step 3.
DO i = 1, n
DO j = 1, m
$e_1 = D(i-1,j-1) + 1$ if $a_i = b_j$, or $e_1 = D(i-1,j-1)$ if $a_i \neq b_j$
$e_2 = D(i-1,j) + 1$
$e_3 = D(i,j-1) + 1$
$D(i,j) = \min(e_1, e_2, e_3)$

Step 4. $d(x,y) = D(n,m)$, exit

G.2. The Minimum Spanning Tree

Algorithm G.2

Input: $X = \{x_1, x_2, \ldots, x_n\}$, a set of sentences

Output: The minimum spanning tree of X

Method:

Step 1. Assume that there are n nodes

Step 2. Compute $d(x_i,x_j)$ for all i, j. Let $d(x_i,x_j)$ be the length of the arc connected node i and j, and denote as d(i,j).

Step 3. List all arc (i,j) in the order of increasing d(i,j)

Step 4. Put the first arc (p,q) on the list into list A

Step 5. Put the next arc on the list into A, except if a circuit can be found with the arcs already in A.

Step 6. If all nodes are connected, stop, otherwise go to Step 5.

BIBLIOGRAPHIC DATA SHEET	1. Report No. AFOSR-TR- 77-1258	2.	3. Recipient's Accession No.
4. Title and Subtitle: ERROR-CORRECTING PARSING FOR SYNTACTIC PATTERN RECOGNITION			5. Report Date: August, 1977 6.
7. Author(s): S. Y. Lu and K. S. Fu			8. Performing Organization Rept. No. TR-EE 77-31
9. Performing Organization Name and Address: School of Electrical Engineering Purdue University W. Lafayette, Indiana 47907			10. Project/Task/Work Unit No. 11. Contract/Grant No. AFOSR Grant 74-2661
12. Sponsoring Organization Name and Address: AFOSR/NM Bldg. 410 Bolling AFB Washington, D.C. 20332			13. Type of Report & Period Covered: Technical Report 14.

15. Supplementary Notes

16. Abstracts

The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patters is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers (Cont.)

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers Open Ended Terms

17c. COSATI Field/Group

18. Availability Statement: Release unlimited.	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 322
	20. Security Class (This Page) UNCLASSIFIED	22. Price

are employed to determine the distance between an input syntactic pattern and a formed cluster, or a language. Finally, using the error-correcting parsing techniques, real data examples on texture modeling and discrimination are presented.