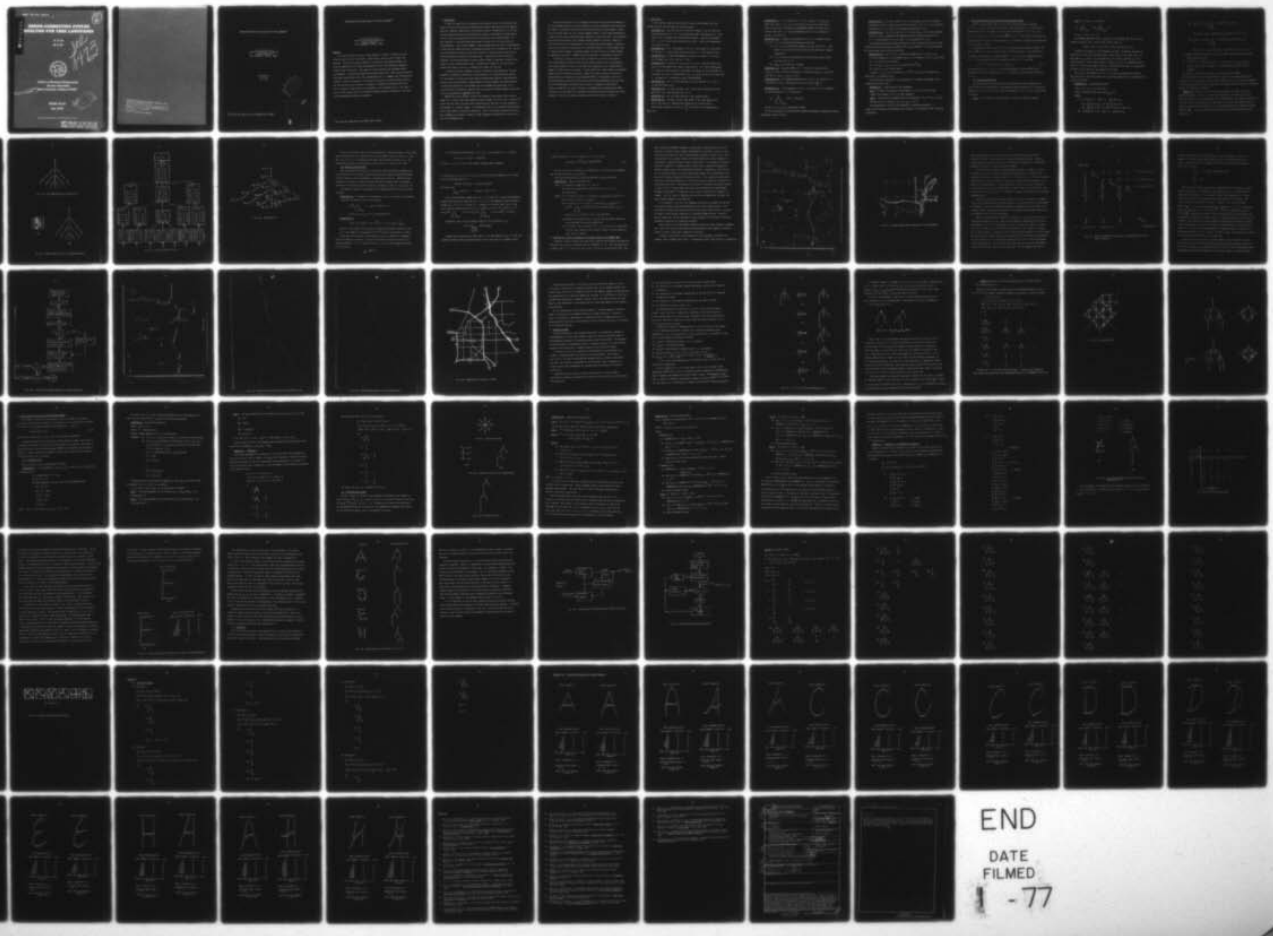
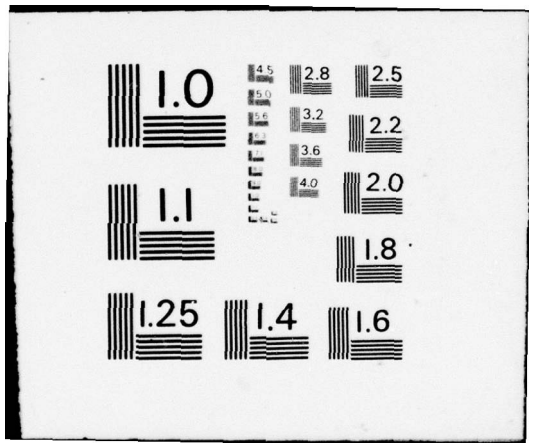
AD-A032 744 PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGI--ETC F/G 12/1
ERROR-CORRECTING SYNTAX ANALYSIS FOR TREE LANGUAGES.(U)
JUL 76 S Y LU, K S FU AF-AFOSR-2661-74

UNCLASSIFIED TR-EE76-24 AFOSR-TR-76-1207 NL

1 OF 1
ADA032744



END
DATE FILMED
1 -77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

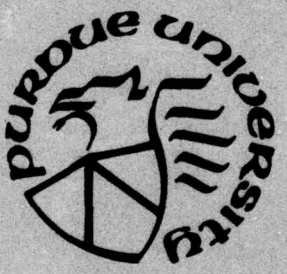
AFOSR - TR - 76 - 1207

AD A032744

# ERROR-CORRECTING SYNTAX ANALYSIS FOR TREE LANGUAGES

S. Y. Lu

K. S. Fu




School of Electrical Engineering

Purdue University

West Lafayette, Indiana 47907

Approved for public release;
Distribution unlimited.

DDC
DEC 3 1976
DISTRIBUTED

TR-EE 76-24

July 1976

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

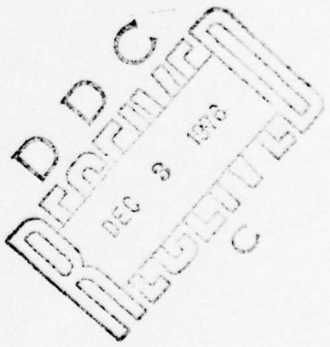
COPY AVAILABLE TO DDC DOES NOT
PERMIT FULLY LEGIBLE PRODUCTION

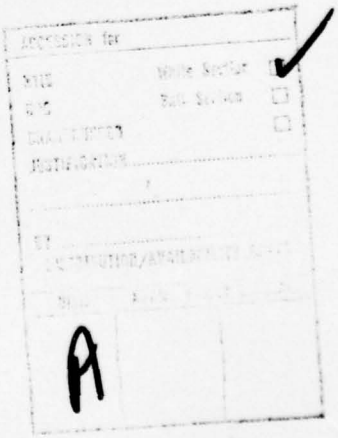
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

# ERROR-CORRECTING SYNTAX ANALYSIS FOR TREE LANGUAGES[†]

S. Y. Lu and K. S. Fu
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE-76-24
July 1976



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

# ERROR-CORRECTING SYNTAX ANALYSIS FOR TREE LANGUAGES[†]

S. Y. Lu and K. S. Fu
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

## Abstract

An error-correcting version of tree automaton is studied. Assume the syntax errors on trees to be of five types: substitution, stretch, split, branch, and deletion. The error-correction problem is to determine the distance between two trees as measured by the minimum cost sequence of error transformations needed to transform one to the other. Two types of error-correcting tree automaton (ECTA) are proposed: structure-preserved ECTA and generalized ECTA. The structure-preserved ECTA takes substitution errors into consideration only. A LANDSAT data interpretation problem is used as an example to illustrate the process of structure-preserved ECTA. The restriction on tree structure is removed in formulating the generalized ECTA. All five types of syntax errors are considered. Any erroneous tree, in this case, is able to find its most similar (in terms of minimum distance) correction in the sense of both node-labeling and tree structure.

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

## 1. Introduction

In order to tackle with the uncertainty that usually exists in the process under study, error-correcting parsing techniques have recently been applied to the areas of compiling, computer communication, and syntactic pattern recognition [3, 15, 23, 24]. The most widely used error-correcting parsing scheme is formulated to include substitution, insertion, and deletion errors for string-to-string correction [28]. The basic approach is to define these three types of syntax errors in terms of error transformations. The original grammar is then expanded such that error transformations on each terminal symbol has their corresponding terminal error productions [1]. During the error-correcting process, a decision criterion is required when the parsing procedure faces multiple choices of the next move. Two decision criteria have been proposed: the minimum-distance criterion, where distance is measured in terms of the number of error transformations used in a deterministic model [1, 21, 23], and the maximum-likelihood criterion in a probabilistic model [15, 22, 24].

In applying syntactic methods to pattern recognition, one-dimensional (string) grammars are sometimes inefficient in describing two- or three-dimensional objects. For the purpose of effectively describing high-dimensional patterns, high-dimensional grammars such as web grammars, graph grammars, and tree grammars have been proposed. In practical application, tree grammars and tree automata have been used in the classification of fingerprint patterns [19], the analysis of bubble chamber events [4], and the interpretation of LANDSAT data [17].

Generalized finite automata, called tree automata, which accept finite trees of symbols as its input, have been studied by several authors [6, 8, 20, 26]. Brainerd [6] proves that the class of systems which generate exactly the sets of trees accepted by the automata is a regular system. Fu and Bhargava [14] first introduced the application of the tree systems into pattern recognition. Later, as studied by Brayer and Fu [7], tree languages are actually a subset of graph languages corresponding to the type 3 in the string languages case.

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 paper 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 (ECTA) proposed in Section 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 ECTA in LANDSAT data interpretation is presented in Section 4.

In Section 5, syntax errors on trees are defined in terms of five types of 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. By this definition, tree distance is measurable between trees of different structures. Based on the metric defined in Section 5, a generalized error-correcting tree automaton (GECTA) is formulated in Section 6. The GECTA accepts structurally distorted as well as node-mislabeled trees such that their minimum-distance corrections can be found. An example of hand-printed character recognition is given in Section 7 to demonstrate the operation of GECTA. A discussion on the capability of general error-correcting parsing schemes and their prospects in the area of pattern recognition can also be found in Section 7.

## 2. Definitions

In this section, some basic definitions on trees, tree grammars, and tree automata given by Brainerd [6] are briefly reviewed.

DEFINITION 2.1. 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: $d(0)=0$, $d(a \cdot i)=d(a)+1$, $i \in N^+$. $a \leq b$ 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 2.2. 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 2.3. A stratified alphabet is a pair $\langle \Sigma, r \rangle$ where $\Sigma$ is a finite set of symbols and $r: \Sigma \to N$. Let $\Sigma_n = r^{-1}(n)$.

DEFINITION 2.4. A tree over $\Sigma$ (i.e., over $\langle \Sigma, r \rangle$) 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 stratification 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 $D_\alpha$. Let $T_\Sigma$ be the set of all trees over $\Sigma$. The depth of $\alpha$ is defined as, $d(\alpha) = \max\{d(a) \mid a \in D_\alpha\}$.

DEFINITION 2.5. 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 2.6. Let $\alpha \in T_\Sigma$, and $a \in D_\alpha$, $\alpha/a = \{(b, x) \mid (a \cdot b, x) \in \alpha\}$, $\alpha/a$ is the subtree of $\alpha$ at a and $\alpha/a$ occurs at a in $\alpha$.

DEFINITION 2.7. Let $\alpha \in T_\Sigma$, $a \in U$, then $a \cdot \alpha = \{(b, x) \mid (b/a, x) \in \alpha\}$.

DEFINITION 2.8. 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 $\alpha/a$ at a by the tree $\beta$.

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

DEFINITION 2.9. t is a term over $<\Sigma, r>$ iff $t = x\varepsilon\Sigma_0$ or $t = t_1t_2\ldots t_nx$ where $x\varepsilon\Sigma_n$, and $t_i$, $1 \leq i \leq n$, is a term, $\bar{T}_\Sigma$ is the set of terms over $\Sigma$. There is obviously a one-to-one correspondence between the terms over $\Sigma$ and the trees over $\Sigma$. If the corresponding tree of t is a subtree of $\alpha$, we say t is a term in $\alpha$.

DEFINITION 2.10. 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 element of V and $V-\Sigma$ are called terminal and nonterminal symbols, respectively.

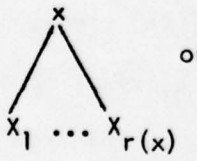
(2) P is a finite set of productions of the form $\phi \mapsto \psi$, where $\phi$ and $\psi$ are trees over $<V, r'>$.

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

DEFINITION 2.11. Derivation $\alpha \overset{a}{\mapsto} \beta$ is in $G_t$ iff $\phi \mapsto \psi$ in P such that $\alpha/a = \phi$ and $\beta = \alpha(a \leftarrow \psi)$. $\alpha \overset{*}{\Longrightarrow} \beta$ 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 2.12. The language generated by $G_t = (V, r', P, S)$ over $<\Sigma, r>$ is defined as, $L(G_t) = \{\alpha\varepsilon T_\Sigma | \text{there exist } x\varepsilon S \text{ such that } x \overset{*}{\underset{G_t}{\Longrightarrow}} \alpha\}$.

DEFINITION 2.13. A tree grammar $G_t = (V, r', P, S)$ over $<\Sigma, r>$ is expansive iff each production in P is of the form

$X_0 \rightarrow$  or $X_0 \rightarrow x$ where $x\varepsilon\Sigma$.



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

For our convenience, we further define immediate predecessor, immediate successor, and adjacent node as follows.

DEFINITION 2.14. Let $\alpha$ be a tree over $<\Sigma, r>$, and $a, b \in D_\alpha$, $\alpha(a)$ is the immediate predecessor of $\alpha(b)$ iff there exists $i \in N^+$ such that $a \cdot i = b$. If x is the immediate predecessor of y in $\alpha$, then y is an immediate successor of x in $\alpha$.

DEFINITION 2.15. Let $\alpha$ be a tree over $<\Sigma, r>$ and $a, b \in D_\alpha$. $\alpha(a)$ is right adjacent to $\alpha(b)$ if $\alpha(a)$ is immediate right adjacent to $\alpha(b)$ in postfix notation of $\alpha$. If x is right adjacent to y in $\alpha$, then y is left adjacent to x in $\alpha$.

Tree automata have been studied and defined by several authors [6, 8, 20, 25, 26]. Following the definition given by Brainers [6], a tree automaton is a finite automaton with many-to-one state transition functions.

DEFINITION 2.16. Let $<\Sigma, r>$ be a stratified alphabet and $\Sigma = \{\sigma_1, \sigma_2 \ldots, \sigma_k\}$. A finite $\Sigma$-automaton or tree automaton over $\Sigma$ 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 [14].

ALGORITHM 2.1. Construction of tree automaton.

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

Step 2. The equivalent nondeterministic tree automaton is $M_t = (V'-\Sigma, f_1, \ldots, f_k, \{S\})$ where $f_i(X_1, \ldots, X_n) \sim X_0$ 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 parallel branches simultaneously then transfers to the states of their immediate predecessors.

## 3. Structure-Preserved Error-Correcting Tree Automaton (ECTA)

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

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

$$\mu(\alpha, \alpha') = \min_{\beta} \{\mu(\beta, \alpha') | \beta \in L, D_\beta = D_{\alpha'}\} \tag{3.1}$$

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

### 3.1 Minimum-distance ECTA

By using the idea of adding terminal error production rules corresponding to substitution error transformations, as proposed by Aho and Peterson [1], 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 $P'$

$$X_0 \rightarrow \overset{y}{\underset{X_1 \cdots X_{r(x)}}{\diagup\diagdown}} \quad , \text{ if } X_0 \rightarrow \overset{x}{\underset{X_1 \cdots X_{r(x)}}{\diagup\diagdown}} \in P$$

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

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

Following the concept of tree automaton, the ECTA is a backward procedure of constructing a tree-like transition table. Assume that $\alpha$ is the input tree. For each node $a \in D_\alpha$, there is a corresponding set of triplets, denoting $t_a$ in the transition table. Each triplet $(X,\ell,k)$ is added to $t_a$ if X is a candidate state of node a, $\ell$ is the minimum number of errors in subtree $\alpha/a$ when node a is represented by state X, and k *specifies the production rule used.*

For a given tree grammar $G_t = (V, r, P, S)$ the tree automaton that accepts $L(G_t')$ and emits a parse that consists of the minimum number of error production rules is given as follows.

ALGORITHM 3.1. Minimum-distance ECTA.

Input: $G_t = (V, r, P, S)$ and tree $\alpha$.

Output: Transition table of $\alpha$ and $\mu_{L(G_t)}(\alpha)$.

Method:

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

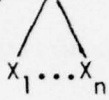
(a) $(X_0,0,k)$ if $X_0 \rightarrow x$ is the $k^{th}$ rule in P.

(b) $(X_0,1,k)$ if $X_0 \rightarrow y$ is the $k^{th}$ rule in P and $y \neq x$.

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

(a) $(X_0,\ell,k)$, if $X_0 \rightarrow$  is the $k^{th}$ 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) $(X_0,\ell,k)$ , if $X_0 \rightarrow$  is the $k^{th}$ 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 $\ell = \ell_1 +\ldots+ \ell_n + 1$.

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

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

The parse of $\alpha$ can easily be traced out from the transition table.

The tree grammar in the following example is part of the highway grammar used in Section 4. In the meantime, we use it as an example here to illustrate the operation of structure-preserved ECTA and to demonstrate the highway patterns recognition procedure that will be discussed in Section 4.

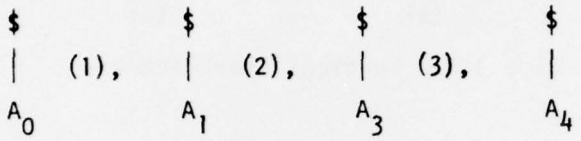
EXAMPLE 3.1. Consider a set of vertical line patterns as given in Fig. 3.1. Assume that elements in the 4 x 4 array are connected as a tree shown in Fig. 3.2. Thus, each pattern has its corresponding tree representations. For example, pattern (b) in Fig. 3.1 can be represented by the tree shown in Fig. 3.3, where nodes labeled by symbol "b" represent blank 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 $\langle \Sigma, r \rangle$ where

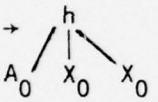
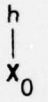
$V = \{S, A_0, A_1, A_2, A_3, X_0, I_1, I_2, I_3, \$, b, h\}$

$\Sigma = \{ \bullet \quad \square \quad \blacksquare \}$
$\$ \quad , \quad b \quad , \quad h$

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

P: $S \rightarrow$  (1), (2), (3), (4)

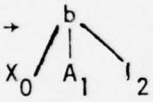


$A_0 \rightarrow$  (5),  (6), h (7)



$A_1 \rightarrow$  (8)

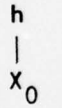


$A_2 \rightarrow$  (9)

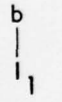


$A_3 \rightarrow$  (10)

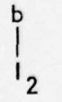


$I_1 \rightarrow$  (11) $I_1 \rightarrow h$ (17)

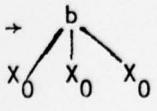
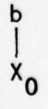


$I_2 \rightarrow$  (12)



$I_3 \rightarrow$  (13)



$X_0 \rightarrow$  (14),  (15), b (16)

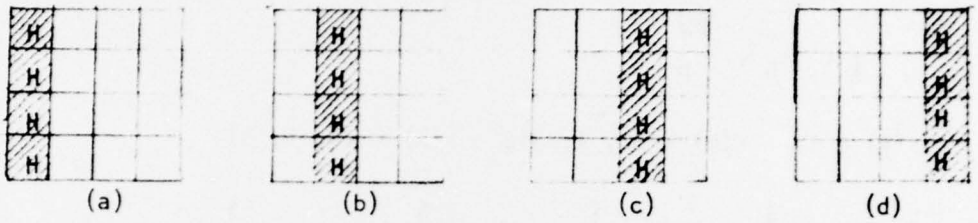



Fig. 3.1. Vertical Line Patterns

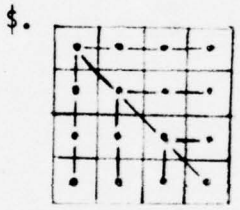



Fig. 3.2. A Tree Structure

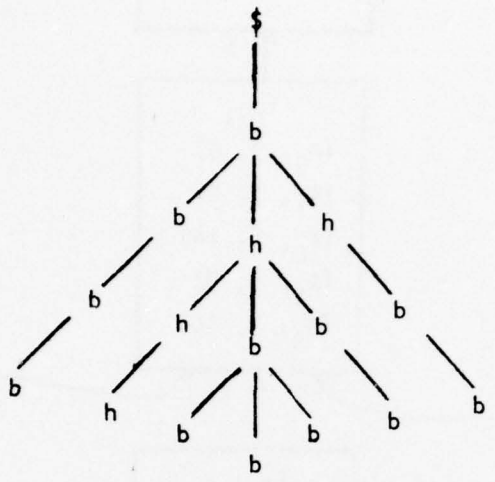
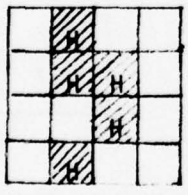
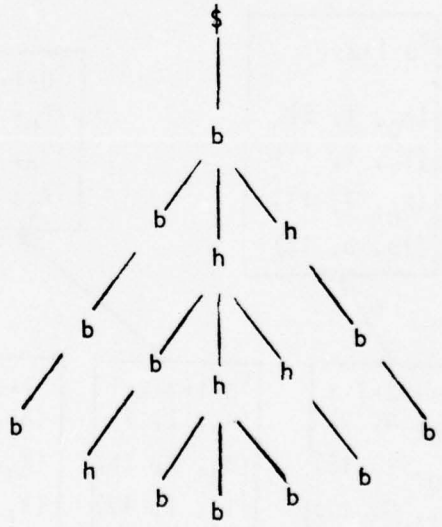



Fig. 3.3. Tree Representation of pattern (b)



(a)




(b)

Fig. 3.4. A Noisy Pattern and its Tree Representation $\alpha$

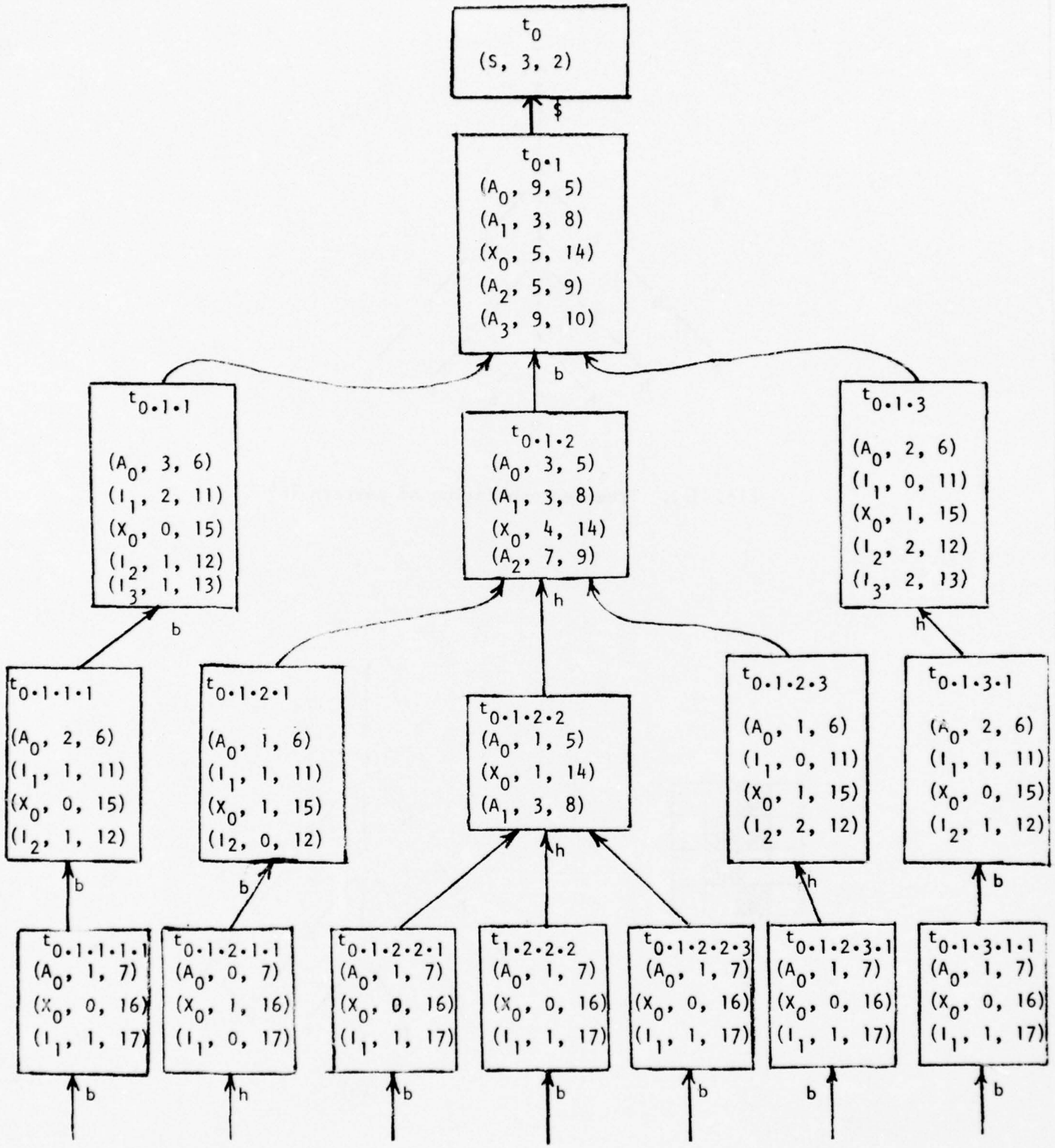



Fig. 3.5. Transition Table of Tree $\alpha$

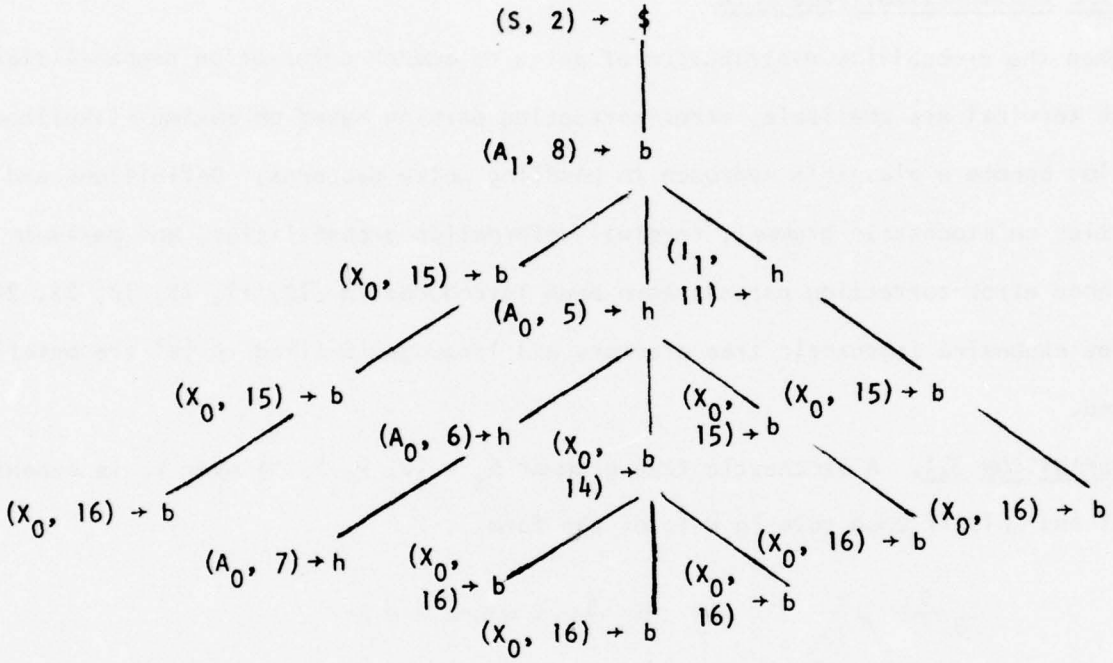



Fig. 3.6. The Parse of $\alpha$

Given a noisy pattern and its tree representation $\alpha$, which are shown in Fig. 3.4(a) and (b) respectively, the recognition of $\alpha$ by using ECTA is shown in Fig. 3.5. Since (S, 3, 2) is in $t_0$, $\alpha$ is accepted by the ECTA, and the number of errors is 3. The parse that generates the corresponding correct tree of $\alpha$ is shown in Fig. 3.6.

## 3.2 Maximum-likelihood ECTA

When the probability distribution of patterns and/or deformation probabilities of each terminal are available, error-correcting parsing based on maximum-likelihood criterion become a plausible approach in handling noisy patterns. Definitions and properties on stochastic grammar, terminal deformation probabilities, and maximum-likelihood error-correcting parsers have been introduced in [10, 11, 15, 18, 23, 24].

The expansive stochastic tree grammars and languages defined in [4] are briefly reviewed.

**DEFINITION 3.1.** A stochastic tree grammar $G_S = (V, r, P, S)$ over $V_T$ is expansive if and only if each rule in P is of the form

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

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

**DEFINITION 3.2.**

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

where k is the number of all distinctly different derivation of $\alpha$ from S, and $p_i$ is the probability associated with the $i^{th}$ distinct derivation of $\alpha$ from S.

Assume that the occurrence of substitution error on a terminal is independent from its neighboring terminals. Fung and Fu [15] define substitution error as a stochastic mapping: $\delta: \Sigma \to \Sigma$ such that $\delta(a) = b$, $a, b \in \Sigma$, with probability $q(b|a)$, and furthermore,

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

In our application, assume that $t = t_1 \cdots t_n\, x$ is a term over $\langle \Sigma, r \rangle$. We have

$$\delta(t_1 \cdots t_n\, x) = \delta(t_1) \cdots \delta(t_n)\delta(x)$$

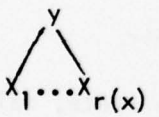
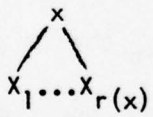
A tree $t = t_1 \cdots t_n\, x$ is said to be locally corrupted under $\delta$ mapping.

If two trees $\alpha = t_1 \cdots t_n\, x$, $\alpha' = t_1' \cdots t_n'\, x'$ are in $T_\Sigma^D$, the probability of $\alpha'$ being the noisy deformed tree of $\alpha$ is

$$q(\alpha'|\alpha) = q(t_1'|t_1) \cdots q(t_n'|t_n)\, q(x'|x)$$

We further have

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

For a given stochastic grammar $G_S = (V, r, P, S)$ over $\langle \Sigma, r \rangle$, when the deformation probabilities $q(y|x)$ are known for all $x \in \Sigma$, $y \in \Sigma'$, the stochastic covering grammar becomes $G_S' = (V', r', P', S)$ over $\langle \Sigma', r' \rangle$ where $V' = (V-\Sigma) \cup \Sigma'$ 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 $p' = p\, q(y|x)$.

Apparently, $L(G_S')$ can be written as $L(G_S') = \{(\alpha', p'(\alpha')) | \alpha' \in T_{\Sigma'}$,

$$p'(\alpha') = \sum_{\substack{\alpha \in L(G_S) \\ D_{\alpha'} = D_\alpha}} q(\alpha'|\alpha)\, p(\alpha)\}$$

Suppose that the given noisy input tree $\alpha'$ is in tree domain D, i.e., $\alpha' \in T_\Sigma^D$, the maximum-likelihood error-correcting decision rule in this case is to choose a tree $\alpha$

in $L(G_S)$ of domain D, i.e., $\alpha \in L(G_S)$ and $\alpha \in T_\Sigma^D$, such that

$$q(\alpha'|\alpha)p(\alpha) = \max_{\beta \in L(G_S) \cap T_\Sigma^D} q(\alpha'|\beta)p(\beta) \tag{3.2}$$

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

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

ALGORITHM 3.2. Maximum-likelihood ECTA

Input: (1) Stochastic grammar $G_S = (V, r, P, S)$.

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

(3) Input tree $\alpha$.

Method: (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 $k^{th}$ rule in P and $p' = p\, q(y|x)$.

(2) If $r[\alpha(a)] = n$, $n > 0$, $\alpha(a) = y$ then add to $t_a$ $(X_0,p',k)$, if $X_0 \xrightarrow{p} \overset{x}{\diagup\ \diagdown}\ X_1 \ldots X_n$ is the $k^{th}$ 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 $p' = p_1'*\ldots*p_n'*p*q(y|x)$.

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

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

## 4. Application of ECTA to Recognition of Highway Patterns from LANDSAT Data

Recently, syntactic methods have been used to analyze and interpret data obtained from the earth resource technology satellite (LANDSAT) [7, 17]. The input data used by Brayer and Fu or Li and Fu are the results of pointwise classification [27]. Each

pixel collected by LANDSAT represents a ground area of approximately 60 x 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 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 patterns is given in Fig. 4.1 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. Fig. 4.2, which is obtained from the official highway map, indicates the major divided highways of the same area.

As illustrated in Fig. 4.1, 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. Therefore, the need of using syntactic methods as a refinement becomes evident. Syntactic pattern recognition has the advantage of using contextual and structural information contained in patterns for recognition purposes.

Brayer and Fu [7] use web grammars modeling classes of clouds, downtown, highways, etc. Due to the lack of an efficient parsing procedure for web grammar, a matching process is used in the actual recognition of patterns.

In [17], Li and Fu use a tree grammar and a tree automaton to analyze line patterns such as highways and rivers. It demonstrates fairly good results in recognizing

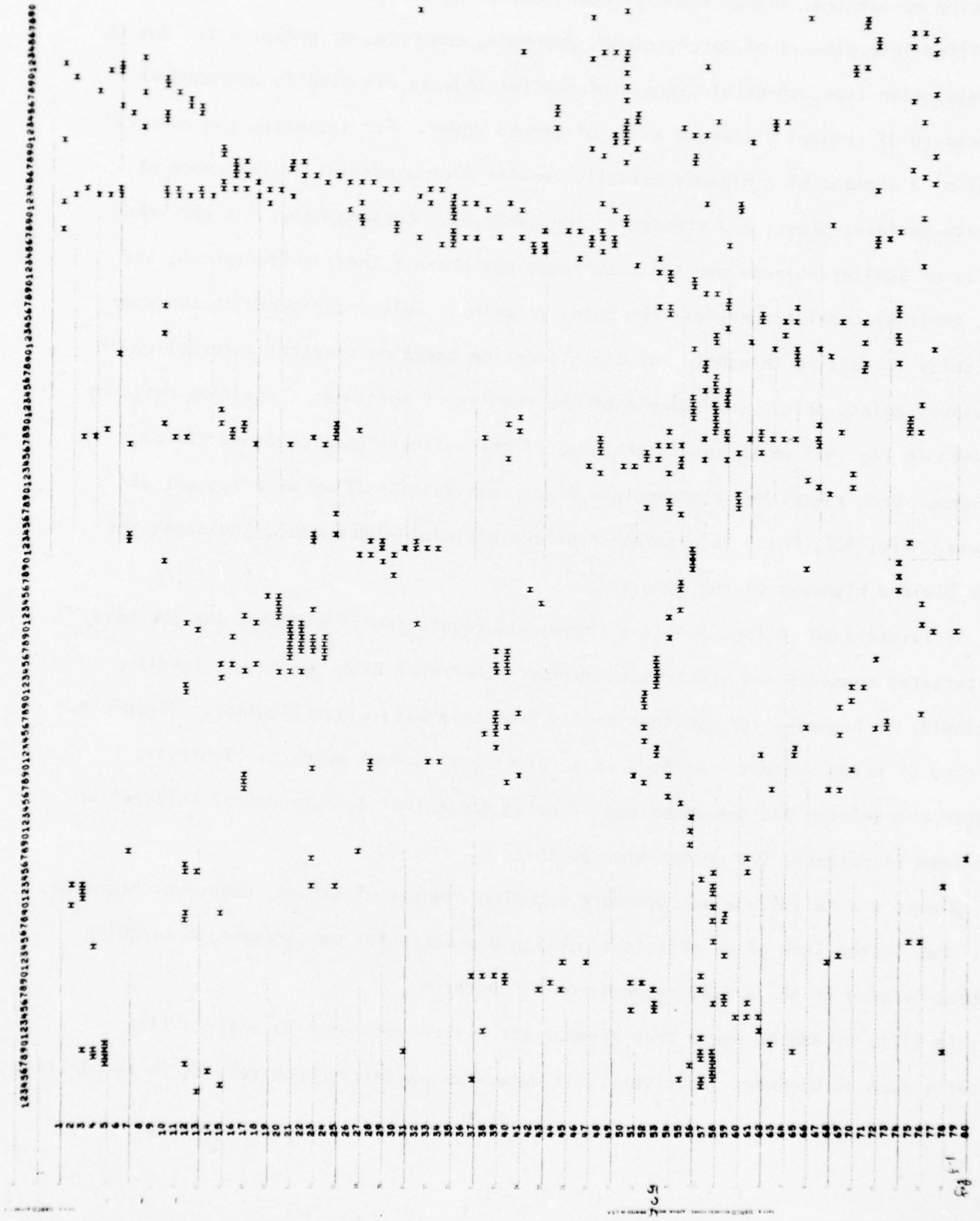


Fig. 4.1. Pointwise Classified Highway Data Obtained from Grand Rapids, Michigan

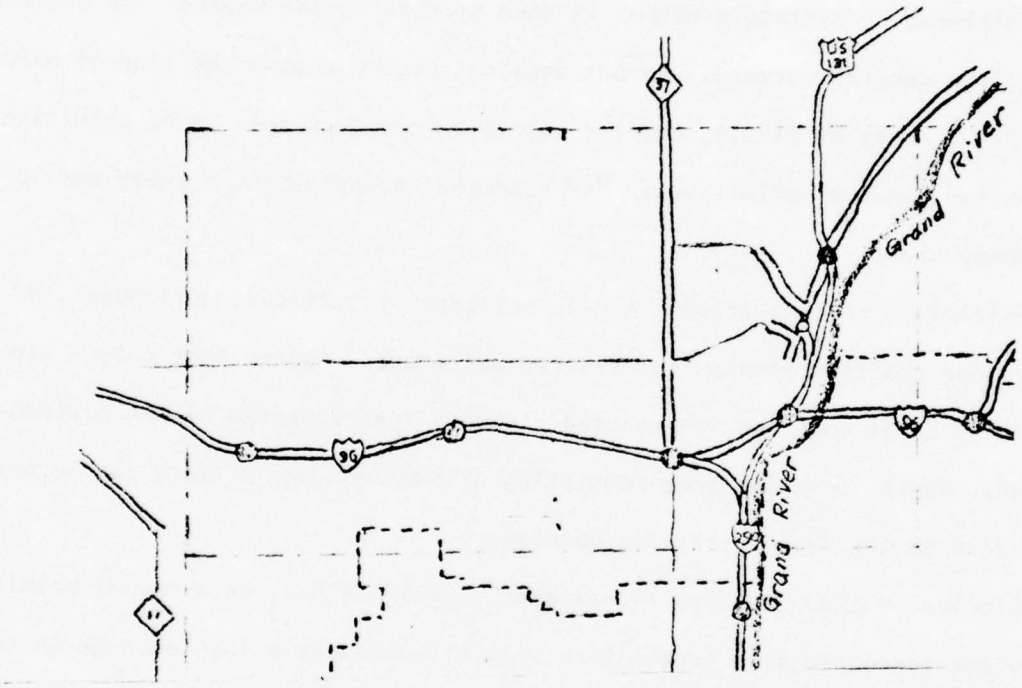



Fig. 4.2. Divided Highway Map of Northern Part of Grand Rapids

rivers among ponds and some modern buildings with glass walls having similar reflecting surfaces as water, but poor in analyzing highway patterns. Evidently, highway patterns are usually too noisy to be effectively analyzed by a conventional grammatical inference procedure and parsing methods.

In [7] and [17], pictures are scanned window by window. A window is an array of pixels. Each pixel is either represented by symbol "H" or is a blank. Some subpatterns consisting of several pixels (2 x 2 in [17] and 5 x 5 in [7]) are selected as primitives. A syntactic method is then used to decide whether the pattern in a window is a desired pattern. In our application, we choose the size of window to be an 8 x 8 array of pixels, and the labels on single pixels to be primitives. Thus, we have two kinds of primitives: "h" represent a segment of highway and "b" represent nonhighway area.

Initially, we take primary sample patterns of vertical, horizontal, or diagonal line 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 combined primary sample patterns are also added to the set of positive samples.

Similar to the procedure illustrated in Example 4.1, an array of primitives in a window are connected 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." Hence, there are $2^{64}$ tree representations in $T_\Sigma^{D}$, where $\Sigma = \{b,h,\$\}$, \$ is a start terminal. Apparently, the set of all possible patterns in an 8 x 8 window and the set of all labeled trees in the tree domain $D_H$ are one-to-one correspondence. Fig. 4.3 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 tree grammar, $G_H$, that is

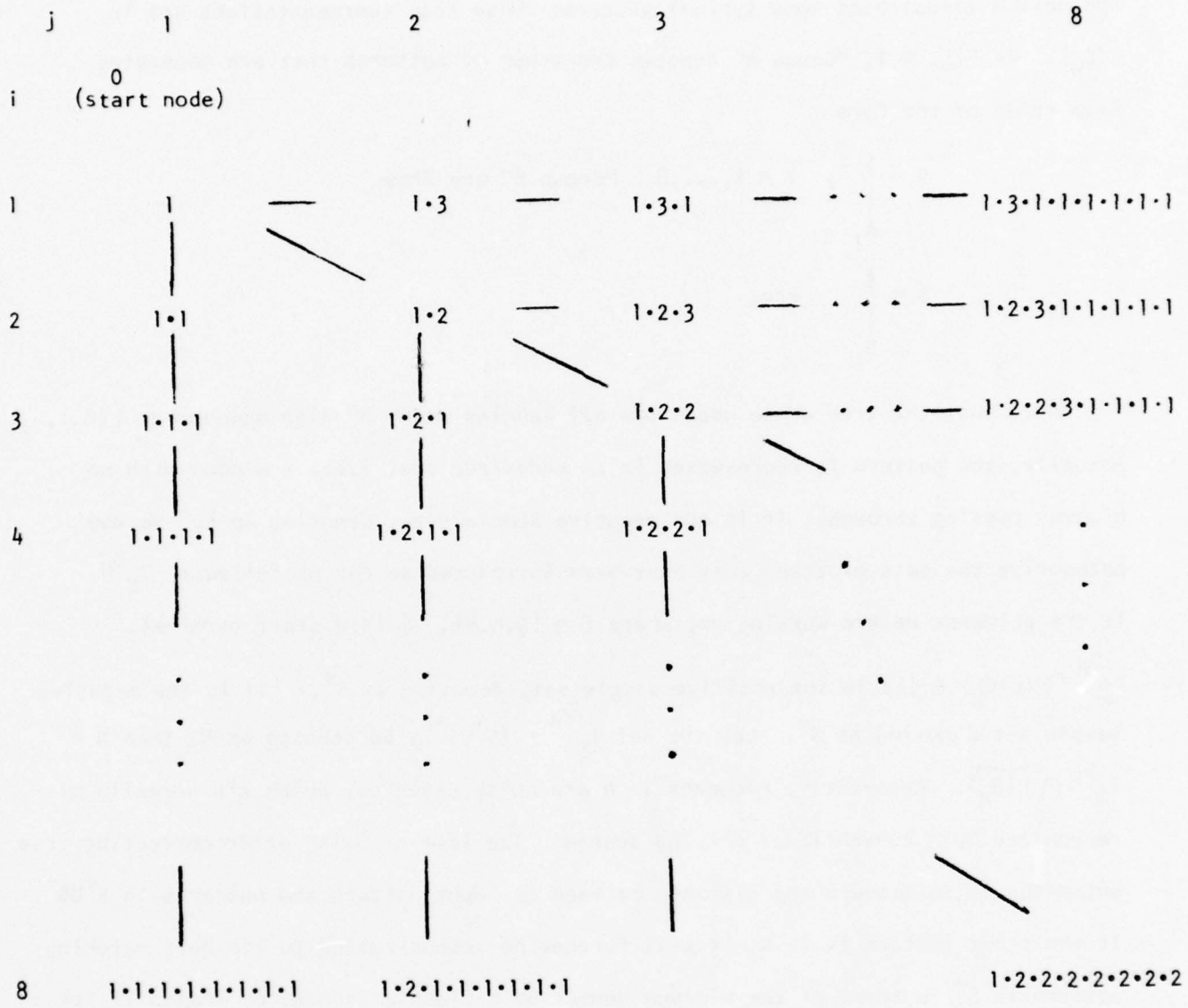



Fig. 4.3. Nodes in Tree Domain $D_H$ and their Corresponding Positions in the 8 x 8 Array

inferred to generate positive sample trees is given in Appendix A. Fig. A.1 in Appendix A illustrates some typical patterns whose tree representations are in $L(G_H)$. In Fig. A.1, "Group A" denotes the group of patterns that are generated from rules of the form

$$S \rightarrow \begin{array}{c} \$ \\ | \\ A_i \end{array} \quad , \quad i = 1,\ldots,8, \quad \text{"Group B" are from}$$

$$S \rightarrow \begin{array}{c} \$ \\ | \\ B_i \end{array} \quad \text{etc.}$$

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 $\lambda$. 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 $\Sigma = \{\$,h,b\}$. $\$$ is a 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 $S^-$. 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 misrecognized by 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 $S^+$ in terms of the minimum number of mislabeled nodes, 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 ECTA, 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. The pattern that 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 Col. 141-148, row 1-18 of Fig. 4.1, the window-by-window correcting process is illustrated in Fig. 4.4. The flowchart of the entire recognition procedure is given in Fig. 4.5.

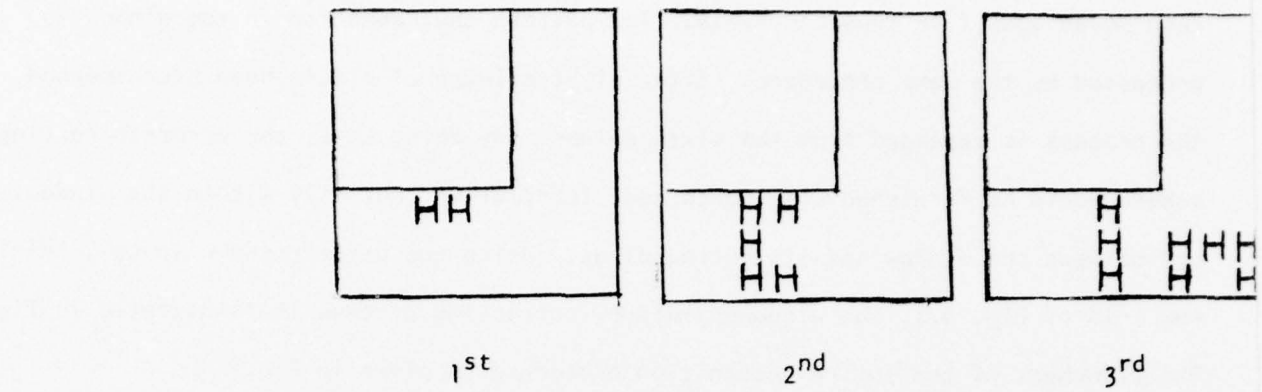
This error-correcting scheme of the highway recognition problem is programmed in Fortran TV on CDC 6500 computer and tested using the data shown in Fig. 4.1. The result is shown in Fig. 4.6. There are 80 x 160 pixels in the input data. The cpu time for processing is 150 sec.

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 Fig. 4.7 which contains 125 x 125 pixels. The result of the error-correcting analysis is shown in Fig. 4.8. The cpu time used is 101 sec. For comparison, we use the highway map shown in Fig. 4.9 as ground truth.

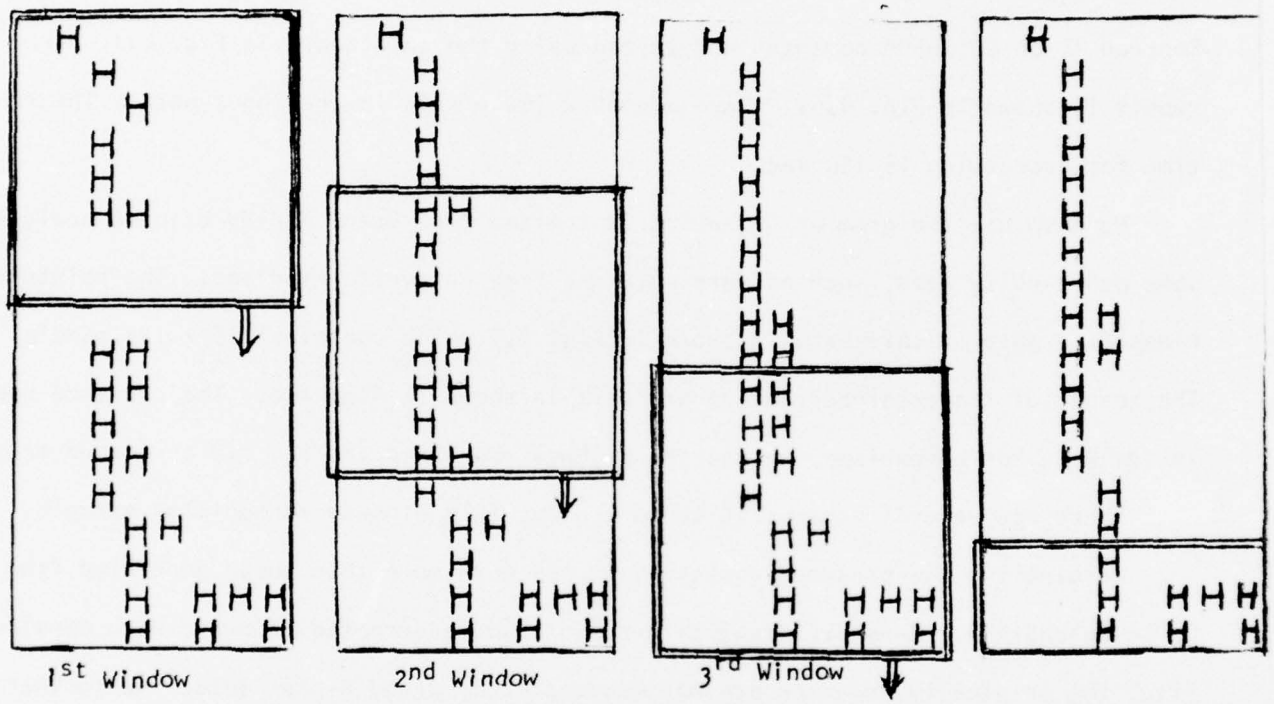
There are several remarks to be made about this highway recognition example.

Originally, the presumed positive samples were more than those generated from $G_H$ in Appendix A. Many different patterns of two intersected highways were considered. After the originally inferred 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 A.

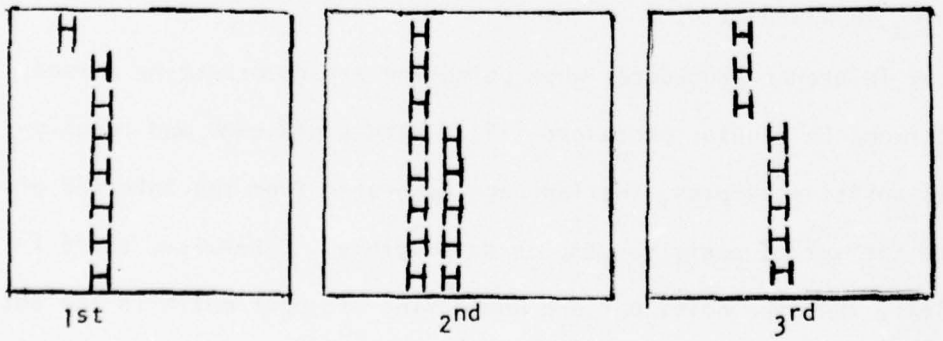
The grammatical inference procedure, when using the error-correcting parser, is a little different from the regular procedure [13]. With a presumed and roughly-sketched pattern as positive samples, the language generated from the inferred grammar must be as close to the set of positive samples as possible. Otherwise, there is always the possibility that the noisy pattern would find its best match in the set of illegitimate sentences (sentences in L(G) but not in $S^+US^-$).




(a) Erased Windows




(b) Before and After Window Scanning




(c) Corrected Windows

Fig. 4.4. Window-by-Window Correcting Process

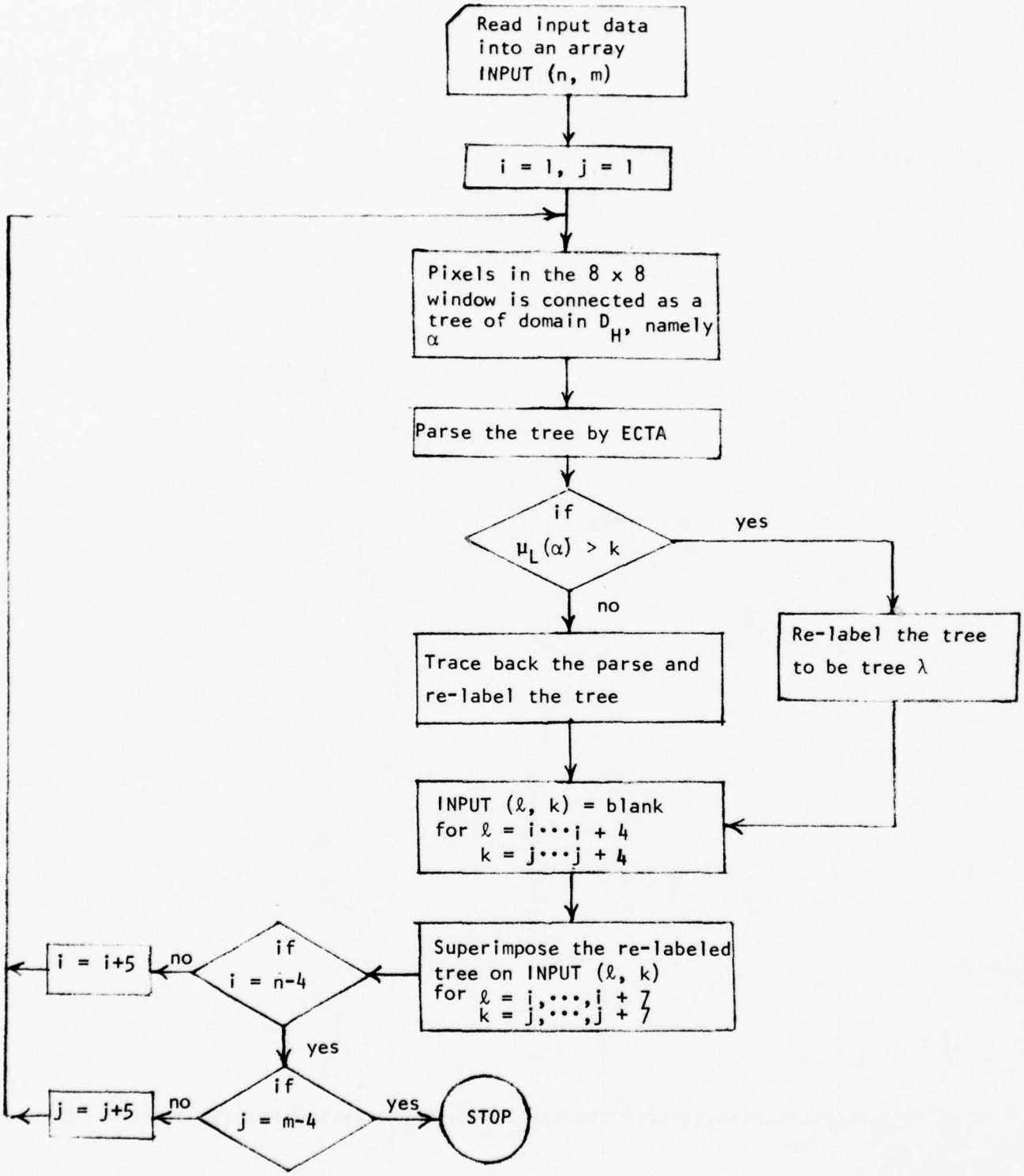



Fig. 4.5. Flow Chart of Highway Recognition Problem Using ECTA

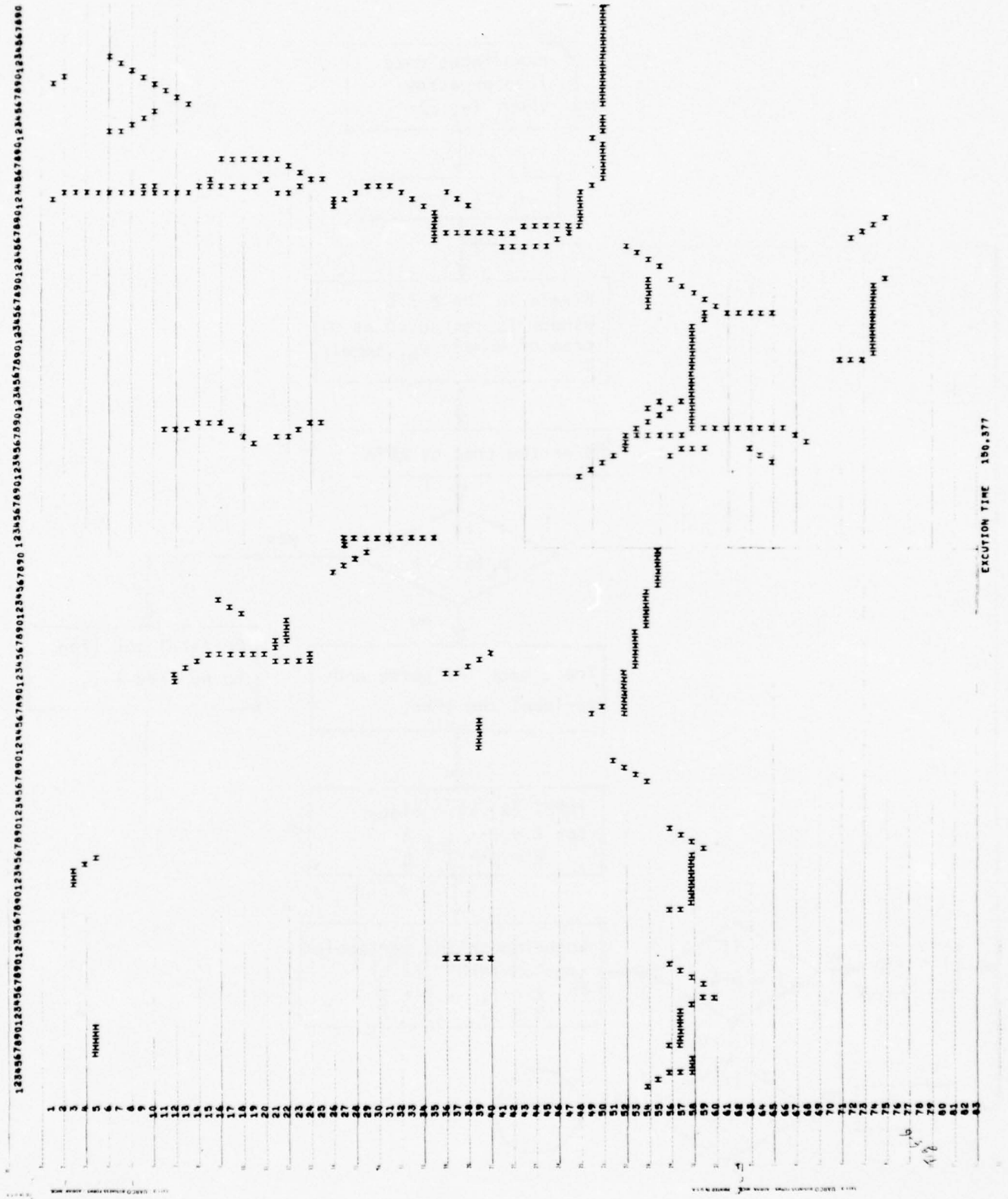



Fig. 4.6. ECTA Processed Result of the Grand Rapids area



Fig. 4.7. Pointwisely Classified Data of the Lafayette area



Fig. 4.8. ECTA Processed Result of the Lafayette area

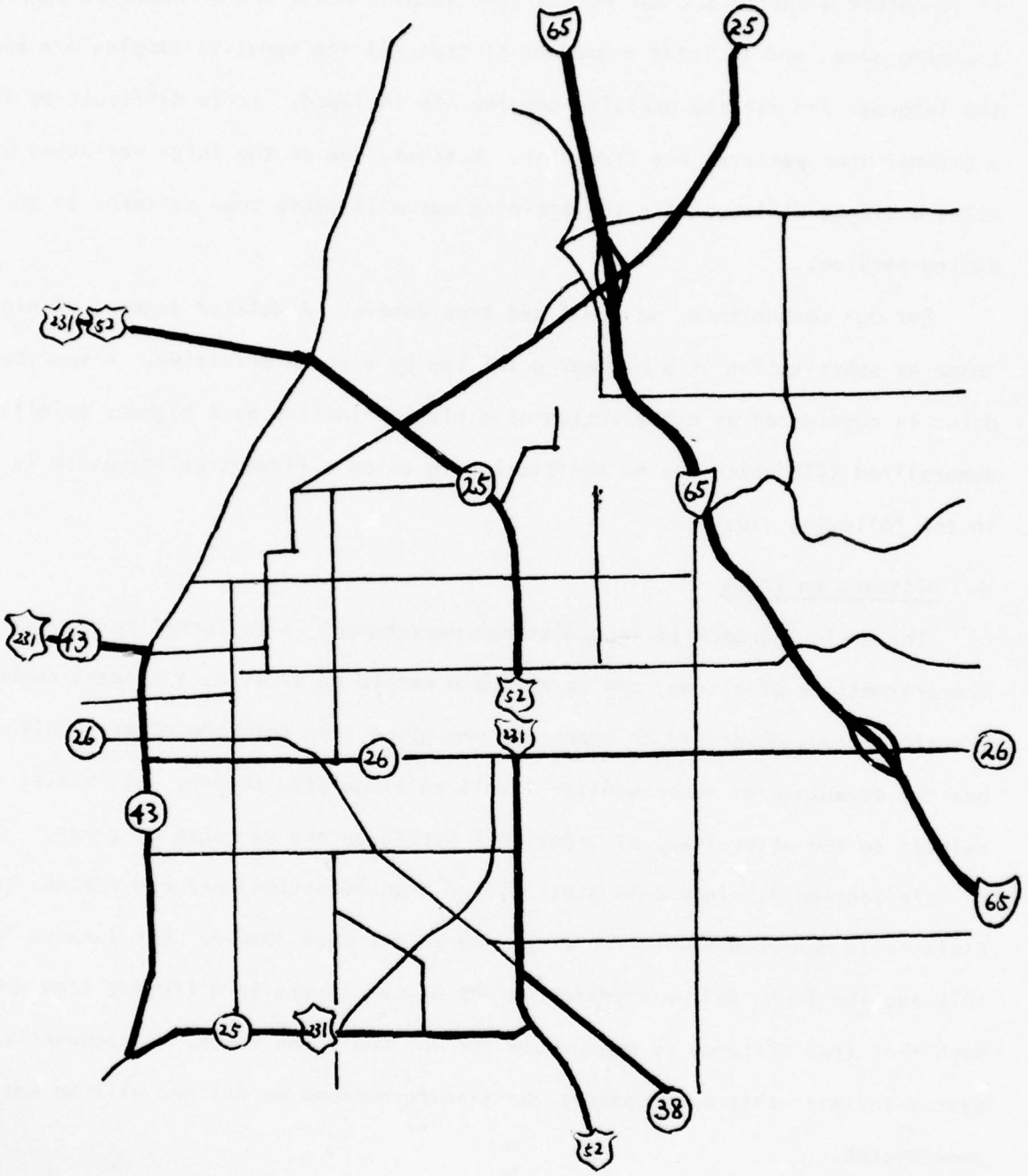



Fig. 4.9. Highway Map of Lafayette, Indiana

The alternative solution, if not using the error-correcting scheme, of course, is to obtain a sufficient set of positive samples and a set of negative samples from training data, and to infer a grammar so that all the negative samples are excluded from the language and all the positive samples are included. It is difficult to infer 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.

For our convenience, we use fixed tree domain. A deleted segment of highway is taken as 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 has no restriction on using a fixed-tree structure is introduced in the following sections.

## 5. Distance on Trees

The basic approach to tree distance measurement is to formalize a concept of transformations of a tree, and to define a metric on trees as the least number of transformations necessary to transform one given tree into the other. This approach has the advantage of accommodating itself to linguistic notion. It is also closely related to the natural way of imposing a metric on the vertices of graph.

In Section 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, a syntax analyzer that accommodates the transformations we defined will be easily constructed.

Extending the three types of syntax errors defined on strings, namely, substitution, insertion, and deletion, we define errors on trees to be the following five operations:

(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 types of transformation S, T, P, B, D, from $T_\Sigma$ to the subsets of $T_\Sigma$, to describe substitution, stretch, split, branch, and deletion errors respectively. Let $\alpha$ 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) $S_{a/y}(\alpha) = \alpha(a\leftarrow\{(0,y)\}\cup\{i\cdot\alpha/a\cdot i \mid 1 \leq i \leq r\ [\alpha(a)]\})$,

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

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

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

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

S, T, B, P and D transformations are illustrated in Fig. 5.1 (a), (b), (c), (d) and (e), respectively.

We write $\alpha \vdash\!\!\!\xrightarrow{\Delta} \beta$, if $\beta$ is in $\Delta(\alpha)$, where $\Delta\in\{S, T, B, P, D\}$, and further denote that $\alpha \vdash\!\!\!\xrightarrow{\Delta^k} \beta$ for $k \geq 0$ if $\beta$ is derived from $\alpha$ by applying k transformations where $\Delta^k$ denotes the composition of $\Delta$ with itself k times. The distance on trees over $\Sigma$, $\mu(\alpha,\beta)$, is defined as the smallest integer k for which $\alpha \vdash\!\!\!\xrightarrow{\Delta^k} \beta$, if $\alpha$ and $\beta$ are two trees in $T_\Sigma$. The function $\mu$ is symmetric and satisfies triangle inequality.

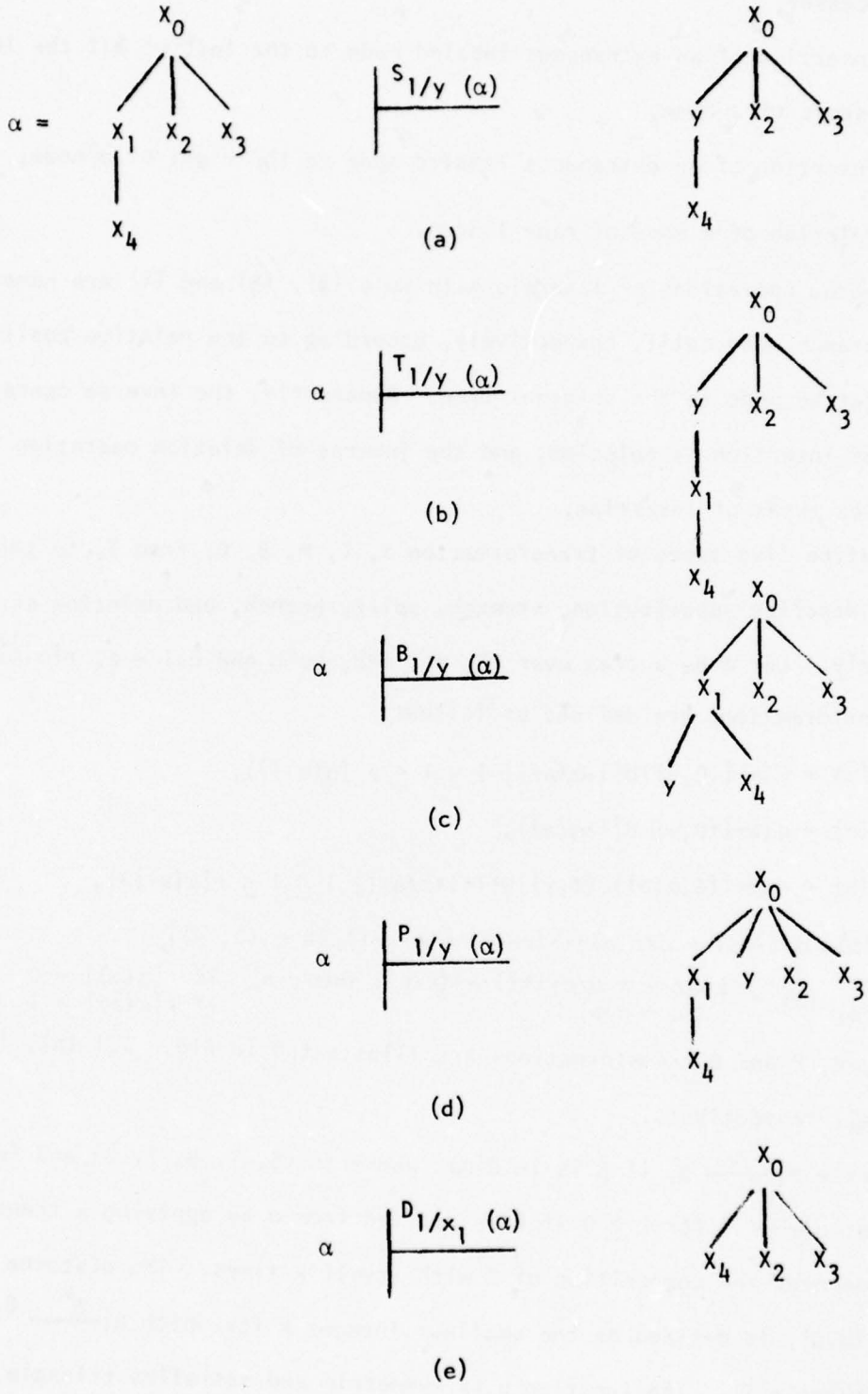



Fig. 5.1. S, T, B, P, and D transformations on α

For example, suppose $\alpha = \{(0,\$), (1,\sim), (1\cdot1, P), (2,\sim), (2\cdot1,q)\}$. Then given the tree $\beta = \{(0,\$), (1,V), (1\cdot1, P), (1\cdot2, q), (2, q)\}$, we can say that $\mu(\alpha,\beta) = 3$, because $\beta = D_{2/\sim}(P_{1\cdot1/q}(S_{1/v}(\alpha)))$ and no other derivation of $\beta$ from $\alpha$ costs transformations less than three. $\alpha$ and $\beta$ are shown in Fig. 5.2.

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

$$\mu_L(\beta) = \min_{\alpha}\{\mu(\alpha,\beta) \mid \alpha \in L\}.$$

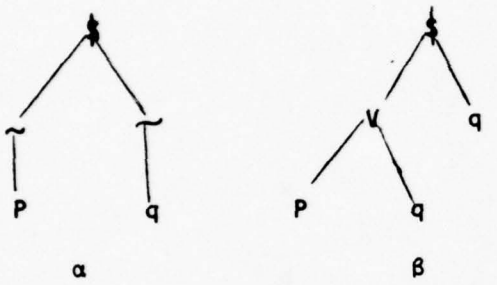



Fig. 5.2. $\beta = D_{2/\sim}(P_{1.1/q}(S_{1/v}(\alpha)))$.

When a tree $\beta$ is inserted between subtrees with roots at $a_1$ and $a_2$ of tree $\alpha$, respectively, the root of $\beta$ is said to be caused by a split error of $\alpha(a_1)$, and the rest nodes in $\beta$ are inserted by sequence of branch and/or split operations. Similarly, when $\beta$ is inserted to the left of all the immediate successor of node a of $\alpha$, then the root of $\beta$ is said to be caused by a branch error of $\alpha(a)$. If $\alpha/a$ matches subtree $\beta/b$ of $\beta$, $b \neq 0$, i.e., $\alpha/a = \beta/b$, then the immediate predecessor of b is said to be caused by a stretch error of $\alpha(a)$ followed by a sequence of stretch, branch, and/or split operations. When deletions cause a branch of the resulting tree to be a subtree of the corresponding branch of the original tree, nodes at the lowest level will first be deleted, then followed by their predecessors.

Since each grammar rule eventually generates a branch, the tree metric is related to linguistic model in such a way that, if $\alpha/a$ is assumed to be $\Psi$, and $X_0 \overset{*}{\underset{G_t}{\Longrightarrow}} \Psi$, where $G_t$ is the given tree grammar, then $\alpha/a$ is represented by $X_0$. This procedure can actually be considered as pruning, although $\alpha/a$ and $\Psi$ may be different.

EXAMPLE 5.1 Measure the distance between acyclic directed graphs by the proposed tree metric.

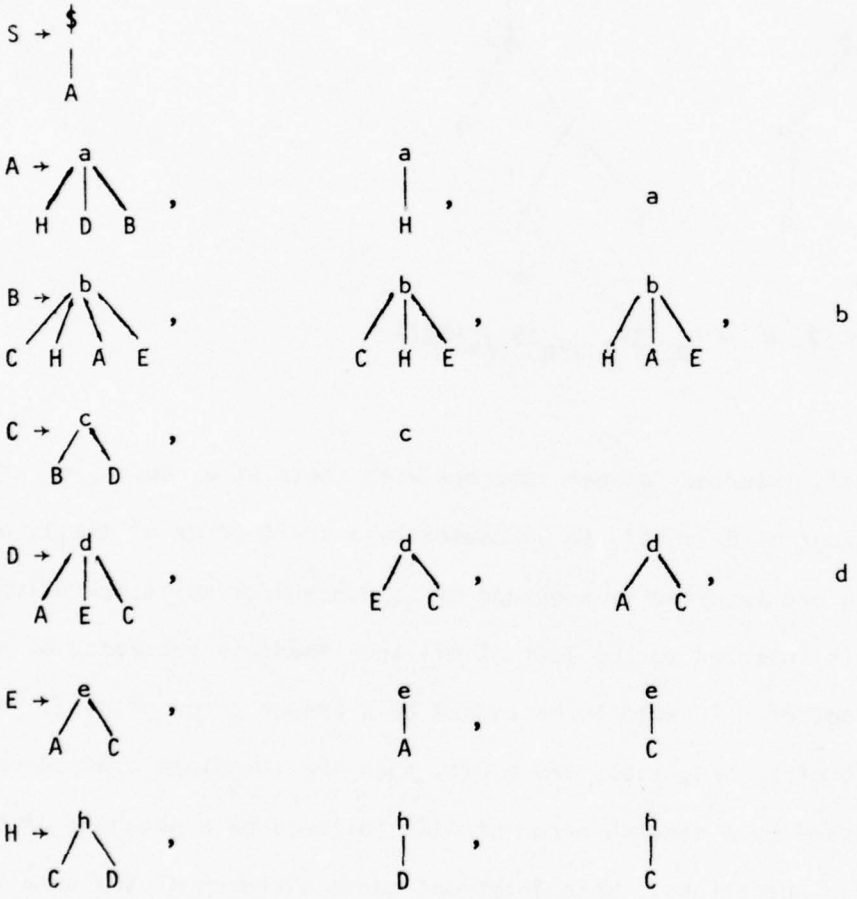
Assume a directed graph of labeled vertice and unlabeled branches as shown in Fig. 5.3, the tree grammar inferred to generate such patterns is given as follows.

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

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

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

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




Suppose that $\alpha'$ is the given distorted pattern. Following the grammatical rules, graphs after each tree error transformation used are illustrated in Fig. 5.4.

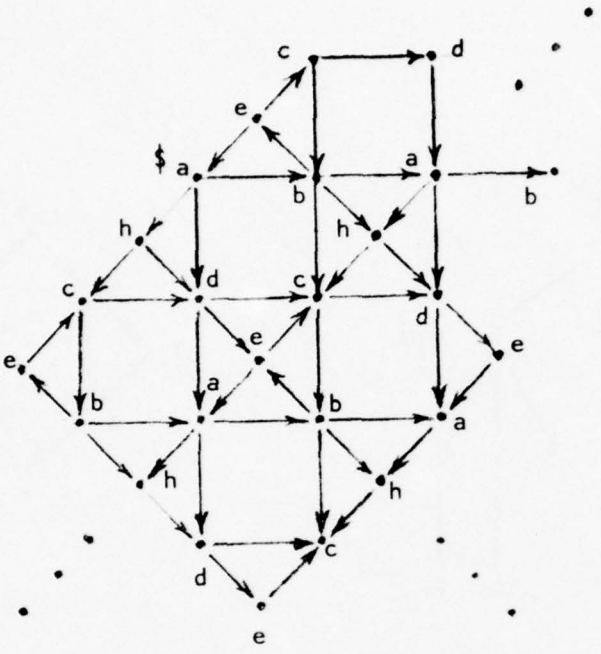
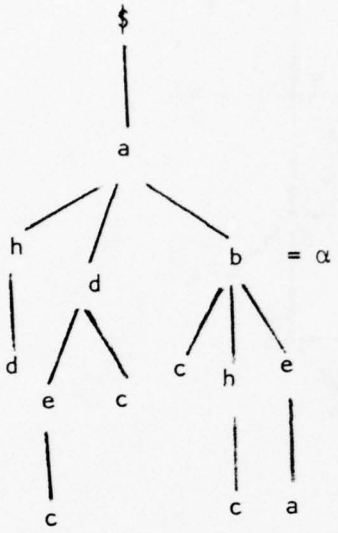
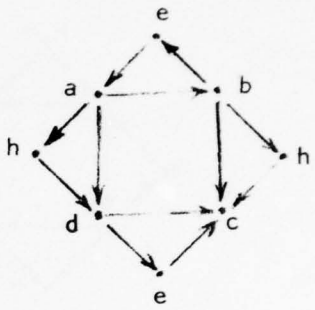
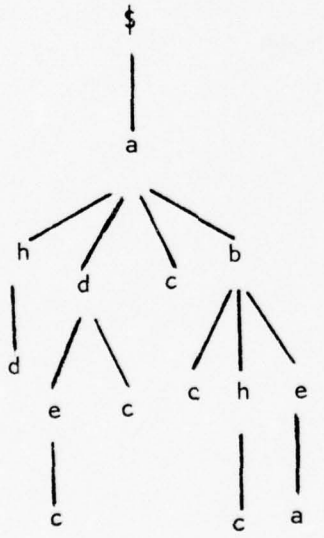


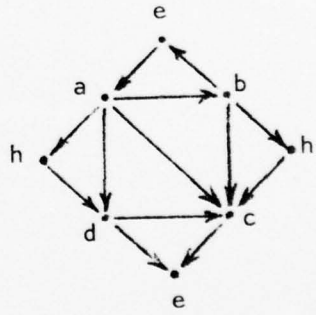

Fig. 5.3. A Directed Graph

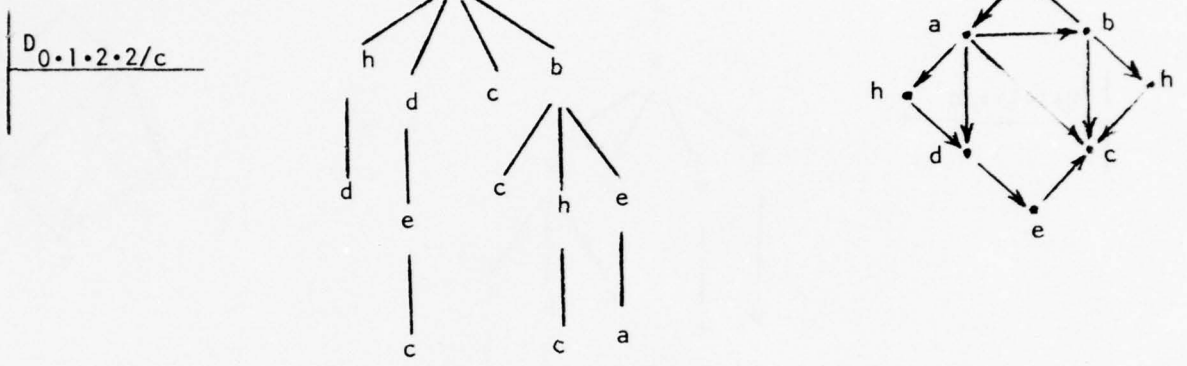


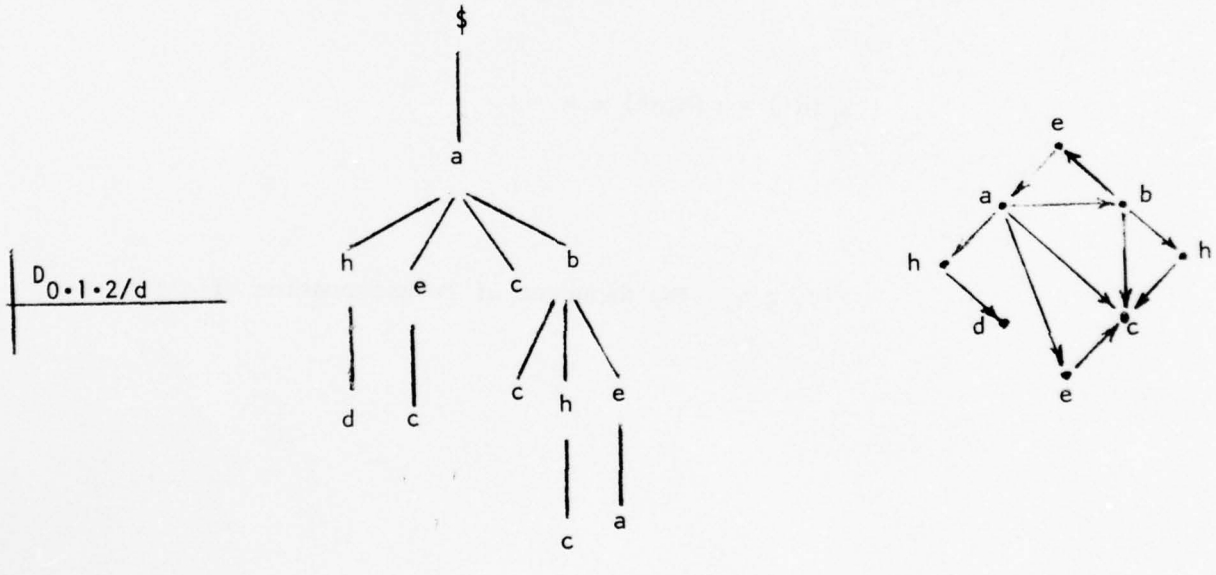

$P_{0 \cdot 1 \cdot 2/c}$



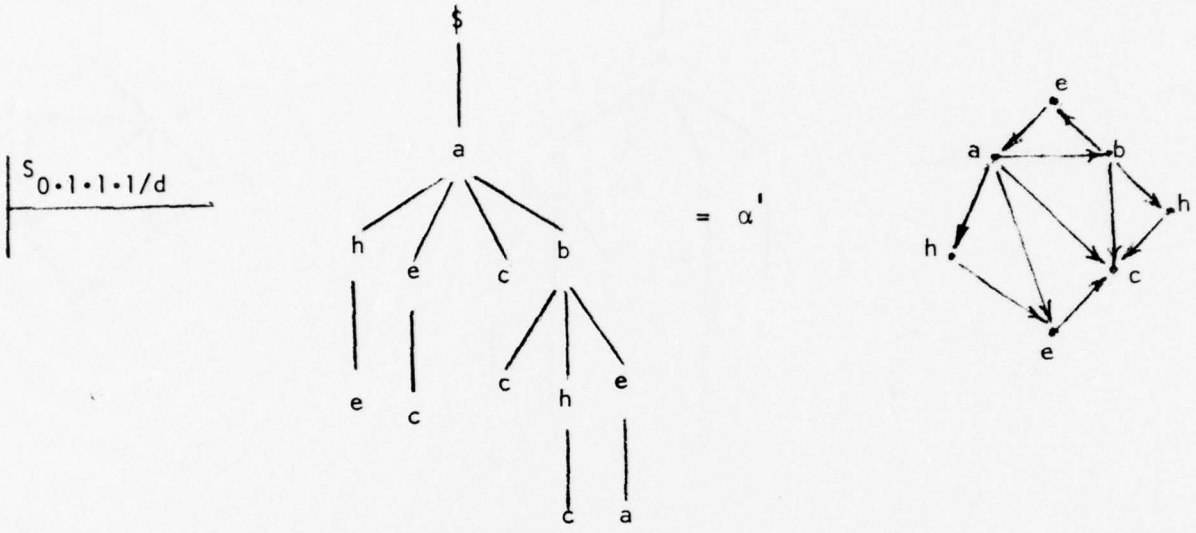






$D_{0 \cdot 1 \cdot 2 \cdot 2/c}$




$D_{0 \cdot 1 \cdot 2/d}$




$\mu_L(\alpha') = \mu(\alpha,\alpha') = 4$

Fig. 5.4. The Sequence of Transformation of $\alpha'$ from $\alpha$

## 6. Generalized Error-Correcting Tree Automata (GECTA)

As discussed in Section 3, for any given $\alpha'$ not in language L, the essence of an error-correcting tree automaton is to search for $\alpha$ in L such that the distance of $\alpha$ from $\alpha'$ is the smallest among all the sentences in L. That is,

$$\mu(\alpha,\alpha') = \min_{\beta \in L} \mu(\beta,\alpha') \tag{6.1}$$

Note that the restriction $D_\alpha = D_{\alpha'}$ in eq. (3.1) has been removed here.

A search algorithm for the best match solution based on grammar rules and error transformations defined in Section 5 will be studied in this section. We called it generalized error-correcting tree automaton (GECTA) to distinguish from the ECTA of Section 3. Before introducing the GECTA, a normal form of trees and tree grammars, called the binary form, is defined.

### 6.1 Tree Binary Form

We first define a binary tree grammar as follows:

DEFINITION 6.1. A tree grammar $G_b = (V_b, r_b, P_b, S)$ over $<\Sigma_b, r_b>$ is said to be in binary form if,

(1) A pseudo symbol * is in $\Sigma_b$.

(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 $V_b - \Sigma_b$, $x \in \Sigma_b - \{*\}$.

Let $\alpha$ be a tree in $T_\Sigma$, and * be a pseudo symbol not in $\Sigma$, the conversion of tree $\alpha$ into its binary form $\alpha^*$ is given in the following Algorithm.

ALGORITHM 6.1. Binary Form Conversion.

Input: Tree $\alpha$ in $T_\Sigma$.

Output: $\alpha^*$, binary form of $\alpha$.

Method: Repeat CONVERT until $\alpha$ is in binary form.

<CONVERT>: (1) If $t_0 = x$ is a term in $\alpha$, 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 $\alpha$, $x \in \Sigma_n$ 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

$$t_1^* = (t_1)*$$

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

$$\vdots$$

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

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

The construction of the binary tree grammar $G_b = (V_b, P_b, r_b, S)$ from a given tree grammar $G_t = (V, r, P, S)$ 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 $X_0 \to x$, $x \in \Sigma_0$, add $X_0 \to x$ to $P_b$ and $r_b(x) = 0$,

Step 2. For each production in P of the form $X_0 \to X_1 x$, $x \in \Sigma_1$, add $X_0 \to X_1 x$ 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,

$U_{01} \rightarrow X_1 *$

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

$\vdots$

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

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

to $P_b$, and $r_b(*) = \{1, 2\}$, $r_b(x) = 2$. Add $\{U_{0i} | 1 \leq i \leq n-1\}$ 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 6.1. Character E

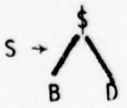
Using tree representation of character E, $\alpha$, and its inferred tree grammar $G_t$, the binary form of $\alpha$ and $G_t$, namely, $\alpha^*$ and $G_b$, are illustrated. Pattern primitives, pattern of character E, and its corresponding tree representation $\alpha$, are shown in Fig. 6.1 and 6.2(a), (b), respectively. The tree grammar $G_t$ which generates character E of different sizes is

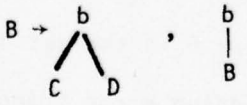
$G_t = (V, r, P, S)$ over $<\Sigma, r>$

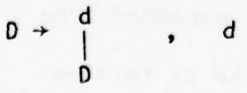
$V = \{S, B, C, D, \$, b, d\}$ $\Sigma = \{\$, b, d\}$

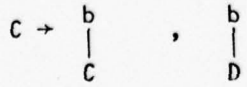
$r(\$) = 2$, $r(b) = \{1, 2\}$, $r(d) = \{0, 1\}$

P:



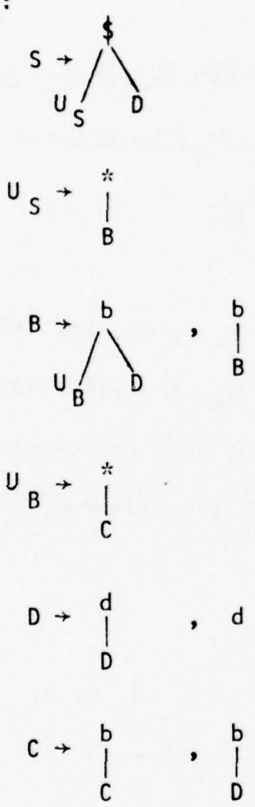
The converted binary form of $G_t$ is as follows:

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

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

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

$P_b$:




The binary form of $\alpha$, $\alpha^*$, is shown in Fig. 6.3.

## 6.2 Minimum-Distance GECTA

Let $M_b = (V_b\text{-}\Sigma_b, F, \{S\})$ be a tree automaton that accepts a tree grammar in binary form $G_b = (V_b, r_b, P_b, S)$ over $\Sigma_b$, where F is the set of transition functions on $(V_b\text{-}\Sigma_b)^{r(x)} X (V_b\text{-}\Sigma_b)$, for all $x \in \Sigma_b$. By adding error transitions according to the transformations defined in Section 5, the expanded tree automaton that accept all the possible erroneous trees is constructed as follows:

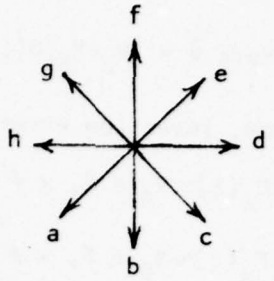



Fig. 6.1. Pattern primitives

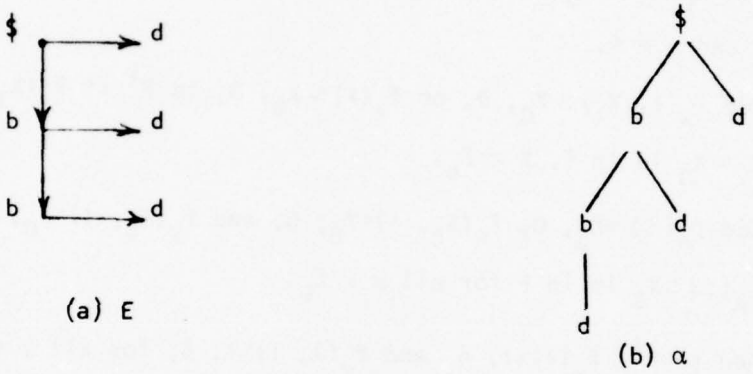



Fig. 6.2. Character E and its tree representation

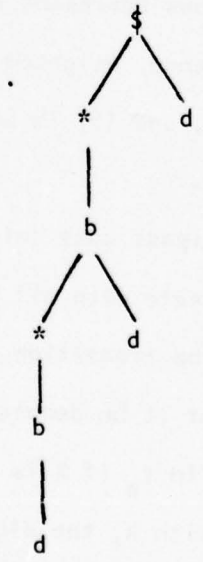



Fig. 6.3. α*, Binary form of α

ALGORITHM 6.2. Expanded Tree Automaton $\bar{M}_b$.

Step 1. $\bar{M}_b = (\bar{V}, \bar{F}, \{S\})$ where $\bar{V} = (V_b - \Sigma_b) \cup \{I\}$, $\bar{F} = F \cup F^S \cup F^D \cup F^I$, $F^S$, $F^D$, $F^I$ are called substitution, deletion, 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. $F^D = \{ f(\xi) \sim X_0, \nu | F_x(\xi) \sim X_0 \in F, x \neq * \} \cup$
$\{ f(\xi) \sim X_0, 0 | F_* (\xi) \sim X_0 \in F \}$

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 $f_x(I, X_1) \sim X_0$, 0, or $f_x(I) \sim X_0$, 0, 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, $f_*(X_0, I) \sim X_0$, 0, and $f_y(X_0, I) \sim X_0$, $\sigma$ to $F^I$ if $f_x(\xi) \sim X_0$ is in F for all $x \in \Sigma_b$.

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

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

$\sigma$, $\nu$, and $\delta$ of error 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 state 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 $\alpha'$ be the input tree, then a pair $(X, \eta)$ is in $t_a$ if X is a candidate state representing $\alpha'/a$, $\eta$ is the minimum cost associated with X, the algorithm is given as follows:

ALGORITHM 6.3. 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 $\alpha'$.

Output: $\mu_{L(G_t)}(\alpha')$ and transition table of $\alpha'$.

Method:

<SUBTREE(Start)>

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

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

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

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

<SUBTREE(X, $\theta$)>

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

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

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

(d) Delete redundant states.

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

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

(b) $(X_0, \eta)$ is added to $t_a$ if $f_x(X_1, X_2) \sim X_0$, $\lambda \in FUF^S \cup F^I$, $(X_1, \pi_1)$ 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) $(X_0, \eta)$ is in $t_a$ if $f_x(Y) \sim X_0$, $\lambda \in FUF^S \cup F^I$, for all $(Y, \pi) \in$ SUBTREE $(X, \theta)$ and $(X, \theta) \in t_{a.1}$, $\eta = \lambda + \pi$.

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

(c) Delete redundant states.

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

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

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

(c) Delete redundant states.

For all states in $M_b$, a table of minimal deleted tree is first computed by the two subroutine SUBTREE(Start) and SUBTREE$(X, \theta)$. $(X_0, \eta)$ is generated from subroutine SUBTREE(Start), if $X_0 \overset{*}{\underset{G_b}{\Longrightarrow}} w$, and w is the smallest subtree among all the possible derivations starting from nonterminal $X_0$. By referring to the smallest subtree, we mean the number of nodes in the subtree is the least. Similarly, $(X_0, \eta)$ is generated from subroutine SUBTREE$(X, \theta)$ if $X_0 \overset{*}{\underset{G_b}{\Longrightarrow}} \Psi$, and if all the nodes in $\Psi$ are terminal nodes except one frontier, it is labeled with nonterminal $X_0$. Furthermore, $\Psi$ 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 actual 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 $(X, \theta) \in t_{a.1}$, then $(Y, \pi)$ must be in SUBTREE$(X, \theta)$. Let $\beta$ be the tree represented by state X and $\alpha$ be the tree represented by state Y, then $\beta$ is a subtree of $\alpha$, odes in $\alpha$ but not in $\beta$, are assumed to be deleted. The cost of deletion is $\pi-\theta$ which is minimum among all the possible deletions. All the insertions are handled by transitions in $F'$ automatically.

EXAMPLE 6.2. Recognition of Hand-Written Character E

Suppose a casually-written character E, as shown in Fig. 6.4(a), whose binary tree representation $\alpha'^*$ is shown in Fig. 6.4(b), is to be recognized automatically. The expanded tree automaton constructed from the tree grammar $G_E$ in Appendix B-1 is as follows:

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

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

$\bar{F}$:

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

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

$f_{b}(U_B, D) \sim B, 0$

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

$f_{d} \sim D, 0$

$f_{b}(D) \sim C, 0$

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

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

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

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

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

$f(C) \sim U_B, 0$

$f \sim D, \nu$

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

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

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

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

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

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

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

$f_b(I) \sim D, 0$

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

$f_b(I, D) \sim C, 0$

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

$f_*(U_S, D) \sim S, 0$

$f_*(S, I) \sim S, 0$

$f_\$(S, I) \sim S, 0$

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

$f_*(U_B, D) \sim B, 0$

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

$f_b(B, I) \sim B, 0$

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

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

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

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

$f_b$ (C, 1) $\sim$ C, 0

$f_y$ (C, 1) $\sim$ C, $\sigma$ $\quad y \in \{a,d,h\}$

$f_y$ (1) $\sim$ 1, $\delta$ $\quad y \in \{a,b,d,h,*\}$

$f_y$ (1, 1) $\sim$ 1, $\delta$ $\quad y \in \{a,b,d,h,*\}$

$f_y \sim 1, \delta$ $\quad y \in \{a,b,d,h\}$

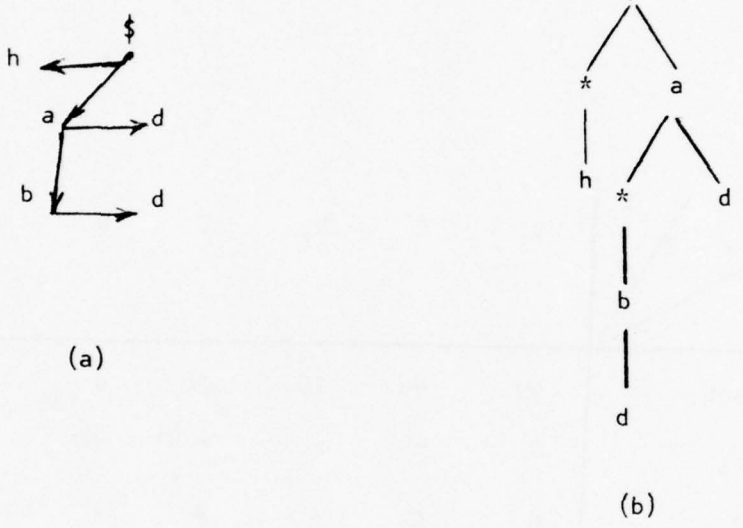


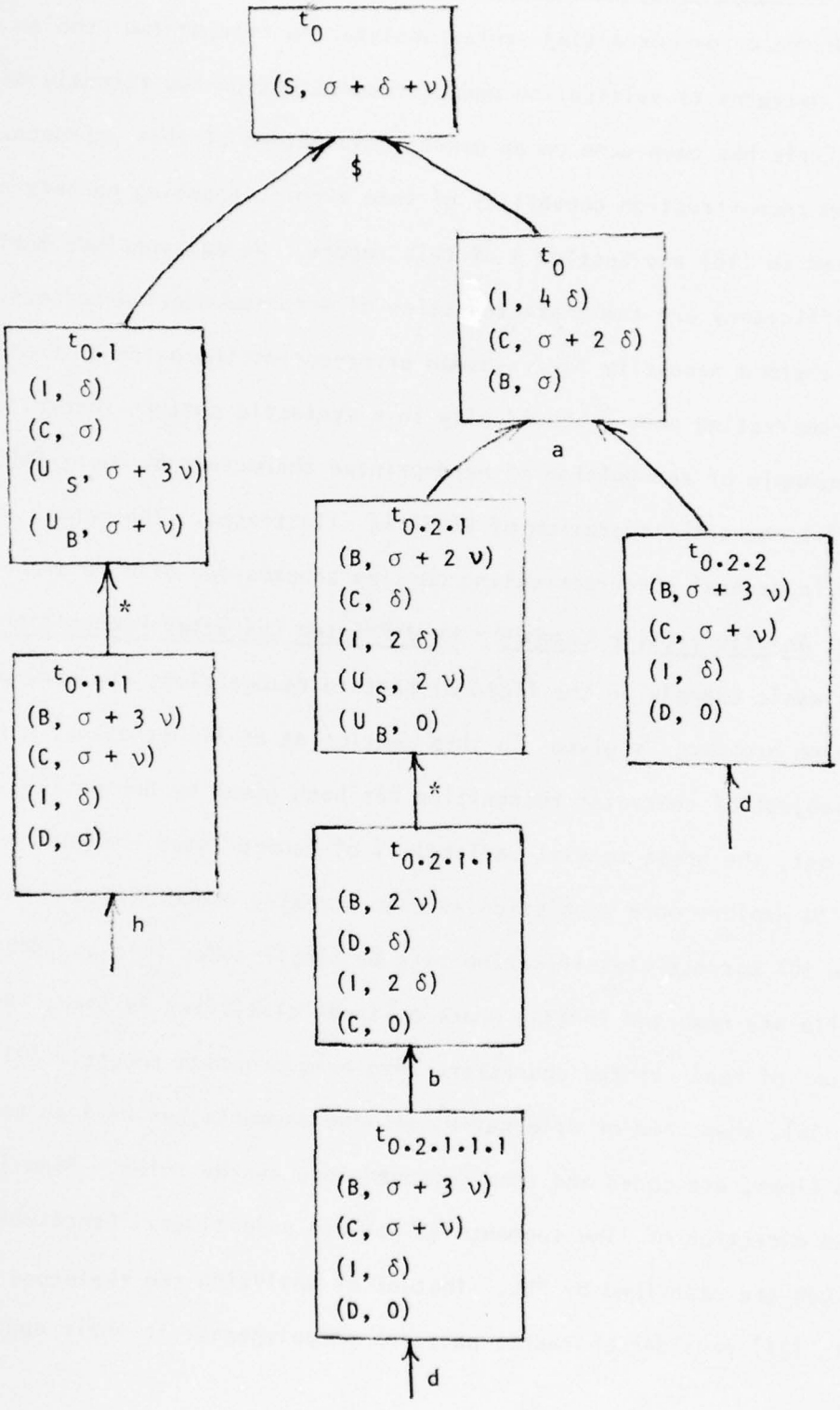

Fig. 6.4. Distorted Character E and its Binary Tree Representation $\alpha'^*$

By using GECTA, the computation of the transition table of $\alpha'^*$ is illustrated in Fig. 6.5(a) and (b). From Fig. 6.5(b), since (S, $\sigma$ + S + $\nu$) is in $t_0$, we have $\mu_{L(G_t)}(\alpha') = \sigma + \delta + \nu$.

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

$(Y, \pi) \in \text{SUBTREE}(X, \theta)$

(a) Table of Minimal Deleted Trees




(b) Transition table of $\alpha'^*$

Fig. 6.5. The Acceptance of $\alpha'^*$ by GECTA

## 7. Application of Error-Correcting Parsing Schemes to Pattern Recognition

Using the error-correcting syntax analyzer in solving the problem of noisy and/or distorted patterns classification and/or reconstruction has recently been suggested [23]. So far, little has been done on an overall evaluation of this approach. The classification and reconstruction capability of some error-correcting parsers have been illustrated in [18] and Section 4 of this report. We may conclude that high accuracy and low efficiency are the characteristics of error-correcting parsers. The questions are: Is there a necessity for using an error-correcting parser? What is the best role an error-correcting parser should play in a syntactic pattern recognition system?

An example of recognition of hand-printed characters by using GECTA is given in Section 7.1 where the operation of GECTA is illustrated. The merits and weaknesses existing in general error-correcting parsing schemes can also be shown by this example.

### 7.1 An Illustrative Example - Hand-Printed Character Recognition

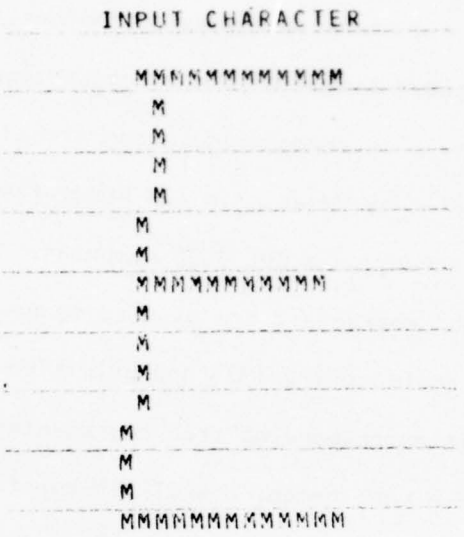
A classic example in the field of pattern recognition, the hand-printed character recognition problem, is given in this section as an illustration. A thorough survey on the subject of character recognition has been given by Munson [34]. As Munson pointed out, the great spatial variability of hand-printed characters has led research workers to explore more sophisticated preprocessing and/or feature extraction methods. Nearly a 90% correct classification rate on single-coder file and 80% on multiple-coder file are reported in [34] where a linear classifier is used. Syntactic descriptions of hand-printed characters have been proposed recently [31, 35, 36, 37]. In [35, 36], shape and/or orientation of line segments, as well as spatial relation between lines, are coded and then composed into syntax rules. Shaw [37] uses variant size and direction of line segments as pattern primitives. Concatenate relations between primitives are described by PDL. Instead of analyzing the skeletons of patterns, Feng and Pavlidis [31] consider character patterns as polygons. In their approach, pattern

primitives are the simpler components obtained from decomposition of polygons. During decomposition, juxtaposition relation results in a labeled graph. The resulting descriptions are generally size and rotation invariant; thus, error tolerable to some extent. The decomposition algorithm, which is implemented in Fortran on an IBM 360/91 computer, costs approximately 4 sec. to process a single Chinese character. Stallings [38] encoded components of Chinese character by using contour tracing technique and Freeman's chain code. Information about direction and concatenation of strokes are contained in the resulting code strings. For recognition accomplished by using a decision tree search method, it is reported by Narasimhan and Reddy [36] that the average processing time per character is 6 seconds on a CDC 3600 computer.

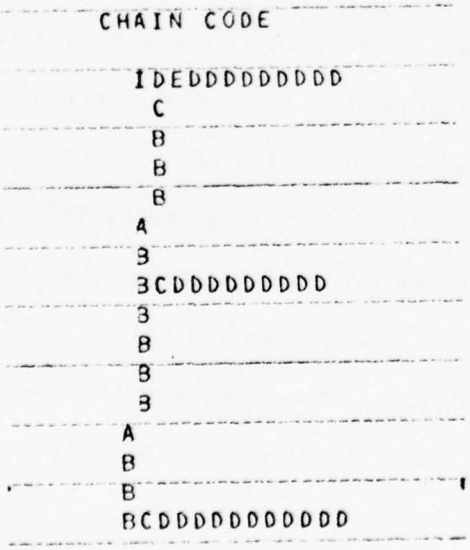
In this section, input characters are assumed to be digitized line patterns in a 16 x 16 format. After chain coding [32] and primitive extracting, input patterns are transformed into their corresponding tree representations. A set of grammars for sample characters are given; hence, a GECTA is constructed. Input patterns are then analyzed by the GECTA and classified based on minimum-distance criterion.

An example of an input pattern is shown in Fig. 7.1(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 Fig. 7.1(b). In Fig. 7.1(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 assigned in Fig. 6.1. 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 Fig. 7.1(c).

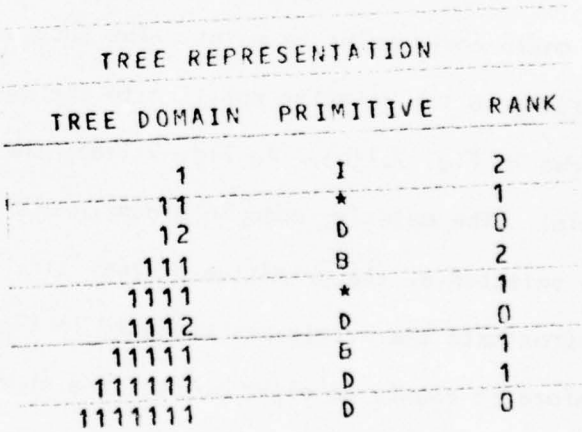
The leftmost 1 of each strings in the column entitled as "tree domain" corresponds to the identity of U in Definition 2.1. The rest symbols in the string are added according to Definition 2.2. Therefore, 1, 11, 12, 111,---,etc. in Fig. 7.1(c) correspond to tree domain 0, 0·1, 0·2, 0·1·1,---, etc. by Definition 2.2.




(a)




(b)




(c)

Fig. 7.1. Input Format, Chain Code Result, and Binary Tree Representation

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 Fig. 7.2. Their respective tree grammars are given in Appendix B.1.

A total of 26 patterns, casually or meticulously printed characters, are tested. Assume that $\sigma = \nu = \delta = 1$, the input patterns, their tree representations, as well as classification results and their distance from the assigned sample pattern are given in Appendix B.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 illustrated as "Time used for parsing," The average recognition time is 4.1 sec. per character.

These 26 test patterns are also processed by the tree automata constructed for grammars $G_A$, $G_C$, $G_D$, $G_E$, $G_H$ shown in Appendix B.1. Evidently, a pattern is acceptable only when its distance from a sample pattern is zero (see Appendix B.2). Therefore, 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 the non-error-correcting scheme, GECTA has the potential of recognizing highly distorted patterns, such as cursive handwritings. Due to the exhaustive search method that the error-correcting parsing scheme employs, the main drawback is its slow processing time. (Some simple techniques we suggest in [18] can be used to reduce the processing time significantly.)

## 7.2 Conclusion

The first question we posed at the beginning of this section, the necessity of using an error-correcting parser in pattern recognition, can possibly be answered by the remarks we gave in Section 4. When reconstruction of noisy and/or distorted

Character

Tree Representation

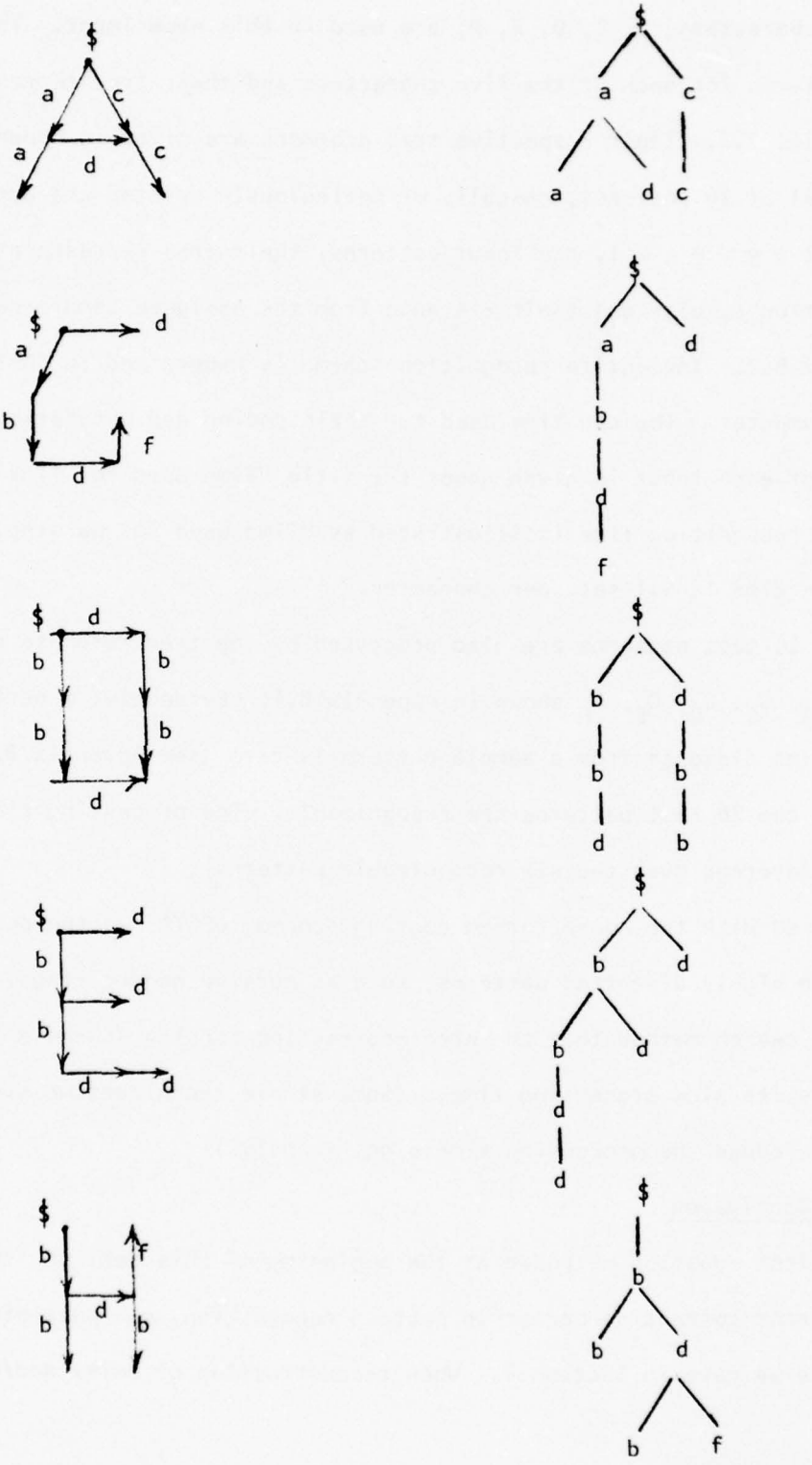



Fig. 7.2. Sample Patterns of Character A, C, D, E, H

patterns is required, or when it is too cumbersome to infer a grammar to generate most of the irregular patterns, the use of an error-correcting parser seems to be necessary.

The main purpose of Section 7.1 is not to solve the problem of hand-printed character recognition. Rather, it demonstrates a classification problem where the number of classes and parameters of each class, i.e., grammars, are known. Yet, no learning was conducted in this example. That is, the pattern grammars were not modified after a new pattern was classified. Due to the inefficiency of error-correcting parsers, we prefer to have pattern grammars to be able to represent most shape and/or size variations that exist in the same class of patterns. In many cases, training data does not provide sufficient information. A feedback between recognition and analysis in a syntactic pattern recognition system [12] that provides updated information can be conducted by adding an error-correcting parser. A block diagram of the feedback system is shown in Fig. 7.3.

A similar approach can also be used in designing a decision-directed learning system where correct classifications of learning observations are unknown. A flowchart of this learning system is illustrated in Fig. 7.4. The error-correcting syntax analyzer in such a system provides the grammatical inference machine the classifications of input samples.

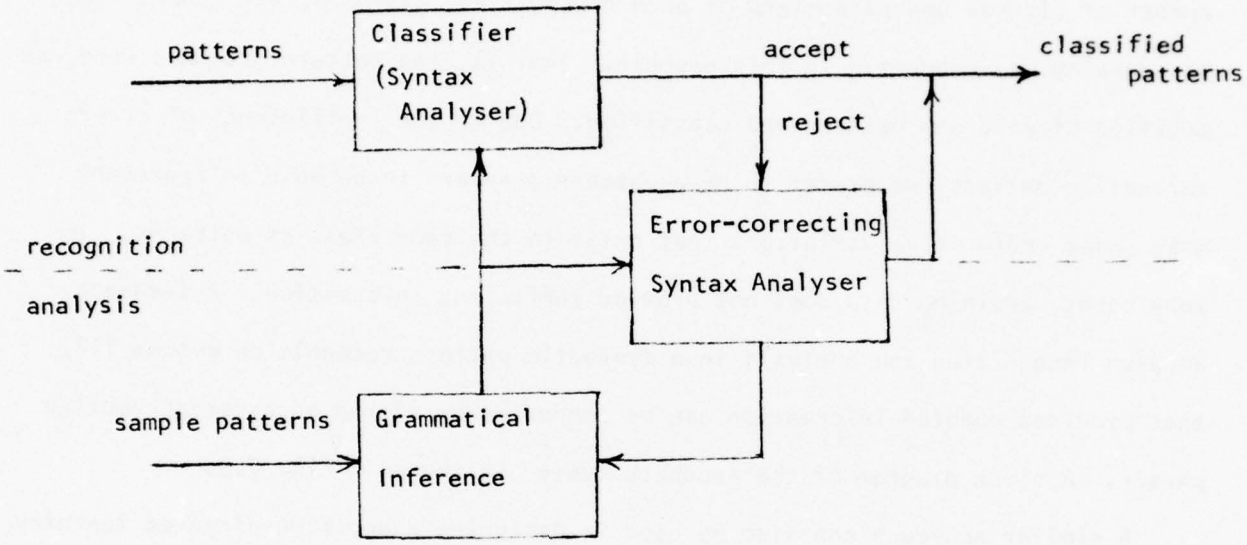



Fig. 7.3. Block Diagram of Feedback Pattern Recognition System

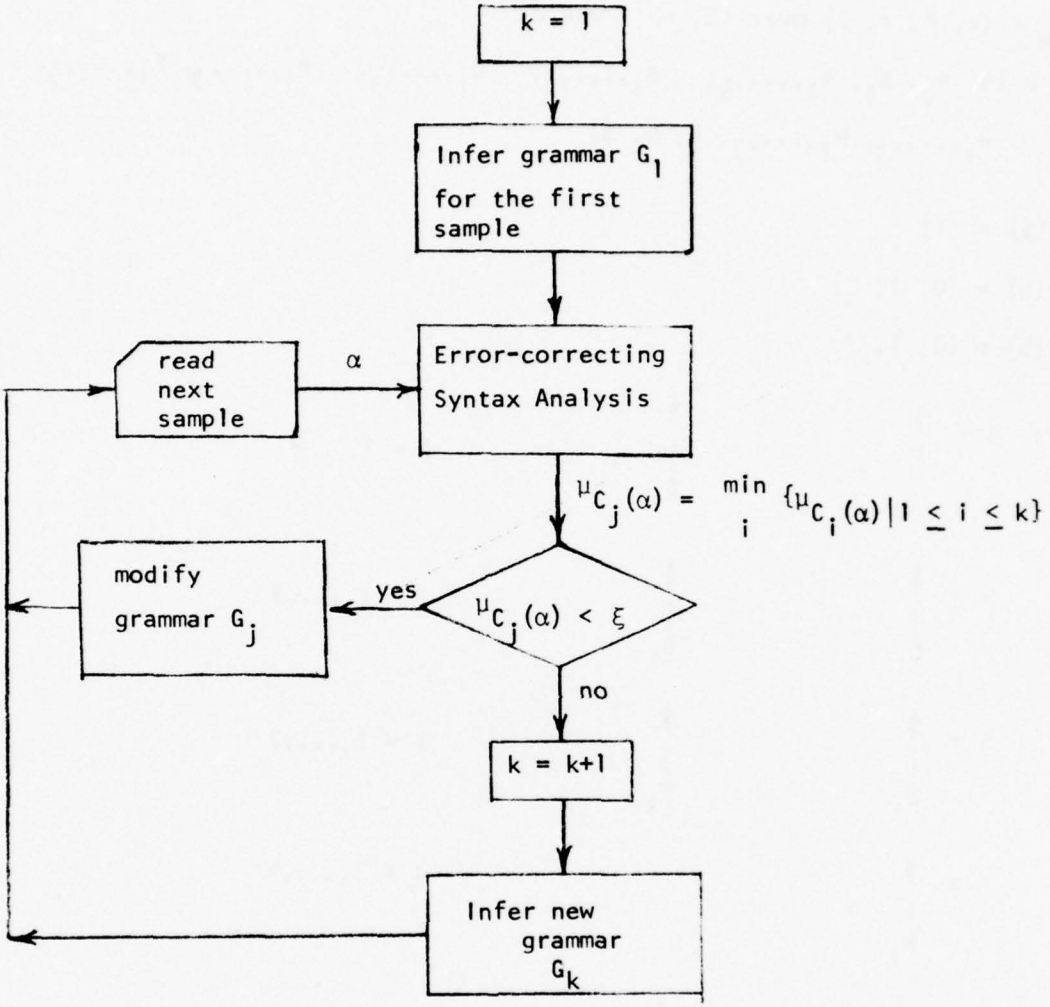



Fig. 7.4. Decision-Directed Learning System

Appendix A Highway Grammar

$G_H = (V, P, r, S)$ over $\langle \Sigma, r \rangle$ where,

$V = \{S, H_0, X_0, A_{1,\ldots,8}, B_{1,\ldots,8}, C_{1,\ldots,3}, D_{1,\ldots,3}, E_{1,\ldots,7}, F_{1,\ldots,7},$
$H_{1,\ldots,7}, M_{1,\ldots,7}, L, b, \$\}$

$r(\$) = \{1\}$

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

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

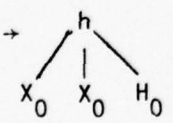
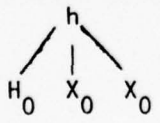
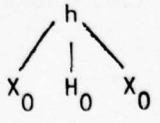
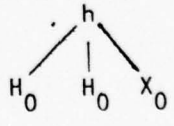
P: $S \rightarrow \begin{matrix}\$\\ |\\ A_i\end{matrix}$ , $\begin{matrix}\$\\ |\\ B_i\end{matrix}$ , $i = 1, \ldots, 8$

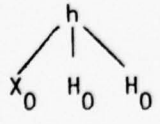
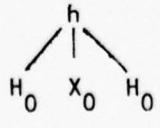
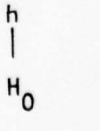
$S \rightarrow \begin{matrix}\$\\ |\\ C_i\end{matrix}$ , $\begin{matrix}\$\\ |\\ D_i\end{matrix}$ , $i = 1, \ldots, 3$

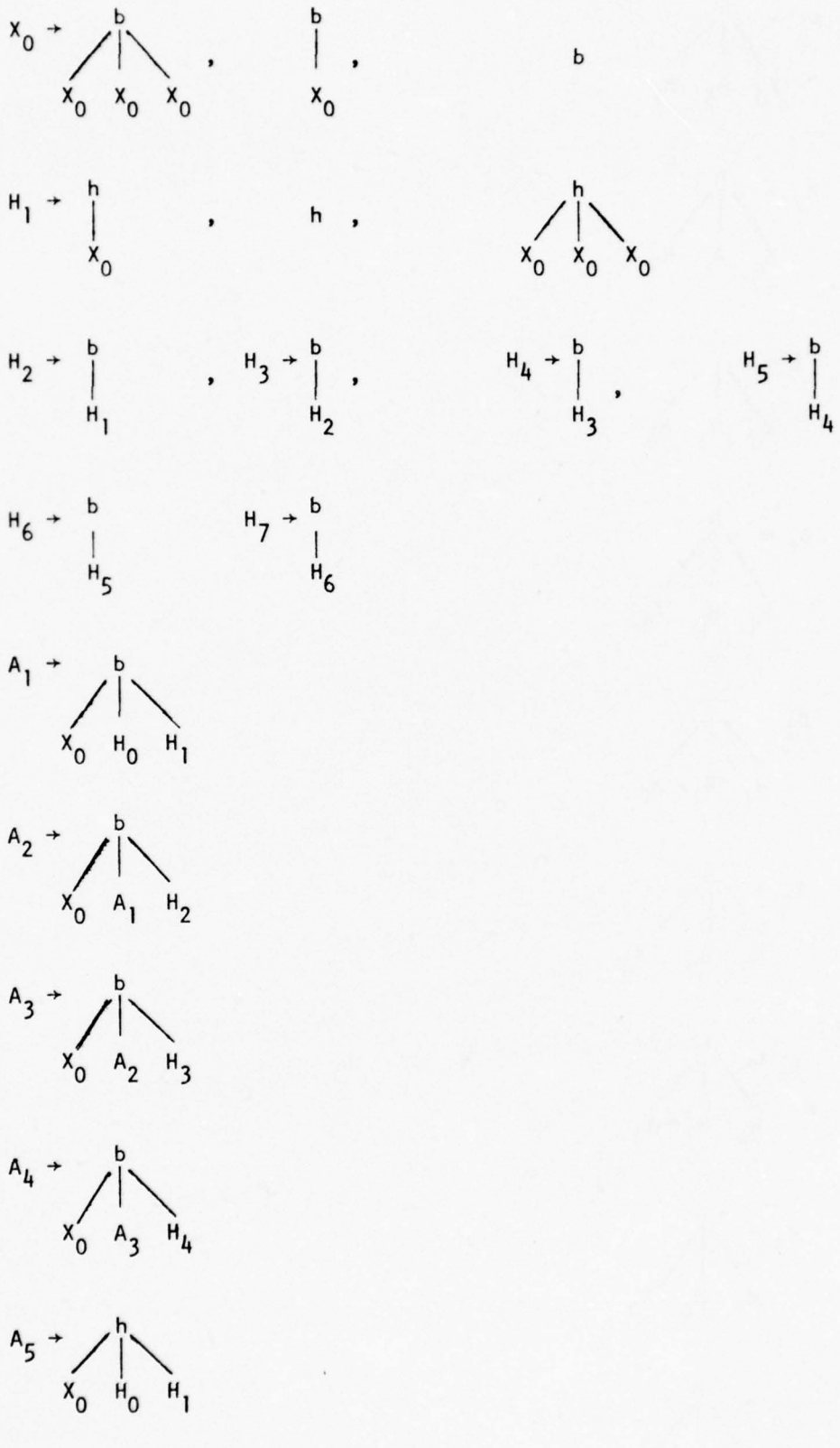
$S \rightarrow \begin{matrix}\$\\ |\\ E_i\end{matrix}$ , $\begin{matrix}\$\\ |\\ F_i\end{matrix}$ , $i = 1, \ldots, 7$

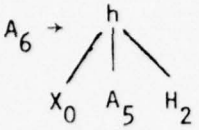
$S \rightarrow \begin{matrix}\$\\ |\\ M_i\end{matrix}$ , $i = 1, \ldots, 4$

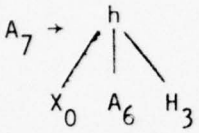
$S \rightarrow \begin{matrix}\$\\ |\\ H_0\end{matrix}$ , $\begin{matrix}\$\\ |\\ X_0\end{matrix}$

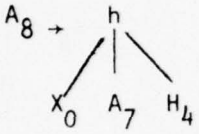
$H_0 \rightarrow$  ,
 ,
 ,


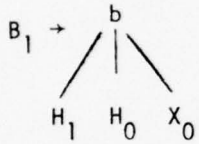
 ,
 ,
 ,
h

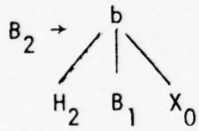

X0 → b
X0 X0 X0
,
b
X0
,
b
H1 → h
X0
,
h
,
h
X0 X0 X0
H2 → b
H1
,
H3 → b
H2
,
H4 → b
H3
,
H5 → b
H4
H6 → b
H5
H7 → b
H6
A1 → b
X0 H0 H1
A2 → b
X0 A1 H2
A3 → b
X0 A2 H3
A4 → b
X0 A3 H4
A5 → h
X0 H0 H1

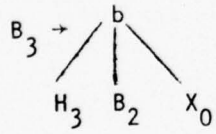

A6 → h
X0 A5 H2

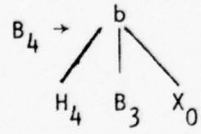

A7 → h
X0 A6 H3

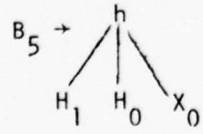

A8 → h
X0 A7 H4

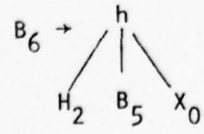

B1 → b
H1 H0 X0

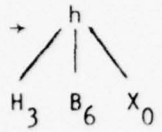

B2 → b
H2 B1 X0

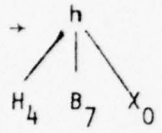

B3 → b
H3 B2 X0

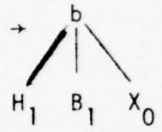
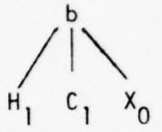

B4 → b
H4 B3 X0

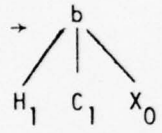
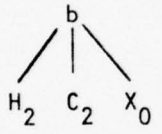

B5 → h
H1 H0 X0

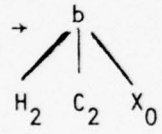
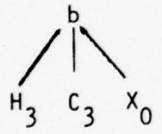

B6 → h
H2 B5 X0

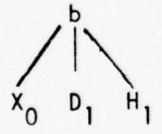
$B_7 \rightarrow$ 

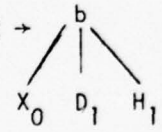
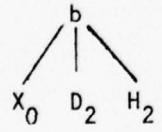

$B_8 \rightarrow$ 

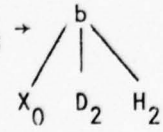
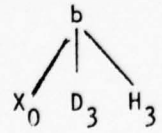

$C_1 \rightarrow$  ,  , b

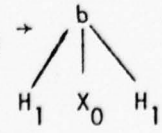

$C_2 \rightarrow$  ,  , b

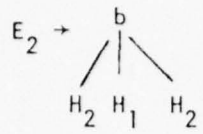

$C_3 \rightarrow$  , 

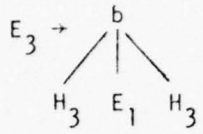

$D_1 \rightarrow$  ,  , b

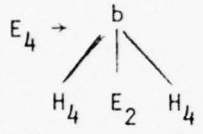

$D_2 \rightarrow$  ,  , b

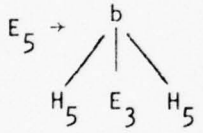

$D_3 \rightarrow$  , 

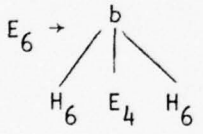

$E_1 \rightarrow$ 

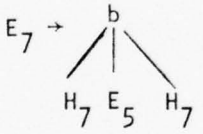

E2 →
b
H2 H1 H2

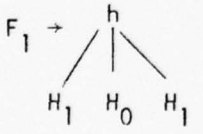

E3 →
b
H3 E1 H3

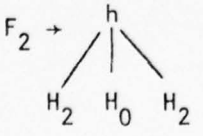

E4 →
b
H4 E2 H4

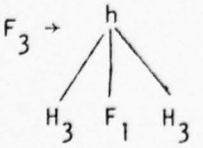

E5 →
b
H5 E3 H5

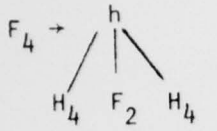

E6 →
b
H6 E4 H6

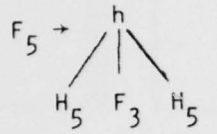

E7 →
b
H7 E5 H7

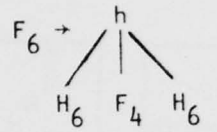

F1 →
h
H1 H0 H1

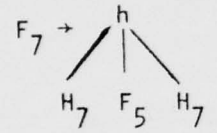

F2 →
h
H2 H0 H2

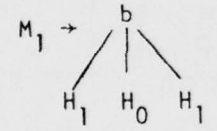

F3 →
h
H3 F1 H3

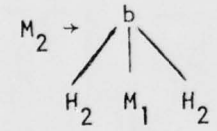

F4 →
h
H4 F2 H4

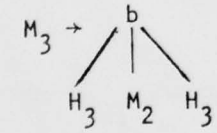

F5 →
h
H5 F3 H5

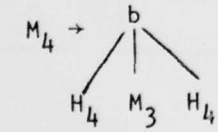

F6 →
h
H6 F4 H6

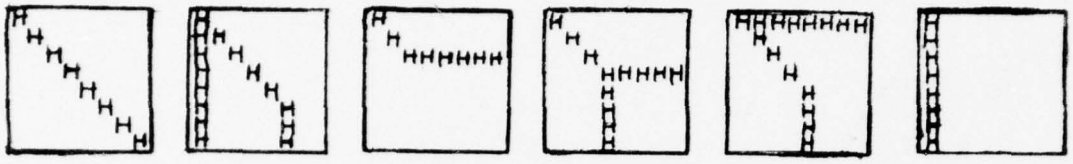

F7 →
h
H7 F5 H7


M1 →
b
H1 H0 H1

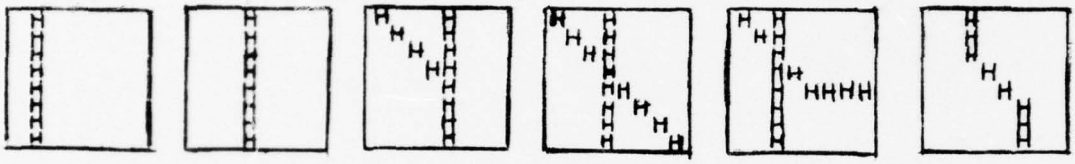

M2 →
b
H2 M1 H2


M3 →
b
H3 M2 H3

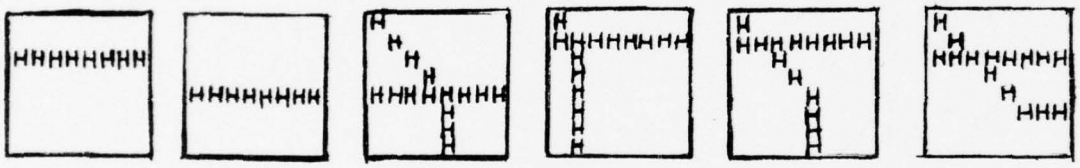

M4 →
b
H4 M3 H4



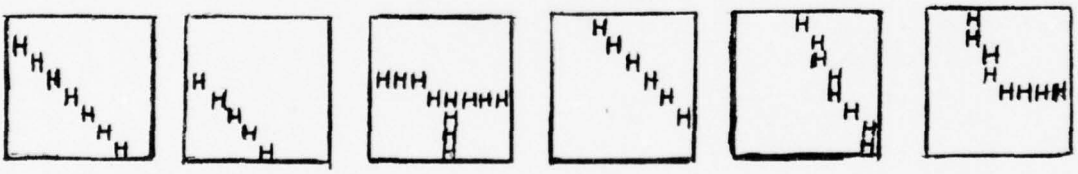
(a) Group $H_0$



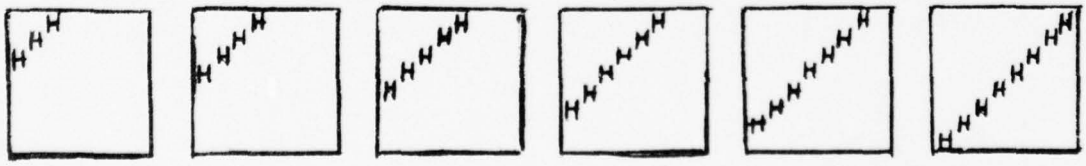
(b) Group A



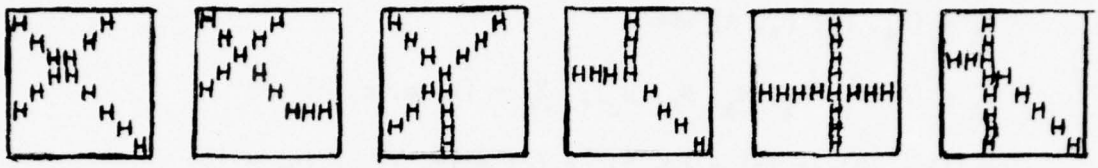
(c) Group B



(d) Group C, D



(e) Group E



(f) Group F, M

Fig. A-1. Typical Patterns Generated from $G_H$

## Appendix B

### B.1. Character Grammar

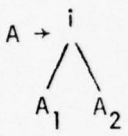
(a) Character A

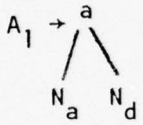
$G_A = (V_A, r_A, P_A, A)$ where

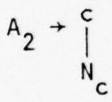
$V_A = \{A, A_1, A_2, N_a, N_d, N_c\}$, $\Sigma_A = \{i, a, c, d\}$

$r_A(i) = \{2\}$, $r_A(a) = \{0, 2\}$, $r_A(c) = \{0, 1\}$, $r_A(d) = \{0\}$

$P_A$:




$N_a \rightarrow a$, $N_d \rightarrow d$, $N_c \rightarrow c$

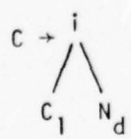
(b) Character C

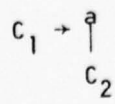
$G_C = (V_C, r_C, P_C, C)$ where

$V_C = \{C, C_1, C_2, C_3, N_d, N_f\}$, $\Sigma_C = \{a, b, d, f\}$

$r_C(i) = \{2\}$, $r_C(a) = \{1\}$, $r_C(b) = \{1\}$, $r_C(d) = \{0, 1\}$, $r_C(f) = \{0\}$

$P_C$:



$C_2 \rightarrow \begin{matrix} b \\ | \\ C_3 \end{matrix}$

$C_3 \rightarrow \begin{matrix} d \\ | \\ N_f \end{matrix}$

$N_d \rightarrow d, \quad N_f \rightarrow f$

(c) Character D

$G_D = (V_D, r_D, P_D, D)$

$V_D = \{D, D_1, D_2, D_3, D_4, N_b, N_d\}, \Sigma_D = \{i, b, d\}$

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

$P_D$:

$D \rightarrow \begin{matrix} i \\ / \quad \backslash \\ D_1 \quad D_2 \end{matrix}$

$D_1 \rightarrow \begin{matrix} b \\ | \\ D_3 \end{matrix}$

$D_3 \rightarrow \begin{matrix} b \\ | \\ N_d \end{matrix}$

$D_2 \rightarrow \begin{matrix} d \\ | \\ D_4 \end{matrix}$

$D_4 \rightarrow \begin{matrix} b \\ | \\ N_b \end{matrix}$

$N_d \rightarrow d, \quad N_b \rightarrow b$

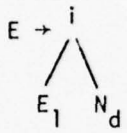
(d) Character E

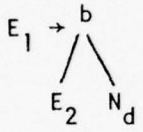
$G_E = (V_E, r_E, P_E, E)$

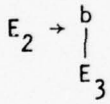
$V_E = \{E, E_1, E_2, E_3, N_d\}$, $\Sigma_E = \{i, b, d\}$

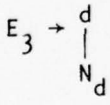
$r_E(i) = \{2\}$, $r_E(b) = \{1, 2\}$, $r_E(d) = \{0, 1\}$

$P_E$:

$E \rightarrow$ 


$E_1 \rightarrow$ 


$E_2 \rightarrow$ 


$E_3 \rightarrow$ 


$N_d \rightarrow d$

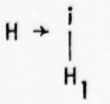
(e) Character H

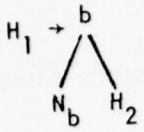
$G_H = (V_H, r_H, P_H, H)$

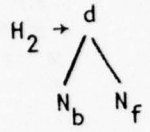
$V_H = \{H, H_1, H_2, N_b, N_f\}$, $\Sigma_H = \{i, b, d, f\}$

$r_H(i) = \{2\}$, $r_H(b) = \{0, 2\}$, $r_H(d) = \{2\}$, $r_H(f) = \{0\}$

$P_H$:

$H \rightarrow$ 



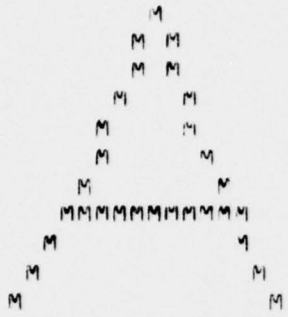




$N_b \rightarrow b$

$N_f \rightarrow f$

## Appendix B.2. Classification Results of 26 Test Patterns

INPUT CHARACTER



TREE REPRESENTATION

| TREE DOMAIN | PRIMITIVE | RANK |
|---|---|---|
| 1 | 1 | 2 |
| 11 | * | 1 |
| 12 | C | 1 |
| 111 | A | 2 |
| 1111 | * | 1 |
| 1112 | D | 0 |
| 11111 | A | 0 |
| 121 | C | 0 |

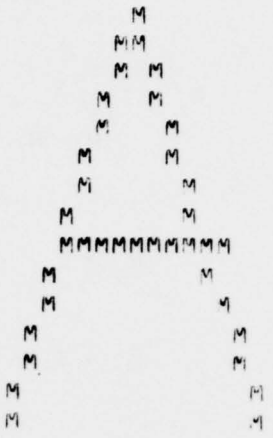
TIME USED FOR LINKING A TREE
.189 SEC

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 | 1 | 2 |
| 11 | * | 1 |
| 12 | C | 1 |
| 111 | B | 2 |
| 1111 | * | 1 |
| 1112 | D | 0 |
| 11111 | B | 0 |
| 121 | C | 0 |

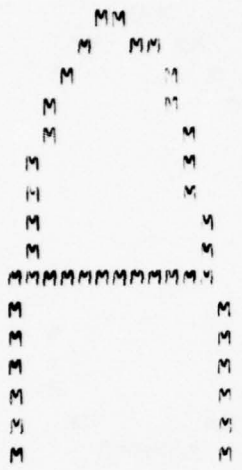
TIME USED FOR LINKING A TREE
.188 SEC

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 |

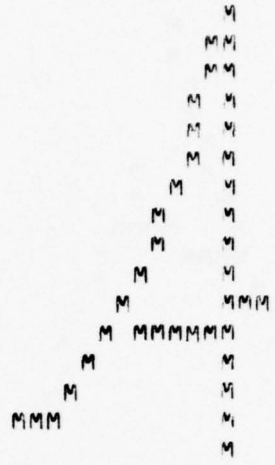
TIME USED FOR LINKING A TREE
.188 SEC

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 | 1 | 1 |
| 11 | A | 2 |
| 111 | * | 1 |
| 112 | D | 1 |
| 1111 | A | 0 |
| 1121 | C | 0 |

TIME USED FOR LINKING A TREE
.183 SEC

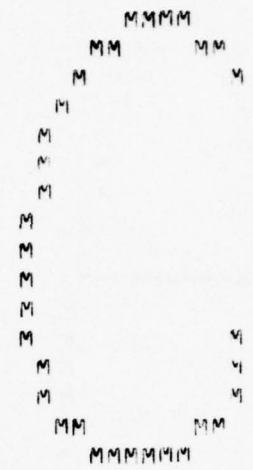
INPUT CHARACTER IS A

DISTANCE FORM NORMAL A

IS 3

TIME USED FOR PARSING
2.669 SEC

INPUT CHARACTER



TREE REPRESENTATION

| TREE DOMAIN | PRIMITIVE | RANK |
|---|---|---|
| 1 | 1 | 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 |

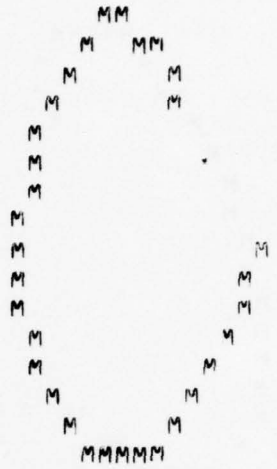
TIME USED FOR LINKING A TREE
.186 SEC

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 | D | 0 |
| 111 | A | 1 |
| 1111 | B | 1 |
| 11111 | J | 0 |

TIME USED FOR LINKING A TREE
.183 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 |

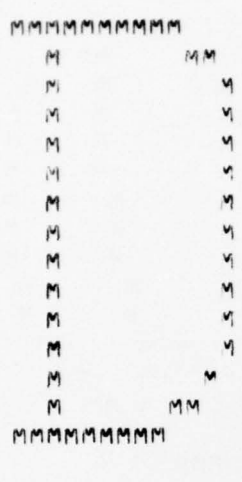
TIME USED FOR LINKING A TREE
.187 SEC

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 | D | 1 |
| 111 | B | 1 |
| 1111 | B | 1 |
| 121 | B | 1 |
| 11111 | D | 0 |
| 1211 | B | 0 |

TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
IS 0

TIME USED FOR PARSING
3.138 SEC

INPUT CHARACTER



TREE REPRESENTATION

| TREE DOMAIN | PRIMITIVE | RANK |
|---|---|---|
| 1 | I | 2 |
| 11 | * | 1 |
| 12 | D | 1 |
| 111 | B | 1 |
| 1111 | B | 1 |
| 121 | B | 1 |
| 11111 | D | 0 |
| 1211 | B | 0 |

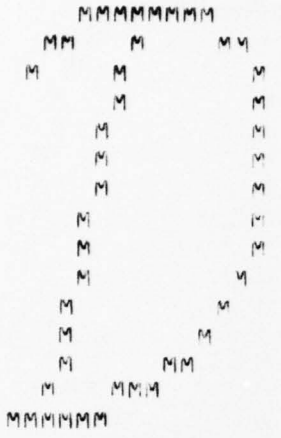
TIME USED FOR LINKING A TREE
.190 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
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 | H | 0 |
| 1121 | B | 1 |
| 121 | B | 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 | H | 0 |

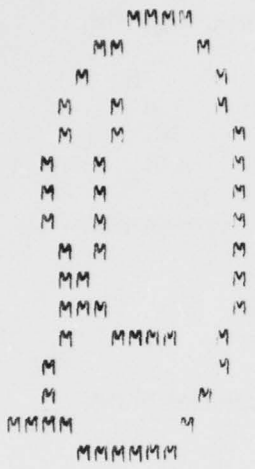
TIME USED FOR LINKING A TREE
.194 SEC

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 |

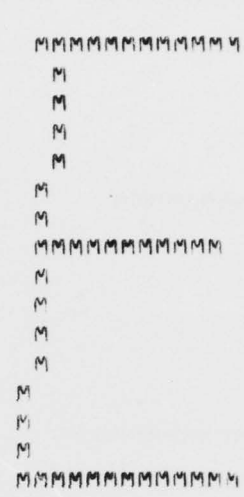
TIME USED FOR LINKING A TREE
.197 SEC

INPUT CHARACTER IS D

DISTANCE FORM NORMAL D
IS 4

TIME USED FOR PARSING
6.668 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 |

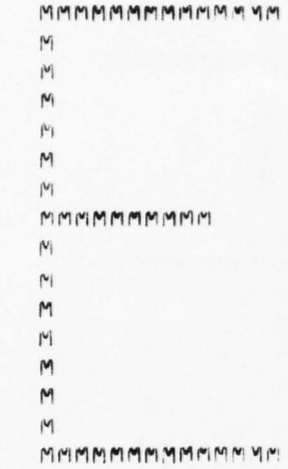
TIME USED FOR LINKING A TREE
.193 SEC

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 | D | 1 |
| 111 | B | 2 |
| 1111 | * | 1 |
| 1112 | D | 0 |
| 11111 | B | 1 |
| 121 | D | 0 |
| 111111 | D | 1 |
| 1111111 | D | 0 |

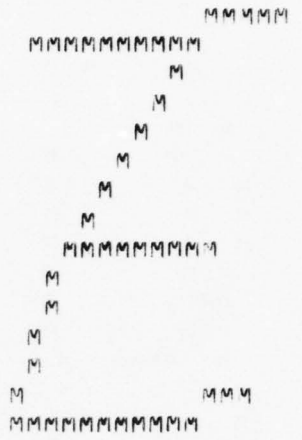
TIME USED FOR LINKING A TREE
.192 SEC

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 | D | 0 |
| 111 | * | 1 |
| 112 | H | 0 |
| 1111 | A | 2 |
| 11111 | * | 1 |
| 11112 | D | 0 |
| 111111 | B | 1 |
| 1111111 | D | 1 |
| 11111111 | D | 0 |

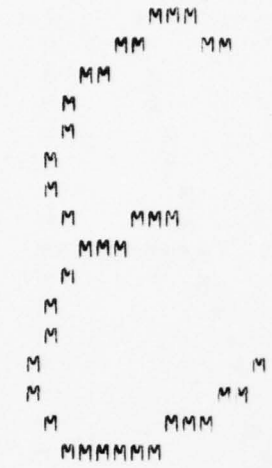
TIME USED FOR LINKING A TREE
.195 SEC

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 |

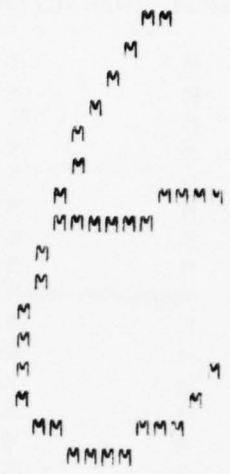
TIME USED FOR LINKING A TREE
.191 SEC

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 | D | 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 | D | 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 |

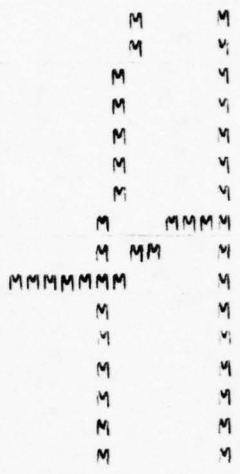
TIME USED FOR LINKING A TREE
.187 SEC

INPUT CHARACTER IS H

DISTANCE FORM NORMAL H
IS 1

TIME USED FOR PARSING
3.755 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 | 1 | 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 H

DISTANCE FORM NORMAL H
IS 2

TIME USED FOR PARSING
3.089 SEC

INPUT CHARACTER



TREE REPRESENTATION

| TREE DOMAIN | PRIMITIVE | RANK |
|---|---|---|
| 1 | 1 | 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

References


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

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

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

4. Bhargava, B. K., "Tree Systems for Syntactic Pattern Recognition," Ph.D. Thesis, Purdue University, May, 1974.

5. Boorman, S. A. and Oliver, D. C., "Metrics on Spaces of Finite Trees," J. of Math. Psychology, 10, 26-59, 1973.

6. Brainerd, W. F., "Tree Generating Regular Systems," Inf. and Control, 14, 217-231, 1969.

7. Brayer, J. M. and Fu, K. S., "Web Grammars and Their Application to Pattern Recognition," Purdue University, TR-EE 75-1, December, 1975.

8. Doner, J., "Tree Acceptors and Some of Their Applications," J. of Comut. and Sys. Sci. 4, pp. 406-451, 1970.

9. Ellis, C. A., "Probabilistic Tree Automata," Information and Control, Vol. 19, pp. 401-416, 1971.

10. Fu, K. S., "On Syntactic Pattern Recognition and Stochastic Languages," Frontiers of Pattern Recognition, ed. S. Watanabe, Academic Press, 1972.

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

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

13. Fu, K. S. and Booth, T. L., "Grammatical Inference: Introduction and Survey - Part I and Part II," IEEE Trans. on Systems, Man, & Cybernetics, Vol. SMC-5 1975.

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

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

16. Hopcroft, J. E. and Ullman, J. D., "Formal Languages and Their Relation to Automata," Addison-Wesley, 1969.

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

18. Lu, S. Y. and Fu, K. S., "Efficient Error-Correcting Syntax Analysis for Recognition of Noisy Patterns," Purdue University, TR-EE 76-9, March, 1976.

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

20. Rounds, W. C., "Mappings and Grammars on Trees," Math. System Theory, Vol. 4, No. 3, 257-286, 1970.

21. Tanaka, E. and Fu, K. S., "Error-Correcting Parser for Formal Language," Purdue University, TR-EE 76-7, March, 1976.

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

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

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

25. Thatcher, J. W., "Characterizing Derivation Trees of Context-Free Grammars Through a Generation of Finite Automata Theory," J. of Comput. and Sys. Sci., 1, 1967, 317-322.

26. Thatcher, J. W., "Tree Automata: An Informal Survey," Currents in the Theory of Computing, ed. by A. V. Aho, Prentice Hall, 1973.

27. Todd, W. J. and Baumgardner, M. F., "Land Use Classification of Marion County, Indiana, Data," LARS Information Note 101673, LARS, Purdue University, 1973.

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

29. Evans, T. G., "Grammatical Inference Techniques in Pattern Analysis," Information Systems, ed. Tou, J. T.

30. Feng, H. Y. and Pavlidis, T., "Analysis of Complex Shapes In Terms of Simpler Ones: Feature Generation for Syntactic Pattern Recognition," Princeton University, TR No. 149, April, 1974.

31. Feng, H. Y. and Pavlidis, T., "Decomposition of Polygons into Simpler Components: Feature Extraction for Syntactic Pattern Recognition," IEEE Trans. on Computers, Vol. C-24, June, 1975.

32. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations," IEEE Trans. Electron. Comput., EC-10, pp. 260-268, 1961.

33. Gonzalez, R. C., Edwards, J. J., and Thomason, M. G., "An Algorithm for the Inference of Tree Grammars," Int. J. of Comp. and Inf. Sci., Vol. 5, No. 2, June, 1976.

34. Munson, J. H., "Experiments in the Recognition of Hand-Printed Text - Part I, Character Recognition," AFIPS Conference Proceedings, Vol. 33, Part 2, pp. 1125-1138, 1968.

35. Narasimhan, R., "Picture Languages," in Picture Language Machines, ed. by S. Kaneff, Academic Press, 1970.

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

37. Shaw, A. C., "A Formal Picture Description Scheme as a Basis for Picture Processing Systems," Inf. and Cont. 14, pp. 9-52, 1969.

38. Stallings, W., "Recognition of Printed Chinese Characters by Automatic Pattern Analysis," Computer Graphics and Image Processing, Vol. 1, No. 1, pp. 47-65, 1972.

39. Williams, K. L., "A Multidimensional Approach to Syntactic Pattern Recognition," Pattern Recognition, Vol. 7, No. 3, September, 1975.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER: AFOSR - TR - 76 - 1207 | 2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle): ERROR-CORRECTING SYNTAX ANALYSIS FOR TREE LANGUAGES | 5. TYPE OF REPORT & PERIOD COVERED: Interim Rept. |
| | 6. PERFORMING ORG. REPORT NUMBER: TR-EE 76-24 |
| 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, Department of Electrical Engineering, West Lafayette, Indiana 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, Washington, DC 20332 | 12. REPORT DATE: July 1976 |
| | 13. NUMBER OF PAGES: 89 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): TR-EE 76-24 | 15. SECURITY CLASS. (of this report): UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: 92 p. |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

AF-AFOSR-2661-74

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

2304 A2

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)

An error-correcting version of tree automaton is studied. Assume the syntax errors on trees to be of five types: substitution, stretch, split, branch, and deletion. The error-correction problem is to determine the distance between two trees as measured by the minimum cost sequence of error transformations needed to transform one to the other. Two types of error-correcting tree automaton (ECTA) are proposed: structure-preserved ECTA and generalized ECTA. The structure-preserved ECTA takes substitution errors into consideration only. A LANDSAT data interpretation problem is used as an example to illustrate the

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

292000

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20 Abstract

process of structure-preserved ECTA. The restriction on tree structure is removed in formulating the generalized ECTA. All five types of syntax errors are considered. Any erroneous tree, in this case, is able to find its most similar (in terms of minimum distance) correction in the sense of both node-labeling and tree structure.