AN EXPERIMENTAL STUDY OF THE SYNTACTIC ANALYSIS AND RECOGNITION OF HAND-WRITTEN NUMERALS

Yu Mong

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 1982

ABSTRACT

AN EXPERIMENTAL STUDY OF THE SYNTACTIC ANALYSIS AND RECOGNITION OF THE HAND-WRITTEN NUMERALS

Yu Mong

The work reported in this thesis is to design and implement a system to recognize hand-written numerals. An algorithm has been developed to extract topological features (such as loops, cavities of different orientations and straight line components in a character image) from the graph representation of a character. The performance of these features used as pattern primitives to recognize a character is tested and their use in the reconstruction of the character image is highlighted. Furthermore, a method of combining syntactic and semantic (such as the position of the primitives in the character frame, depth of a cavity and opening of a cavity, etc.) information for the classification is proposed and tested.

A data base consisting of 6000 hand-written numerals by 30 different authors were tested. An overall recognition rate of 99.15% was obtained.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my supervisor, Professor C.Y. Suen, for his guidance, invaluable suggestions, criticisms and encouragement during the course of this research work. I wish to thank my friend, Pervez Ahmad for many fruitful discussions and encouragement in the preparation of this thesis. I would like to acknowledge my friends Selina W. Wong and Yuen Y. Tam for their assistance, and the Computer Science Department for giving me a lot of convenience during the preparation of the manuscript. Thanks are also due to the Natural Sciences and Engineering Research Council of Canada for financial support of this project and Concordia University for the award of a graduate teaching fellowship.

I would like to give special thanks to my family for their understanding and support for the past years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Automatic pattern recognition has been a very challenging subject for the past two decades. The increasing attention to this research field is due to the demand of large volumes of information processing. Nevertheless, the rapid advances in electronic technology have enabled the use of automatic recognition techniques in various disciplines such as computer science, information theory, medicine, and engineering, etc. In particular, "optical character recognition" (OCR) has long established itself as a practical example of automatic pattern recognition.

Numbers are the most commonly used characters. Since the set of Arabic numerals is almost regarded as universal, a lot of kinds of information are coded by numbers. In our daily life, commercial products are coded with serial numbers, library books are assigned call numbers etc. And even we, human beings, are often identified by some numbers such as social insurance number, student identification number, etc. Obviously, the use of numerals facilitates the processing of information by digital computers in today's computerized world. The numerical represention is compact, and can be processed efficiently in sorting and other manipulation. The automatic reading of machine-printed and handwritten numerals is now no longer just an interesting research subject but also a practical and urgent problem. The reading of numerals by computer has already been applied successfully to read account sheets, bank cheques [31] and postal codes [7,11,12,18]. The algorithms proposed in this thesis on character recognition, though not restricted to a particular character set, were tested only with numerals.

1.2 Basic Structure of a Character Recognition System

A typical character recognition system has three main components, namely, (1) sensor, (2) characteristics extractor, and (3) classifier. Sensing involves the digitization of character image by an optical scanner and also binarization if necessary. For the extraction of characteristics, it can be divided into two stages, (i) preprocessing and (ii) feature extraction. Preprocessing is the enhancement and filtering of the digitized image preparing it for the second stage. In the feature extraction stage, numerical measurements or structural properties representing the characteristics of the input images are extracted and called features. The selection of these features depends very much on their discriminatory qualities. The third component of the system involves the

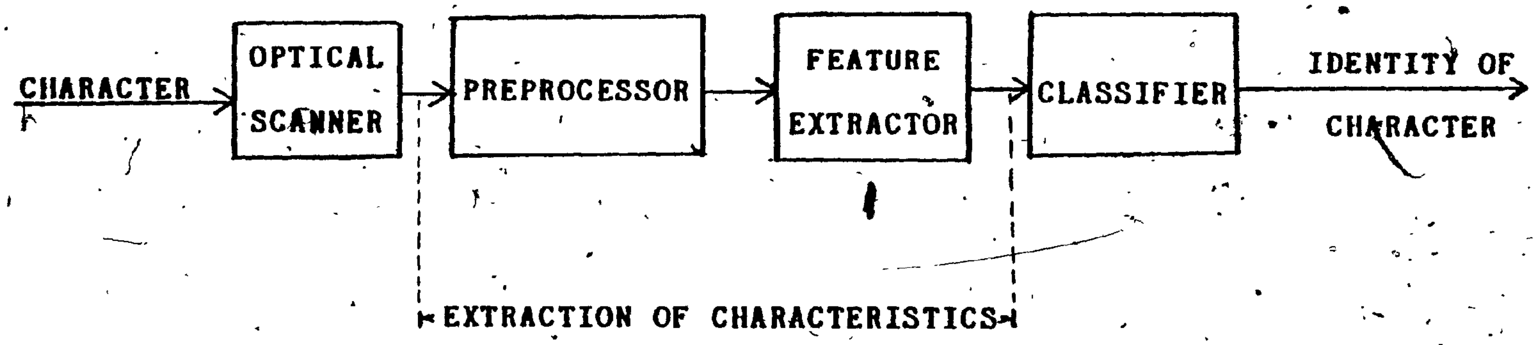


Figure 1.1. Block Diagram of a Typical Character Recognition System.

classification and identification of the character images into their respective classes. Fig.1.1 shows the block diagram of a typical character recognition system.

In each of these components, there are problems to be solved to achieve good recognition performance. To imitate the perception of human beings, an optical scanner should have a reasonably high resolution like that of the human eyes. Fortunately, advances in electronics technology generally have solved this problem at a reasonable cost. In the design of the feature extractor, the selection of features and the ease of extracting them are problems needed to be considered. There is no general formula for feature selection. Different kinds of features have been used such as moments [2,38], pixel counts in different regions [34], crossing counts [34], Fourier descriptors [27] and geometrical structures [4,37]. After the selection of features, a consistent classifier has to be designed to give an optimal decision. The components actually are not independent from each other. Consequently, the performance of a certain stage is always affected by those preceeding it.

1.3 Syntactic Recognition System

Extensive research work on character recognition has been done by numerous researchers. Tremendous efforts have been made to improve the sensing devices, to explore effective preprocessing techniques [3,5,16,24,39] and to evaluate effective feature sets and the design of classifiers [36]. There are three principal methods in pattern analysis, viz. heuristic, mathematical, and syntactic. The heuristic method makes use of human experience and intuition to simulate human perception [36]. The mathematical method makes use of mathematical framework which may be a parametric model based on Bayes' rule or a non-parametric deterministic method [36]. This method is also called a decision-theoretic method. The syntactic method, which has received considerable attention in recent years, is used in this work. It is discussed below.

The techniques of the syntactic approach consist of the decomposition of a pattern into subpatterns or more basic components called primitives according to the structure of the pattern. This approach makes use of the concept of the theory of formal languages to describe the structure of the pattern. With the rules of pattern decomposition as the grammar description of structural relations in terms of primitives, a language called 'pattern description language' can be built [23,28]. The recognition process is then analogous to the grammatical (syntax) analysis of a natural language.

A syntactic approach in character recognition is illustrated by the block diagram shown in Fig.1.2. The system consists of three parts, namely (i) preprocessing

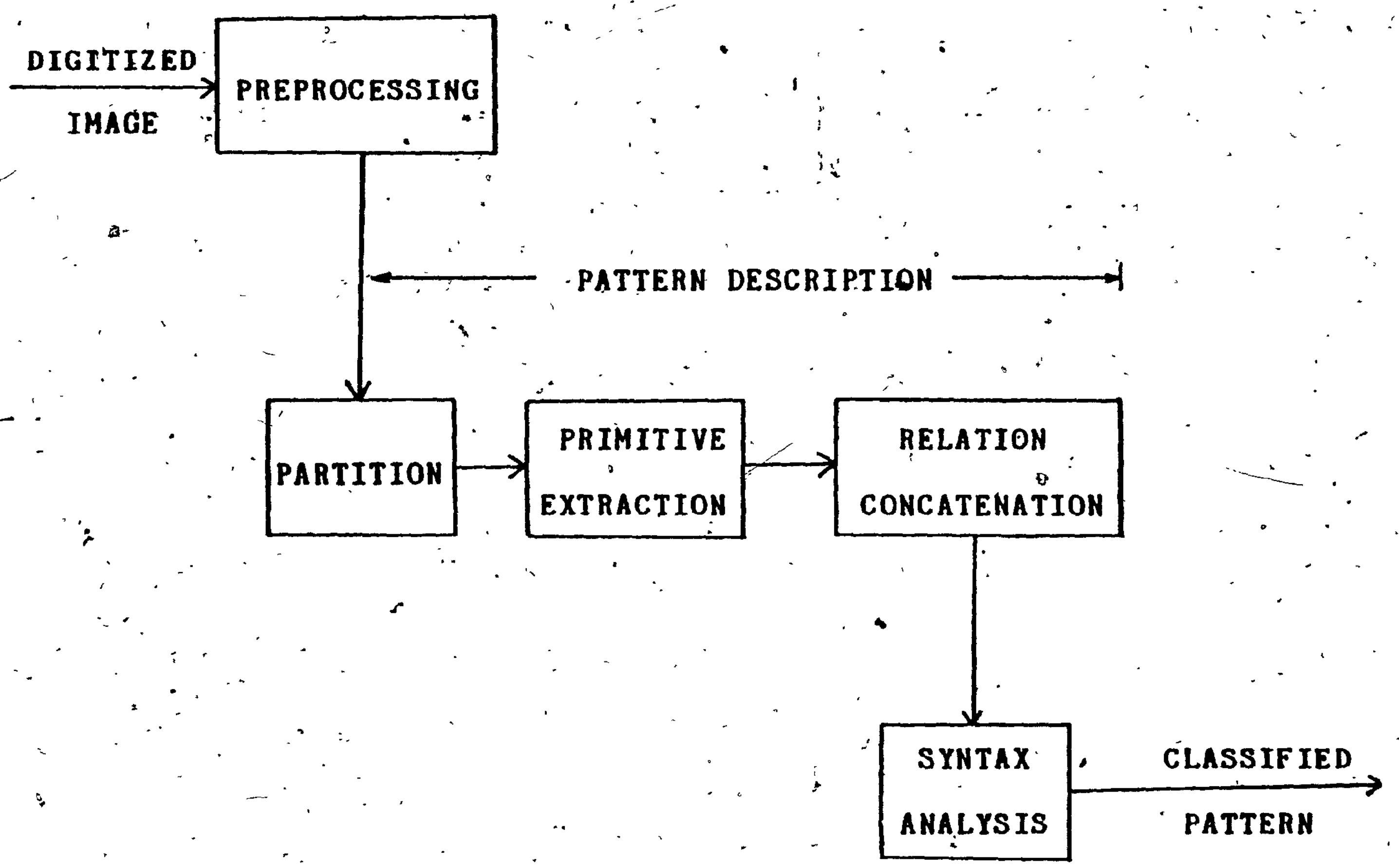


Figure 1.2. Block Diagram of a Syntactic Recognition System.

(ii) pattern description and (iii) syntax analysis. Preprocessing is a basic step which enables a pattern to be described appropriately. Pattern description normally includes the partition of a pattern into subpatterns, primitive extraction and concatenation of primitives and subpatterns. Finally, it is the parsing which identifies the pattern.

1.4 The Proposed Character Recognition System

The principal aim of this work is to define a character pattern by a structural representation based on its geometrical structures and identify it by applying syntactic analysis. A block diagram of the proposed system is shown in Fig.1.3.

The basic structure of the system can be divided into three parts : a) preprocessing - reduction to graph representation, b) pattern description - graph encoding and primitive extraction, and, c) classification - syntactic-semantic analysis. A character is first thinned into a skeleton and represented by line codes. A graph representation of the thinned character is constructed and encoded by quantizing the graph edges into eight directions. Through a compact encoded form, topological features are extracted and used as primitives. Their characteristic measurements are evaluated. A concatenation of the feature primitives into a string form thus describes the structure

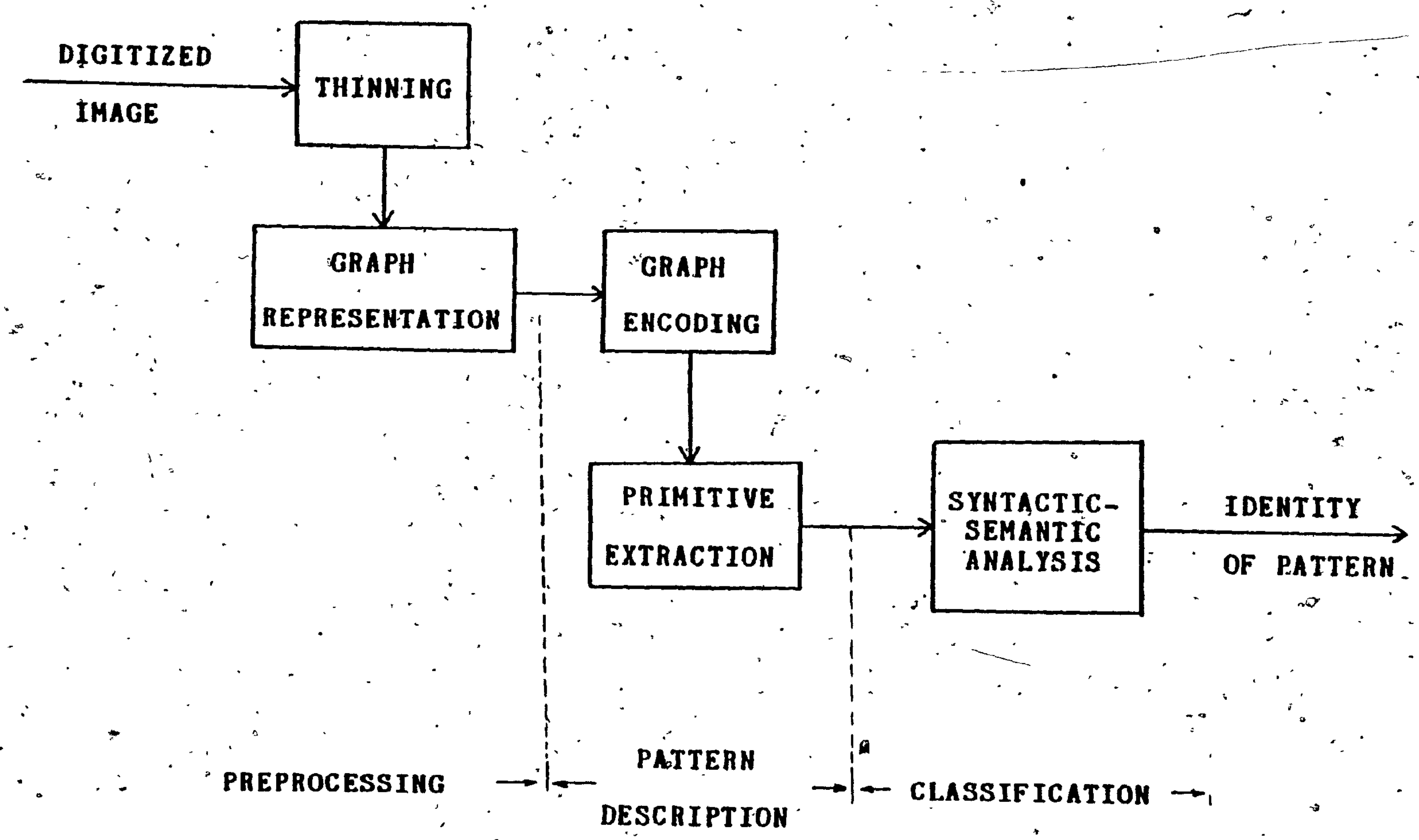


Figure 1:3. Basic Structure of the Proposed System.

of the character. Finally, the string) is parsed and analyzed semantically to complete the recognition process.

1.5 Summary of the Remaining Chapters

In this work, I have developed the algorithm for the reduction of a thinned character into its graph representation, designed the convention to encode the graph and established the algorithm for the extraction of primitives, implemented a character recognition system and tested it with data. The different pieces of research work reported in this thesis are discussed in the remaining chapters as given below.

In chapter 2, the preprocessing techniques are discussed. The transformation of a digitized image into thinned line codes for further processing is described. From these line codes, an algorithm which generates a graph representation of a character is proposed.

In chapter 3, an encoding method is established to generate the code string. An algorithm to extract topological features as pattern primitives from the code string to form a feature string is proposed. Together with the characteristic measurements, a feature string is ready for syntax and semantic analysis.

In chapter 4, the fundamental principle of the syntactic approach is summarized. The basic definitions in formal language theory and the classification model are covered in general. The syntactic and semantic descriptive schema for the experimental data are also discussed.

In chapter 5, experimental results illustrating the performance of the system are presented.

In chapter 6, conclusions on the overall structure of the system and its performance and suggestions of improvement are discussed.

CHAPTER TWO

PREPROCESSING AND GRAPH REPRESENTATION

2.1 Introduction

In a recognition system, the preprocessing of the digitized image is an important process. During optical scanning, noise may be introduced to distort an image, hence preprocessing is necessary in order to obtain a clean and clear image for further recognition processes. Preprocessing, in general, involves the smoothing and enhancement of an image. Moreover, some transformations of the image will enable efficient feature analysis or better expression of the character, e.g. skeletonization to be described in section 2.3. In the proposed system, the skeletonization of the binary image is performed with the purpose of producing a graph. A graph representation is a desirable form to represent a character pattern for syntactic analysis since this representation can reduce the complexity of the grammars for the structural description of the complete character.

In this chapter, smoothing techniques are briefly described. The thinning algorithm applied to the digitized image to produce a skeleton is also included. The conversion of the skeleton into a chain-code curve to facilitate the construction of a graph is described. The second part of the chapter presents the proposed algorithm to depict the character in a graph form.

2.2 Techniques on Smoothing

Depending upon the grey level, there are many techniques of smoothing and enhancing a digitized image [14]. The method employed usually depends on the quality and grey level of the data. For very noisy or severely distorted images, more than one technique has to be used while in some cases, only simple techniques are required. Digitized images are enhanced through the filling of small holes and the deletion of isolated pixels.

One technique suggested by Unger [39] considers the 3x3 window of each point as shown in Fig.2.1. Given a binary image, each pixel p is considered sequentially. To fill in an isolated hole or a small notch in a straight edge, its four adjacent neighbours (b,d,e,g) are examined and if at least three of them are black points, then it is filled. On the other hand, Doyle [5] suggested a way to fill a hole by considering any one of its four neighbour pairs horizontally (d,e), vertically (b,g) and diagonally ((a,h) or (c,f)); if at least one pair is black, then it is filled. To eliminate isolated points and small bumps, their suggestion was to look at the 3x3 window in terms of 4 sets of corner points ({a,b,d},{b,c,e},{e,g,h} and {d,f,g}). At least one black point from each of the opposite corner pairs should be there to allow a black point to be retained. The rules can be summarized by the following boolean functions.

Consider a pixel p in the digitized image shown in Fig.2.1.

Filling operation

Unger's method : $F_{fill} = p + b.g.(d + e) + d.e.(b + g)$

Doyle's method : $F_{fill} = p + a.h + b.g + c.f + d.e$

Retaining operation

$F_{retain} = p.[(a+b+d).(e+g+h) + (b+c+e).(d+f+g)]$

a	b	c
d	p	e
f	g	h

a pixel is true if it is black;

a pixel is false if it is white.

Figure 2.1. 3x3 Window of Points Involved.

The complexity of smoothing operations required depends very much on the type of pattern encountered. In the proposed system, a simple operation is used to eliminate the isolated black points and holes. A white point is filled if all its eight neighbours in the 3x3 window are black while a black point is eliminated only if all its eight neighbours are white.

2.3 Skeletonization

Reducing a digitized image to a thin line is recommended for line-like structure such as characters. The necessity of obtaining the skeleton of a pattern is usually determined by the method of feature extraction. In this system, a graph form is chosen to represent a character and thus the thinning operation gives advantages in locating graph vertices and end points. The skeleton of a character should not only define the original shape, but also retain the connectivity of each image component. Quite a few algorithms have been developed to achieve this goal [3,24,16].

In this system, a package by H. Ma and J. Yudin [22] was used in the skeletonization of the experimental data. The approach involves the repeated boundary tracing of the character image and removal of those boundary points which do not contribute to the skeleton. The whole idea is to shear off an image layer after layer until each segment is one pixel wide. To maintain connectivity of the image points, the criteria to retain an edge point are established by examining the image with a 3x3 window. The list of criteria to retain a point can be summarized as follows :

1) removal of the point splits a connected component in two;

2) removal of the point introduces a loop;

3) removal of the point causes shrinkage of the length of the segment;

4) the point has three immediate neighbours which are boundary points;

5) the point is the tip of a thin line.

2.4 Condition Codes for the Thinned Character

Before the transformation of the skeleton into a graph form, each point in the skeleton is given a condition code in order to locate the end points of strokes and branch points (junctions). A point in the skeleton is regarded as an end point if only one of its eight neighbours is also a point in the skeleton while a branch point has more than two points in the skeleton as its neighbours. The following coding scheme is adapted from Ogawa and Taniguchi [25]. Given a thinned character represented by an NxM boolean matrix, if a point is in the skeleton, it is true and else false. Consider a point in the skeleton, $P_{i,j}$ with its 3x3 window, as shown in Fig. 2.2.

$P_{i-1,j-1}$	$P_{i-1,j}$	$P_{i-1,j+1}$
$P_{i,j-1}$	$P_{i,j}$	$P_{i,j+1}$
$P_{i+1,j-1}$	$P_{i+1,j}$	$P_{i+1,j+1}$

Figure 2.2. A Skeleton Point $P_{i,j}$ with its 8 Neighbours.

Denote the condition code of $P_{i,j}$ by $C_{i,j}$ which is assigned a value in the set $\{-10,-8,-7,..,-1,0,1,2,..,9\}$. See Fig. 2.3.

The rules are given for each $P_{i,j}$ as follows :

Rule 1 : If any one and only one of the following conditions is satisfied, then

$$0 \leq C_{i,j} \leq 9 .$$

Conditions :

(1) if $\{P_{i-1,j-1}$ and $P_{i-1,j+1}\}$ or $\{P_{i+1,j-1}$ and $P_{i+1,j+1}\}$, then $C_{i,j} = 0$;

(2) if $P_{i,j-1}$ and $P_{i,j+1}$, then $C_{i,j} = 1$;

(3) if $P_{i-1,j+1}$ and $P_{i+1,j-1}$, then $C_{i,j} = 2$;

(4) if $P_{i-1,j}$ and $P_{i+1,j}$, then $C_{i,j} = 3$;

(5) if $P_{i-1,j-1}$ and $P_{i+1,j+1}$, then $C_{i,j} = 4$;

(6) if $\{P_{i,j-1}$ and $P_{i-1,j+1}\}$ or $\{P_{i,j+1}$ and $P_{i+1,j-1}\}$, then $C_{i,j} = 5$;

(7) if $\{P_{i-1,j}$ and $P_{i+1,j-1}\}$ or $\{P_{i-1,j+1}$ and $P_{i+1,j}\}$, then $C_{i,j} = 6$;

(8) if $\{P_{i-1,j-1}$ and $P_{i+1,j}\}$ or $\{P_{i-1,j}$ and $P_{i+1,j+1}\}$, then $C_{i,j} = 7$;

(9) if $\{P_{i-1,j-1}$ and $P_{i,j+1}\}$ or $\{P_{i,j-1}$ and $P_{i+1,j+1}\}$, then $C_{i,j} = 8$;

(10) if $\{P_{i-1,j-1}$ and $P_{i+1,j-1}\}$ or $\{P_{i-1,j+1}$ and $P_{i+1,j+1}\}$, then $C_{i,j} = 9$.

Rule 2 : If more than one of the above conditions are satisfied, then

$$C_{i,j} = -10.$$

Rule 3 : If none of the above conditions is satisfied, then

$-8 \leq C_{i,j} \leq -1$, and

(11) if $P_{i-1,j-1}$, then $C_{i,j} = -1$;

(12) if $P_{i-1,j}$, then $C_{i,j} = -2$;

(13) if $P_{i-1,j+1}$, then $C_{i,j} = -3$;

(14) if $P_{i,j+1}$, then $C_{i,j} = -4$;

(15) if $P_{i+1,j+1}$, then $C_{i,j} = -5$;

(16) if $P_{i+1,j}$, then $C_{i,j} = -6$;

(17) if $P_{i+1,j-1}$, then $C_{i,j} = -7$;

(18) if $P_{i,j-1}$, then $C_{i,j} = -8$;

With the codes defined above, the meaning of each code can be summarized as follows :

(1) $C_{i,j} = -10$ implies $P_{i,j}$ is a branch point;

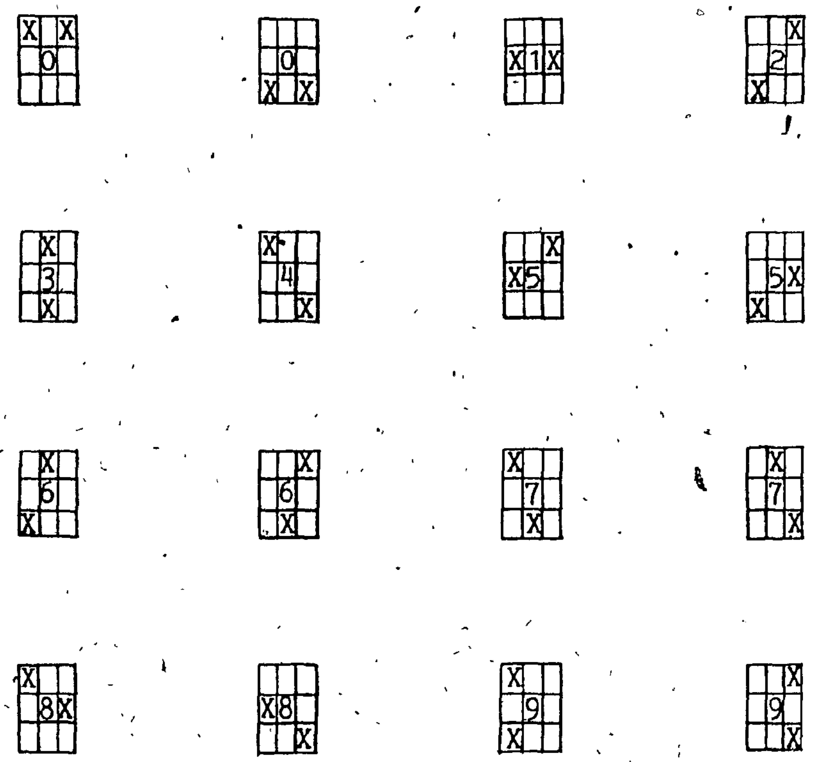
(2) $-8 \leq C_{i,j} \leq -1$ implies $P_{i,j}$ is a terminal (end) point;

(3) $C_{i,j}$ in {0, 5, 6, 7, 8, 9} implies $P_{i,j}$ is likely a turning point of a sharp intrusion;

(4) $C_{i,j}$ in {1, 2, 3, 4} implies $P_{i,j}$ is a point on a horizontal, diagonally right, vertical, diagonally left line respectively. It is illustrated in Fig. 2.3

2.5 Graph Representation

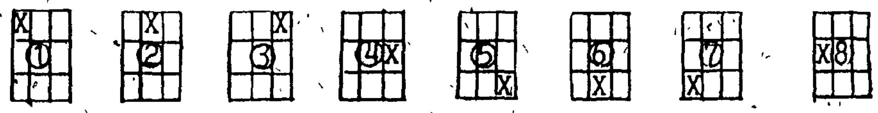
The idea to depict a character as a graph was motivated by Stallings [32], Caskey [4] and Watt [40], etc. The transformation of a binary matrix of black and white points into a graph with the direction of its edges defined, is intuitively advantageous in the description of the picture



where X is a pixel in the skeleton;

0 is a pixel with condition code 0.

(a) The Positive Condition Codes for Rule 1 from 0 to 9.



where ① means minus 1.

(b) The Negative Condition Codes for Rule 2 from -1 to -8.

Figure 2.3 Condition Codes Assigned to a Pixel in the Skeleton.

of a character. A graph does not only give the structure of a character but can also be stored in a compact form such as an adjacency matrix instead of a digitized image matrix occupying a large storage. In addition, this representation also facilitates the extraction of topological features and a graph can be encoded to give a string of feature primitives for syntactic recognition.

A graph can be denoted as an ordered pair G = (V,E) where V is the set of vertices (or nodes) and E is the set of edges. Each edge is assigned an integer to denote the direction of the edge in the plane. Graph nodes can be classified into three kinds according to their degree. The terminology is explained as follows :

(1) a terminal node is a vertex of degree 1;
(2) a corner node is a vertex of degree 2;
(3) a junction node is a vertex of degree greater than 2.

An edge from vertex i to vertex j is denoted by E_i^j.

The program used in the system to construct the graph of a character comprises three distinct procedures : (1) 'TRACE' (2) 'SEETURN' (3) 'ASSIGN'. By calling procedure 'TRACE', a segment of the skeleton is traced and terminal nodes or junction nodes are located. This segment is then passed to procedure 'SEETURN' to locate all the corner nodes, and each segment joining any two nodes is defined as an edge. Finally, procedure 'ASSIGN' is called to quantize the slopes of the edges into eight directions. Details of these three procedures are discussed in the following subsections.

2.5.1 Tracing Process

The 'TRACE' procedure is used for tracing the skeleton of the character. Using the condition codes defined in the previous section, tracing is accomplished by moving from point to point to traverse the whole skeleton of the character. In particular, the end points and the branch points can be located by their condition codes which either range from -1 to -8, or are -10 respectively. The aim of the tracing process is to search all the relevant graph nodes to represent the character. An end point is taken as a terminal node while a branch point can contribute to form a junction node.

The starting point of the tracing process is chosen from the end points if any, otherwise from the branch points. In the case of single loop, when there is no end point or branch point, the upper leftmost point is used. A node is first created at the starting point. Tracing halts once an end point or a branch point is encountered which implies that a potential node has been found. During the tracing of this segment, all the points are linked to form a list for further detection of corner nodes.

2.5.1.1 Segment Tracing

There are four cases which stop the tracing of a segment, and the corresponding actions to complete the tracing are described below. Fig. 2.4(a) illustrates the four cases.

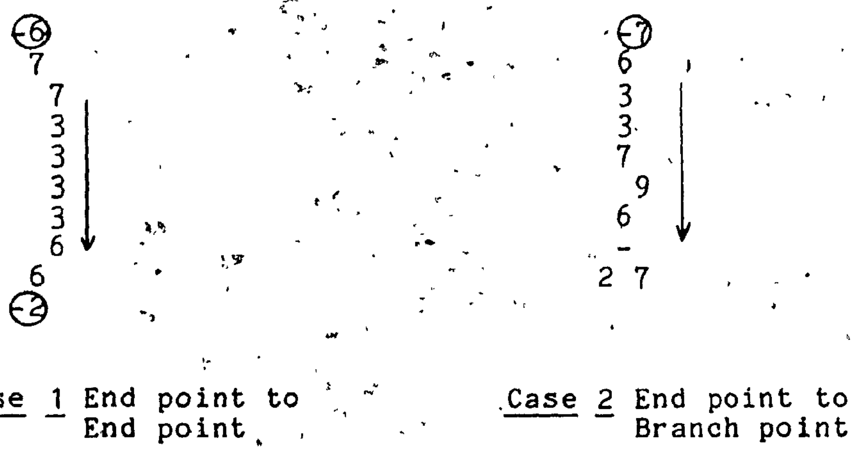
(1) If a segment is traced from a terminal node to an end point, the segment is the only path of the graph and a new terminal node is created.

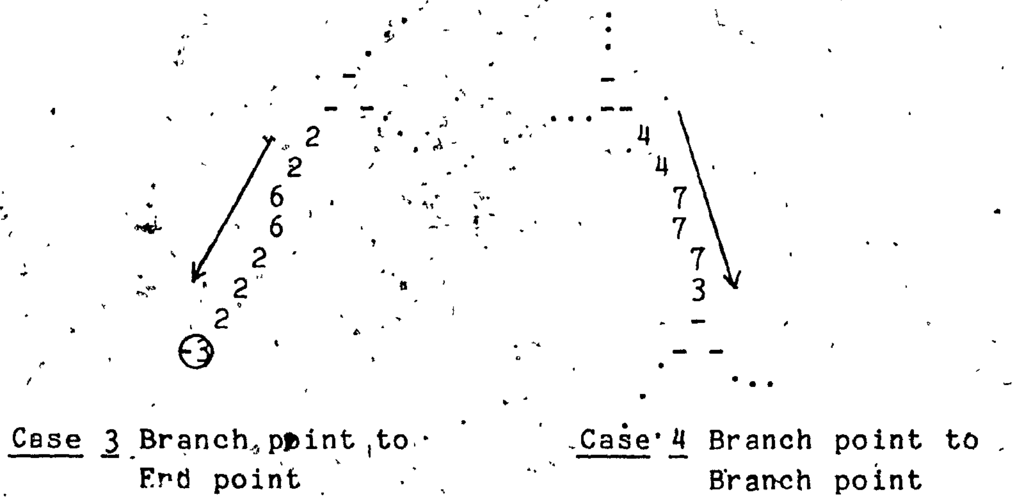
(2) If a segment is traced from a terminal node to a branch point, then

- (i) a procedure 'FINDBRANCH' is called to find out all the possible branches originated from this branch point;
- (ii) all the branch points encountered during this process are grouped together to form a junction node by averaging all their positions;
- (iii) a stack 'BRANCHSTACK' for this node is built up to store its branches.

(3) If a segment is traced from a junction node to an end point, a terminal node is created and the next branch from the 'BRANCHSTACK' of this junction node will be processed.

(4) If a segment is traced from a junction node to a branch point, the procedure in case (2) is again applied. In addition, the new junction node created is stored in a stack 'NODESTACK'.


6
7
7
3
3
3
3
6
6
2
Case 1 End point to End point
7
6
3
3
7
9
6
2 7
Case 2 End point to Branch point


2
2
6
6
2
2
2
3
Case 3 Branch point to End point
4
4
7
7
7
3
Case 4 Branch point to Branch point

–	branch point	⟶	tracing direction
○	end point	...	another segment

Figure 2.4(a) Four Cases in Segment Tracing.

For each node in 'NODESTACK', its branches are processed until its 'BRANCHSTACK' is empty.

2.5.1.2 Branch Searching

'FINDBRANCH' is a procedure used to find all the possible branches originated from a branch point. The principle used is to search through the eight neighbour points of the branch point. For every branch point encountered, the searching process is repeated to ensure all the branches are located. A branch is said to be found if a point which is neither an end point nor a branch point is encountered during the recursive searching. This point is usually the starting point of a branch and is stored in the 'BRANCHSTACK' according to its positions. Fig. 2.4(b) illustrates the result of branch searching.

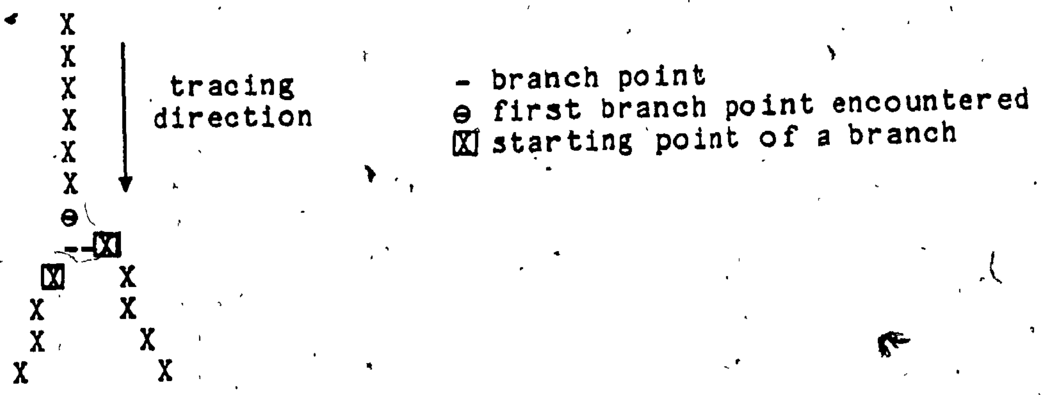

tracing direction
- branch point
⊖ first branch point encountered
☒ starting point of a branch

Figure 2.4(b). Result of Branch Searching.

2.5.1.3 Tracing Techniques

During the tracing, a technique of 'switching off' is used. Once a point is traced, it is wiped off and no more regarded as a skeleton point. Consequently, no branch is repeated for a given node in the procedure 'FINDBRANCH'. Similarly, once a given segment is traced, it will not be considered any more. This convention has the advantage of preventing any duplicate vertices from appearing in the graph representation when the character contains a loop. Moreover, detection of disconnected components in the character can be performed easily by checking if there is any remaining skeleton point.

Every time the procedure 'TRACE' is called, a segment is traced and a linked list is formed and passed to the procedure 'SEETURN' to detect any corner nodes within the segment. For each node in the 'NODESTACK', 'TRACE' is called to process all its branches. Calling of 'TRACE' continues until the 'NODESTACK' has been emptied.

2.5.2 Locating Corner Nodes

Given a curve segment, in order to represent it in a graph form, all the sharp corners should be detected because of their significance in the structural description of the segment. 'SEETURN' is a procedure which performs this corner detection of each segment extracted by the tracing process and it creates corner nodes if necessary.

The corner detection again makes use of the condition codes to find potential turning points. The principle is to measure the angular difference of successive segments at those turning points. Through experimental testing, some thresholds are derived to define the appropriate length of a straight line segment and the critical angle between two line segments to define the corner. An algorithm has been developed to divide a curve segment into an appropriate number of straight line segments which are connected by corner nodes. The algorithm and notations used are presented below.

Denote the sequence of points of a curve segment by

$$P^n_{j=1} = \{p_1, p_2, \ldots, p_n\} ;$$

where p_1 and p_n are the points equivalent to the starting and end nodes respectively.

Let

C_j be the condition code for p_j ;

$T = \{0, 5, 6, 7, 8, 9\}$ be the set of condition codes of the turning points ;

L_t be the threshold for the normal length of a straight line segment ;

M_l be the threshold of minimum length of a straight line segment ;

$\varnothing_t$ be the threshold of the angle between two segments ;

$\emptyset_i^j$ be the slope of the line segment joining p_i and p_j ;

s be the starting position of a straight line segment.

The algorithm is summarized in the following pseudo-code.

```
Given a line segment $P_{j=1}^{n}$,
STEP 1 :(* Initialize the starting point of a new
    segment. *)
        s <--- 1 ;
STEP 2 :(* A minimum segment length is set. *)
        j <--- s + $L_t$ ;
STEP 3 :(* Proceed to next point in the list until a
    turning point is found or end of list is reached.
    *)
        REPEAT
            j <--- j + 1 ;
        UNTIL ($C_j$ in T) or (j = n) ;
        IF j = n GOTO STEP 6 ;
STEP 4 :(* Test if the turning point is relevant. *)
        k <--- min {j + $L_t$, n-j} ;
        IF (k < $M_1$) GOTO STEP 6 ;
STEP 5 :(* Test the angle between two segments at the
    turning point. *)
        IF $\emptyset_t < \left|\emptyset_s^j - \emptyset_j^{j+k}\right| < \pi - \emptyset_t$ THEN
        Assign $p_j$ to a corner node connecting $p_s$ ;
        s <--- j ;
        GOTO STEP 2 ;
      ELSE GOTO STEP 3 ;
STEP 6 :(* The end of list is reached. *)
      Assign $p_s$ to connect with $p_n$.
```

2.5.3 Quantization of Node Linkage

A straight line segment connecting two nodes is now represented by an edge. The orientation of an edge is quantized to one of the eight directions shown in Fig. 2.5(b) to encode the graph.

Consider the set of nodes of the graph consisting of m nodes as $N_m = \{n_i / 1 \leq i \leq m\}$. Given an edge E_i^j from node n_i to node n_j with coordinates (x_i, y_i), (x_j, y_j) respectively, the inclination of E_i^j is given by

$$\theta_i^j = \tan^{-1} \frac{y_j - y_i}{x_j - x_i} \quad \text{if } x_i \leq x_j$$

$$= \tan^{-1} \frac{y_j - y_i}{x_j - x_i} + \pi \text{ if } x_i > x_j .$$

It is illustrated in Fig.2.5(a).

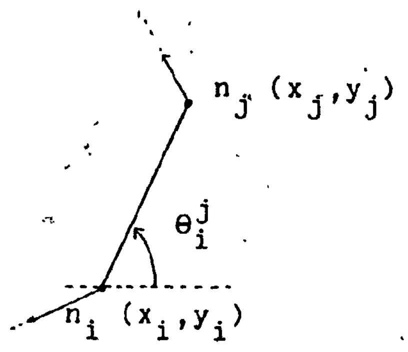


Figure 2.5(a) The Inclination of an Edge.

Let D_i^j be the quantized direction code of E_i^j ;

θ_t be the threshold of quantization angle.

The quantization is given as follows :

(1) if $-\theta_t \leq \theta_i^j \leq \theta_t$, $D_i^j = 0$

(2) if $\theta_t < \theta_i^j < \frac{\pi}{2} - \theta_t$, $D_i^j = 1$

(3) if $\frac{\pi}{2} - \theta_t \leq \theta_i^j \leq \frac{\pi}{2} + \theta_t$, $D_i^j = 2$

(4) if $\frac{\pi}{2} + \theta_t < \theta_i^j < \pi - \theta_t$, $D_i^j = 3$

(5) if $\pi - \theta_t \leq \theta_i^j \leq \pi + \theta_t$, $D_i^j = 4$

(6) if $\pi + \theta_t < \theta_i^j < \frac{3\pi}{2} - \theta_t$, $D_i^j = 5$

(7) if $-\frac{\pi}{2} - \theta_t \leq \theta_i^j \leq -\frac{\pi}{2} + \theta_t$, $D_i^j = 6$

(8) if $-\frac{\pi}{2} + \theta_t < \theta_i^j < -\theta_t$, $D_i^j = 7$

Fig. 2.5(b) shows the quantization in terms of θ_t.

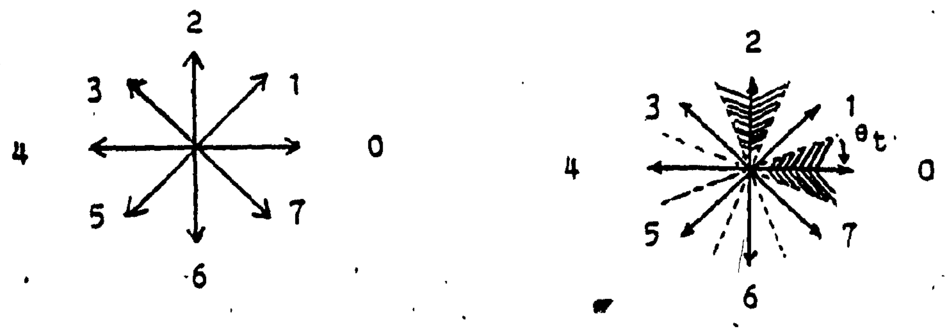


Figure 2.5(b) Quantization into Eight Directions.

The quantization of edges into eight directions (0..7) initiates the data structure of graph nodes linkage to become an 8-tuple $\{L_i[K],\ K = 0..7\}$. The graph of a character can then be described by the interconnections of graph nodes in terms of link tuples. The characteristics of the link tuple are listed below :

(1) $L_i[K] = j$ if node n_i is linked to node n_j in the K-direction, $K = 0..7$;

(2) $L_i[K] = 0$ if node n_i is linked to no node in the K-direction ;

(3) $L_i[K] = j \Leftrightarrow L_j\ [(K + 4)$ Modulus $8] = i$;

(4) $L_i[K] = j$ and $L_i[K] = m \Rightarrow j = m$.

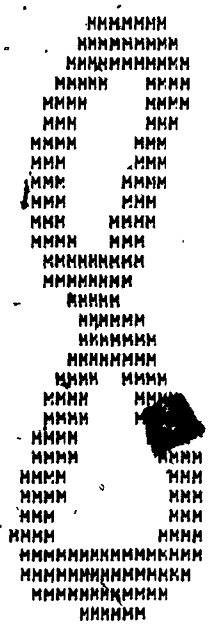
With the above definition and properties of the linkage structure, an algorithm to encode the graph is proposed in the next chapter.

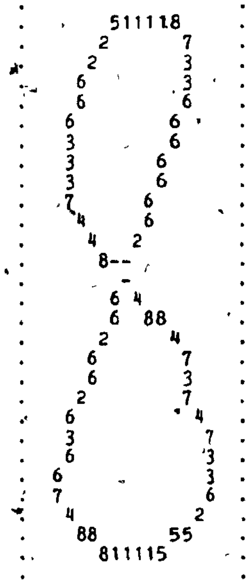
2.6 Concluding Remarks

The basic aim of this chapter is to present the processes of transforming the pixel by pixel binary image matrix of a character into a graph form. This graph is different from Caskey et al's [4]. They used each pixel as the node of the graph of a normalized character which produces a complicated graph for simple character while our use of end points, turning points and junction points as graph nodes, simplified the graph. Traversal of the graph is then easier and less time consuming when extracting the primitives.

During the graph forming processes, the values of thresholds used in each particular case were set through experimental observations and these thresholds are listed in the Appendix. An example is shown in Fig. 2.6 to illustrate the original image, its graph form with the adjacency matrix and the processed thinned-line codes. Since it is impossible to illustrate the two directions of an edge at the same time in the picture, referring to Fig. 2.5 (b), a set of codes, namely H(horizontal)-(0,4), V(vertical)-(2,6), R(diagonally right)-(1,5) and L(diagonally left)-(3,7) is used.

The graph developed up to this point is important since further analysis of a character considers only the graph vertices and their interconnections which is no longer considered at the pixel level. The utilization of this graph representation in encoding and extraction of primitives is discussed in the next chapter.

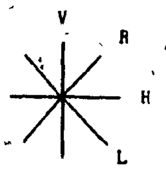

(a) Digitized Image


- denotes $C_{i,j}$ = -10

(b) Thinned-line codes

Vertices :

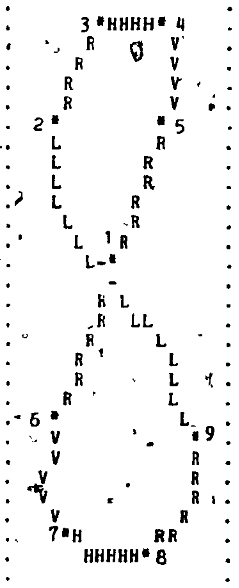
1 : (14,9)
2 : (7,4)
3 : (2,8)
4 : (2,13)
5 : (7,13)
6 : (22,4)
7 : (28,5)
8 : (29,12)
9 : (23,16)



THE ADJACENCY MATRIX :-

From\To	1	2	3	4	5	6	7	8	9
1	-	3	-	-	1	5	-	-	7
2	7	-	1	-	-	-	-	-	3
3	-	5	-	0	-	-	-	-	-
4	-	-	4	2	6	-	-	-	-
5	5	-	-	2	-	-	-	-	-
6	1	-	-	-	-	-	6	-	-
7	-	-	-	-	-	2	-	0	-
8	-	-	-	-	-	-	4	-	1
9	3	-	-	-	-	-	-	5	-

(c) Adjacency Matrix


* denotes graph nodes

(d) Graph Representation

Figure 2.6. Reduction of a Digitized Image to its Graph Representation.

CHAPTER THREE

GRAPH ENCODING AND EXTRACTION OF PRIMITIVES

3.1 Introduction

As pointed out in Chapter one, to perform syntactic analysis of a character pattern class, a 'pattern description language' is needed. This language should provide the structural description in terms of the relationships between pattern primitives of the character class. The selection of primitives is a pattern dependent problem which has no general solution. For different patterns and applications, different types of primitives such as a pixel [15,35], unit length line segments [10], and geometrical shape [1,26] are used.

In this chapter, we propose, from the graph representation developed in the last chapter, an encoding algorithm to generate a numeric code for a character. This code has the function of guiding the extraction of primitives as well as expressing the relations between primitives. The extraction scheme is presented in details. The ultimate goal of this chapter is to establish a string of primitives ready for syntax analysis.

3.2 Graph Encoding

Methods of encoding a character have been proposed by different researchers [8,29]. Their conventions usually have the purpose of simplifying the characteristic extraction and the recognition process. The encoding algorithm developed in this research also has three specific purposes. First, from the graph representation discussed in chapter two, the graph structure is encoded into a more compact form to describe the character shape. Instead of storing the information in terms of a data structure with eight links, a numeric code string can be used to describe the character structure. Second, with this numeric code, reconstruction of the original shape of the character can be accomplished. Third, the encoding scheme enables us to achieve the main objective of detecting the geometrical features of the character as well as concatenating them.

The convention of the algorithm is to encode the boundaries of the graph. In the case of the character with no embedding loops, the code just describes the shape of the character, while for a character which consists of holes, the codes for both the outer and inner boundaries are generated. The encoding is accomplished by the traversal of all the vertices in the graph.

3.2.1 Encoding Algorithm

In the encoding, there are two basic problems to be solved. First, the starting point of the encoding has to be chosen so that all the geometrical structures can be retained in the code. Second, the algorithm should be able to traverse the whole set of vertices of the connected graph. To solve the first problem, we adopt the convention that a vertex is chosen as the starting point according to the following priorities : 1) a terminal node, 2) a corner node, and finally, 3) a junction node. If more than one potential starting point exists, the upper left-most one is chosen.

To solve the second problem, the following convention is used : starting from the first node, the lowest numbered non-empty direction link (0 to 7) is chosen. For every move, starting from the direction opposite to which it came from, search anti-clockwise in the directions of the eight-direction links, the first non-empty link encountered is chosen as the path to the next node. In this way, the path which returns to the node from where it came is taken as the last resort. To avoid the repetition of the same direction path between two nodes, a direction link is disconnected once it has been processed. Fig.3.1 shows this convention.

Given a graph, assume its set of nodes consisting of m distinct elements be

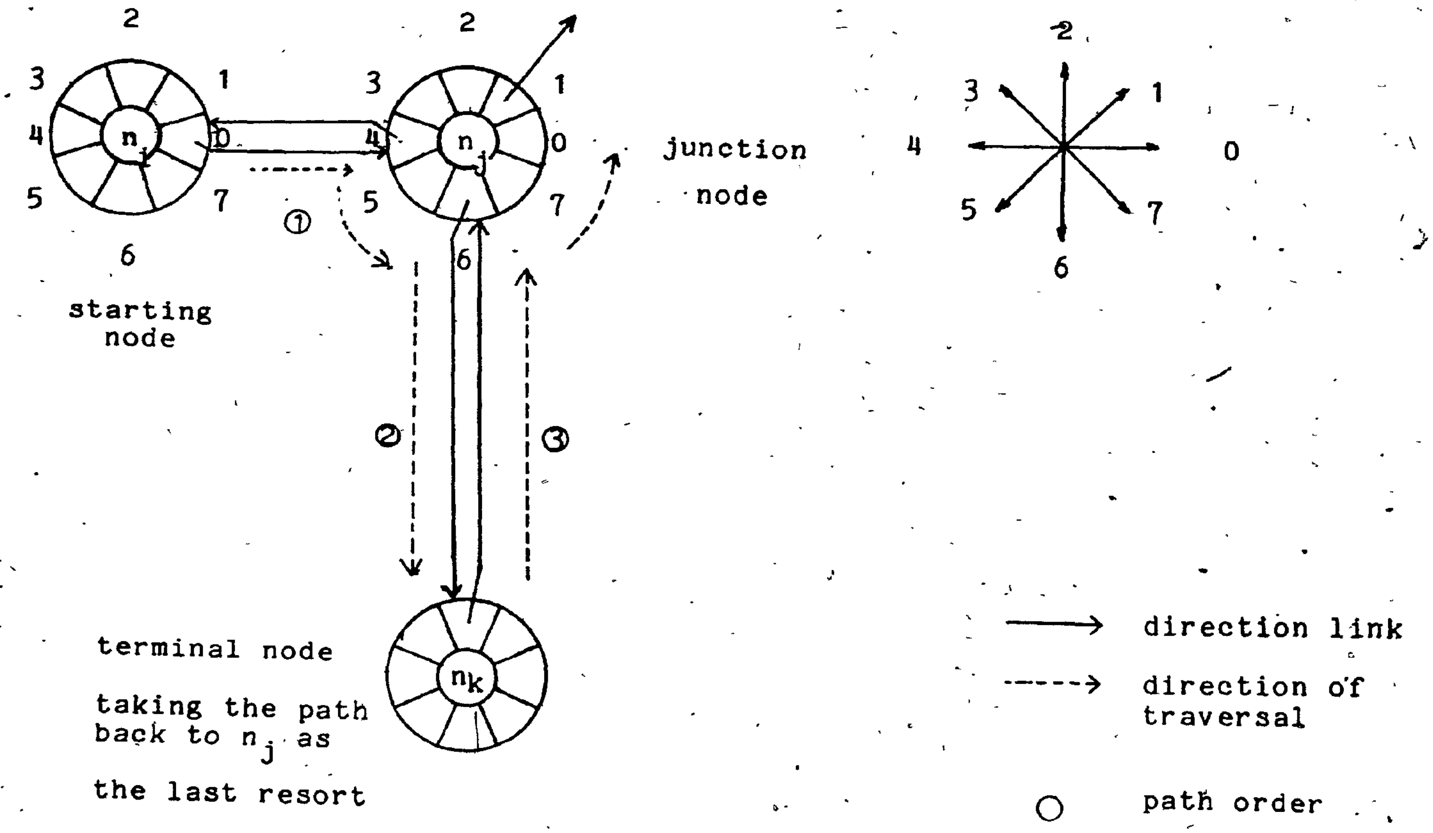


Figure 3.1. Traversal Convention.

$$N_m = \{n_1, \ldots, n_m\},$$

where n_1 is the starting node;

$L_i[k] = j$ denotes that node n_i is linked to n_j in the k-direction.

Let s be the numeric code string to be evaluated;

$D[n_i]$ be the degree of n_i ; and

'//' denotes the concatenation of two code strings.

The algorithm can be described in the following pseudo-code.

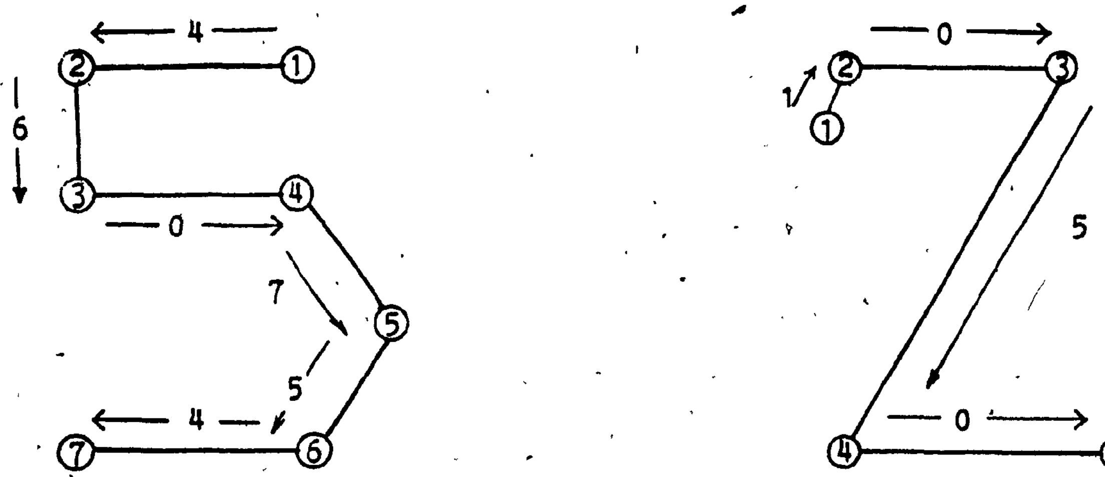
```
STEP 1  :(* To initialize the starting node and direction.
   *)
        $n_i$ <--- $n_1$ ;
        k <--- min{ k'/ $L_i[k'] \neq 0$} ;
STEP 2  :(* Start a new code string.  *)
        s <--- '*' ;
STEP 3  :(* Keep the direction code *)
        IF ($D[n_i] > 2$) THEN
            set FLAG on ;
        j <--- $L_i[k]$ ;
        $L_i[k]$ <--- 0 ;
        s <--- s // 'k' ;
STEP 4  :(* Look for the next path.  *)
        i <--- j ;
        k' <--- (k+4) Modulus 8 ;
        k <--- (k'+1) Modulus 8 ;
        WHILE ($k \neq k'$) AND ($L_i[k] = 0$) DO
            k <--- (k+1) Modulus 8 ;
        IF ($k \neq k'$) GOTO STEP 3 ;
```

```
STEP 5  :(* Process until whole graph is encoded.  *)
        s <--- s // '*' ;
        IF (FLAG is on) AND ($L_i[k] \neq 0$) GOTO STEP 2.
```

There are three cases to be considered in the traversal.
Case 1 : For a graph with no junction node, it is sufficient to traverse all the nodes just once in order to describe the character shape. For example, in Fig. 3.2(a), traversing nodes (1),(2),(3),..,(7) yields the codes 4,6,0,7,5 and 4.
Case 2 : For a graph with junction nodes but no inner loops, each junction node must be traversed the number of times equal to its degree. For example, the junction node (3) is traversed four times in Fig. 3.2(b).
Case 3 : For a graph with inner loops as illustrated in Fig. 3.2(c), when the encoding halts at the starting node (1) with the inner boundary not encoded, then the upper left-most junction node (3) which is not yet traversed completely is taken as the new starting node.
With cases 2 and 3, there is more than one code string generated from the graph.

3.3 Extraction of Primitives

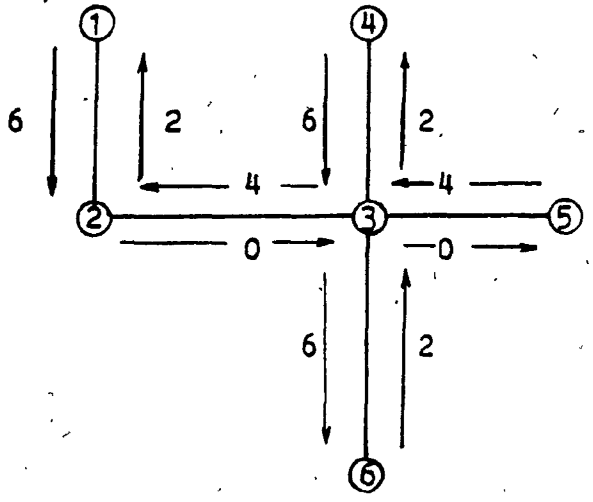
The extraction of primitives is basically a two step process : (i) the selection of primitives and (ii) the identification of the primitives. We have three main criteria to select primitives. First, primitives selected should be the basic structural units which can contribute

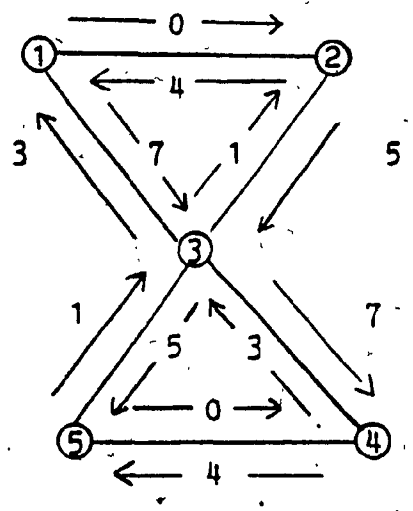


s = * 460754 *

s = * 1050 *

(a) Graph Examples with Terminal and Turning Nodes Only.





s = * 606 *
* 20 *
* 42 *
* 642 *

s = * 057413 *
* 714 *
* 503 *

(b) Graph Example with Junction Node.

(c) Graph Example with Inner Loops.

Figure 3.2. Examples of All Cases of Encoding.

the maximum information of the character pattern. Second, the primitives should be able to characterize the character. Third, they can be extracted easily.

We have mentioned previously that many kinds of primitives can be used to classify different types of patterns. Some of the primitives proposed for character patterns are the direction change in a pixel and its eight neighbours in the skeletonized image which is frequently called the Freeman code [8,10]. One major drawback of using each pixel as primitive is that extremely long and complicated string-representation will result. Moreover, in a digitized image, a pixel which is in fact the most primitive unit, does not contribute concrete global information about the geometrical structure and thus diminishes the efficiency of the classification process.

On the other hand, with the grouping of certain connected pixels, we can have a piecewise string of pixels as primitive to describe the character image. For example, by the reduction of a character pattern into graph-representable form in this work and in that of Watt and Beurle [40], the edges can be used as primitives. However, there are still variations in the quantized edges which could lead to a considerably complex process in formulating the pattern language.

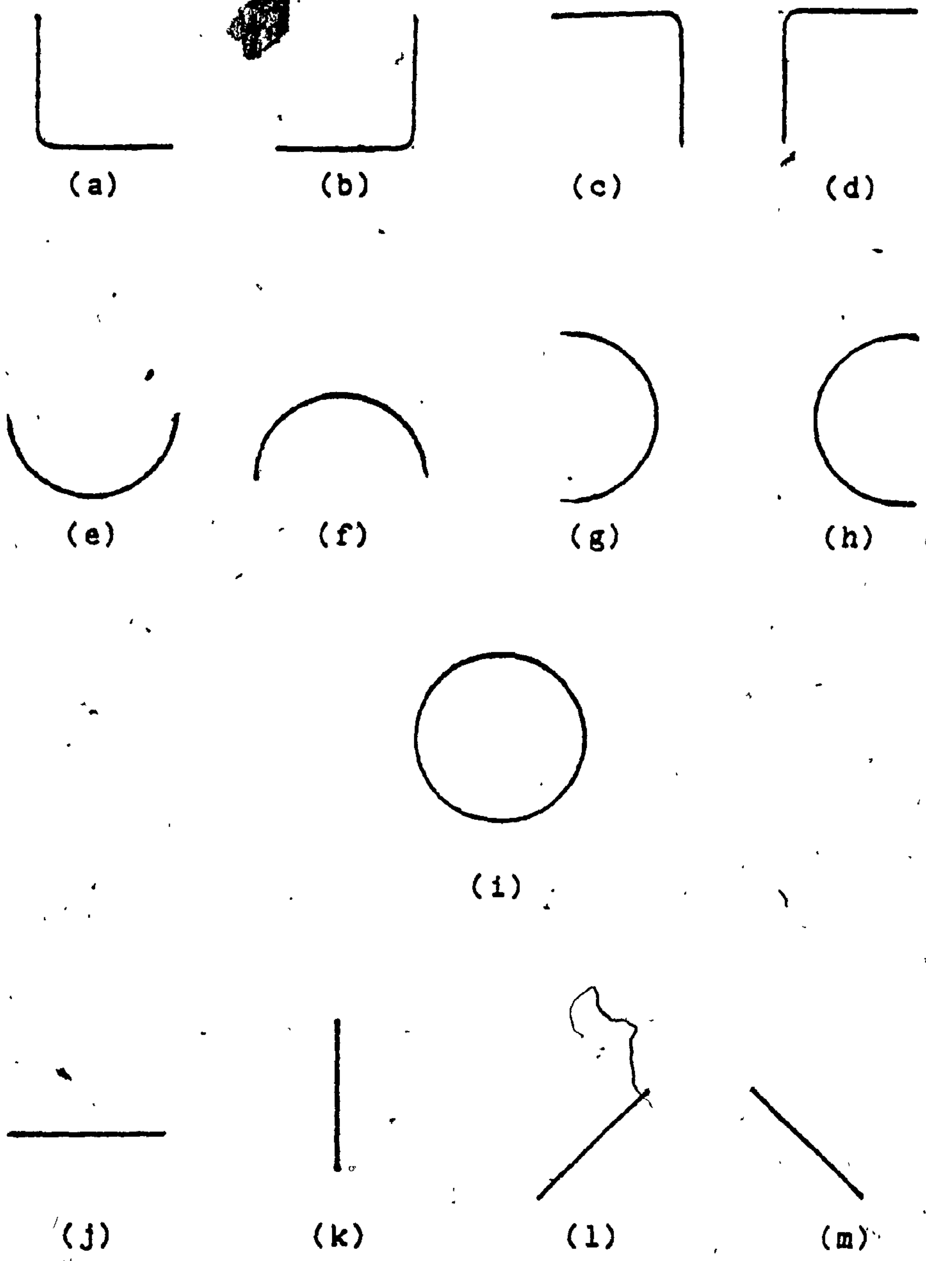


Figure 3.3. The Set of Primitives Defined by Topological Features.

In this system, topological features such as bays, loops and strokes in various orientations as shown in Fig.3.3 are used as primitives. They were chosen for two main reasons:

(1) Topological features are basic structures of most character patterns. The use of these structures is also part of the process of human perception of character. These structures have been used by various researchers through the theoretic-decision approach [4,37].

(2) With the graph-representation discussed above, the extraction of topological features becomes simple. The algorithm will be explained in the following sections.

The topological features we used are classified according to the orientation of the cavities, strokes and the position of loops.

3.3.1 Detection of Primitives

The extraction of primitives is guided by the encoding of the shape of the character. While traversing the graph of a character, the direction change from one edge to another edge is noted. The direction change is defined as the minimal change in a circular sense. For example, a change in the direction code from 0 to 5 in Fig.3.4 implies a clockwise change in direction since it gives shorter path than the anti-clockwise change. Then, according to the direction change in an anti-clockwise and clockwise sense, a sign is assigned to the primitive described by these edges as positive or negative respectively.

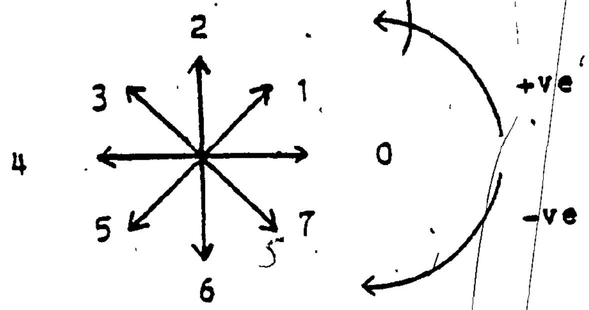


Figure 3.4. 8-Directions with Assigned Signs

During the traversal of edges, a primitive is built up by these edges until a change in sign results. Any difference in the sign of direction change indicates that the extraction of the current primitive is completed and a new primitive is expected. With the sign of each primitive and the end nodes (vertices) of this topological feature, its type can be determined. Fig.3.5 shows some examples of the detection of primitives.

The algorithm for the detection of primitives is given below.

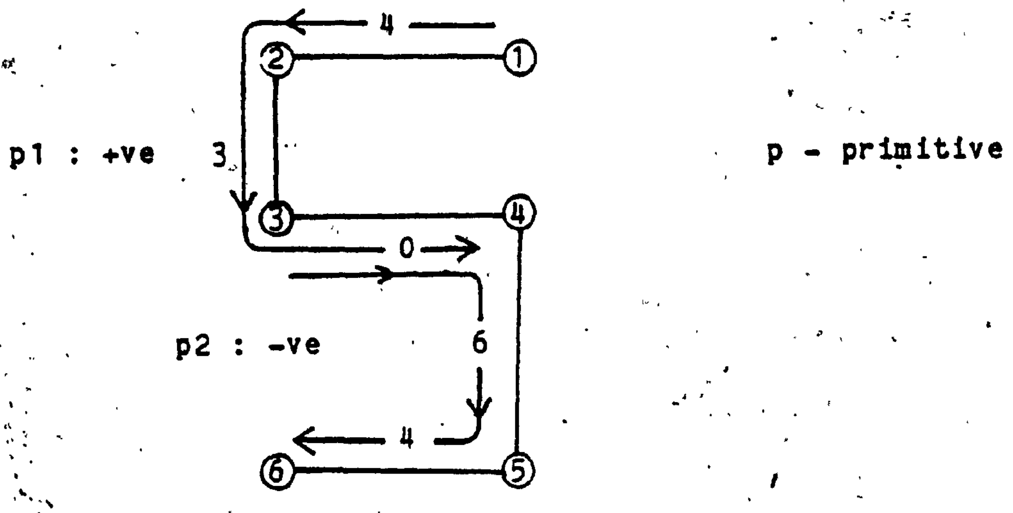
Let $d_{i,j}$ be the direction code of the edge from node n_i to n_j ;

EXSET be the set of direction codes which indicates the encountering of a new primitive.

Assume NEW($d_{i,j}$) returns the direction code during the encoding process.

For each code string :

STEP 1 :(* To initialize the sign of the primitive to be



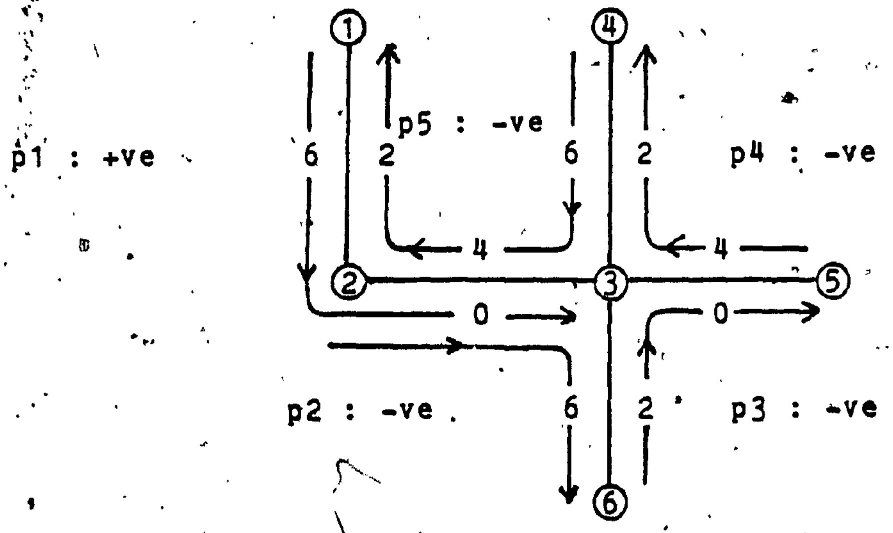


Figure 3.5. Examples for the Detection of Primitives.

```
extracted.  *)
          SIGN <--- 0 ;
          NEW($d_{i,j}$) ;
          IF $n_j$ is the last node THEN
              BEGIN
                  Determine the primitive type by $n_i$, $n_j$ ;
                  STOP
              END ;
          NEW($d_{j,k}$) ;
STEP 2  :(* Assign the sign to the primitive.  *)
          IF ($d_{i,j} < d_{j,k}$) AND ($d_{j,k}-d_{i,j} < 4$) OR
             ($d_{i,j} > d_{j,k}$) AND ($d_{i,j}-d_{j,k} > 4$)
          THEN
              SIGN <--- +1
          ELSE
              SIGN <--- -1 ;
STEP 3  :(*  Extract the whole primitive until a new
          primitive is found or the end of the code string is
          encountered.  *)
          REPEAT
              NEW($d_{k,l}$) ;
              CASE SIGN OF
                  +1 : EXSET = {($d_{j,k}$+c) Modulus 8/ $5 \le c \le 7$} ;
                  -1 : EXSET = {($d_{j,k}$+c) Modulus 8/ $1 \le c \le 3$}
              ENDCASE ;
              IF NOT $d_{k,l}$ IN EXSET THEN
                  BEGIN
          IF $d_{j,k} \neq d_{k,l}$ THEN
              j <--- k ;
          k <---l ;
      END
  ELSE
      BEGIN
          Determine the primitive type by $n_i$ and
          $n_k$ ;
          i <--- j ;
          j <--- k ;
          SIGN <--- -SIGN
      END
UNTIL $n_l$ is the last node.
```

3.3.2 <u>Determination of Primitives</u>

With the primitives detected as described in the previous section, we classify them into three categories. Referring to Fig.3.3, we denote the primitive type by F.

(1) Straight line primitive : it involves no direction change between two end nodes n_i and n_k.

We have,

F = 'j' if $d_{i,k}$ = 0,4 ;

F = 'k' if $d_{i,k}$ = 2,6 ;

F = 'l' if $d_{i,k}$ = 1,5 ;

F = 'm' if $d_{i,k}$ = 3,7 .

(2) Curve segment primitive : it involves direction change

between two end nodes. The line joining the end nodes n_i and n_k of the primitive is quantized to eight directions with a threshold θ_f, see Fig.3.6(a).
Let $Q_{i,k}$ be the quantized direction of the line.
We have $Q_{i,k}$ unchanged if the sign of the primitive is positive ; otherwise, $Q_{i,k}$ is changed to $(Q_{i,k}+4)$ Modulus 8, the opposite direction. See Fig.3.6(b).
We have,

F = 'e' if $Q_{i,k} = 0$;
F = 'b' if $Q_{i,k} = 1$;
F = 'g' if $Q_{i,k} = 2$;
F = 'c' if $Q_{i,k} = 3$;
F = 'f' if $Q_{i,k} = 4$;
F = 'd' if $Q_{i,k} = 5$;
F = 'h' if $Q_{i,k} = 6$;
F = 'a' if $Q_{i,k} = 7$.

(3) Loop primitive : it occurs when two end nodes coincide.
We have,
F = 'i'.

As illustrated in Fig.3.7, the strings 'hg' for a '5' and 'acdae' for a '4' are derived from the rules described above.

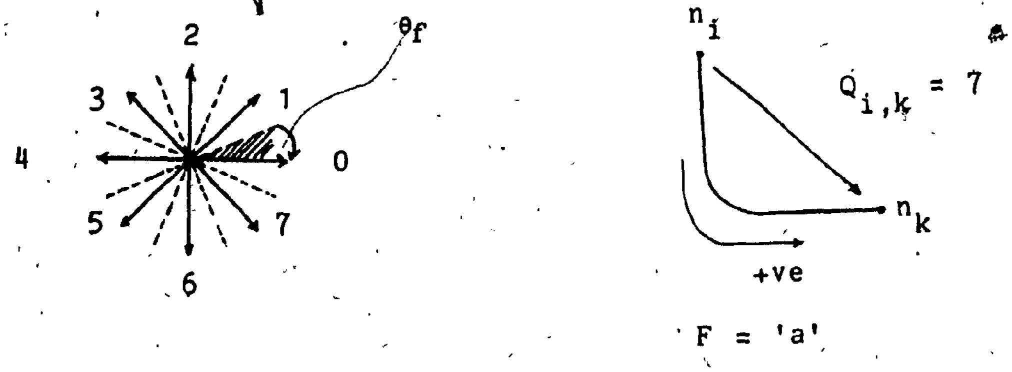


Figure 3.6(a) Quantization of the Line Joining End Nodes.

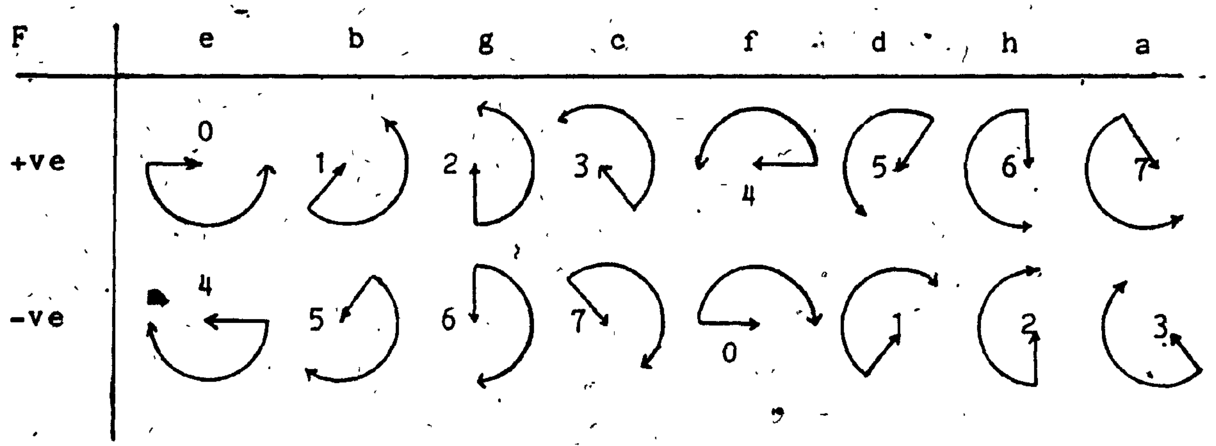
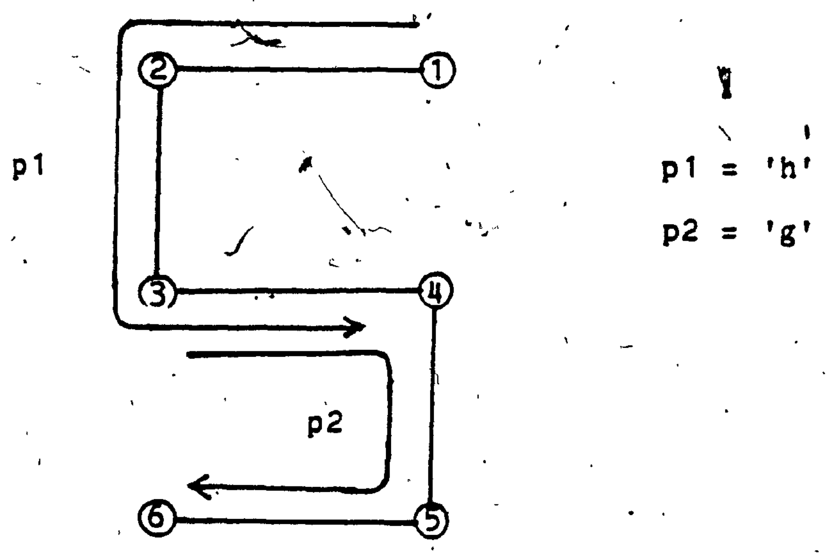
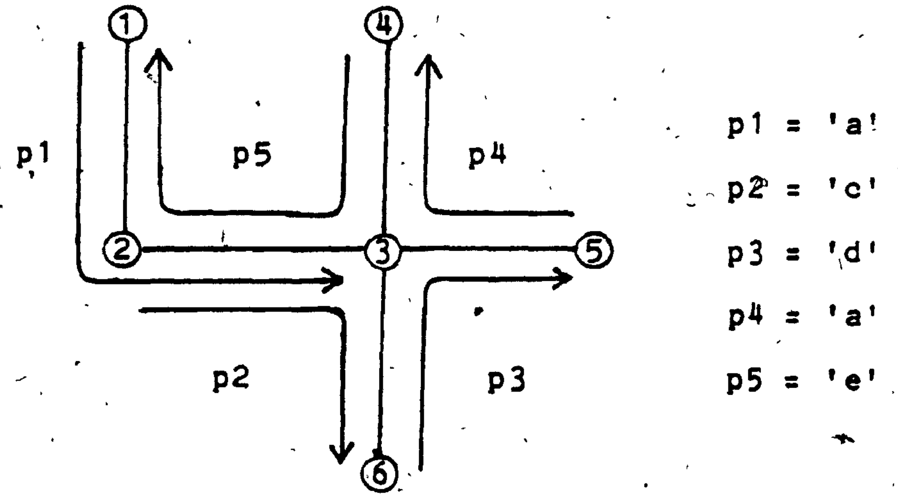


Figure 3.6(b) Determination of Curve Segment Primitives.



Primitive string = hg.



Primitive string = acdae.

Figure 3.7. Examples for the Determination of Primitives.

3.4 Characteristic Measurements of Primitives

Using topological features as primitives has reduced the complexity of the pattern language. However, due to the quantization of the directions between the graph vertices, some mathematical measurements might be required in order to interpret the character image precisely. In a graph representation form, the task of evaluating these meaurements such as the depth of a cavity or the shape of a cavity at a particular part is very easy. In the following, the evaluation of some of the characteristic measurements which we are interested in the proposed system is discussed. All of them involve the properties of loops and cavities.

(1) Loop Position

The position of a loop is very useful both in resolving the ambiguity of primitive strings of different classes and in the interpretation of the character pattern. By observing the span of the loop, that is, the position of its highest node and the distance between the highest and the lowest nodes relative to the height of the character, the location of the loop can be deduced. We classify the loop location into three types : whole loop, upper loop and lower loop. Assume r_h and r_l are the row numbers of the highest and lowest nodes of the loop and T_r the total number of rows of the image matrix. We have,

(a) whole loop, denoted by 'i' if $r_h - r_l > \frac{2}{3} . T_r$.

(b) upper loop, denoted by 'n' if $r_h < \frac{1}{3} . T_r$.

(c) lower loop, denoted by 'o' otherwise.

(2) Opening of a Cavity

We define the opening of a cavity to be the distance between the two end points of the arc. Let the coordinates of the two end nodes of the primitive arc F, be (r_s,c_s) and (r_e,c_e) respectively. We denote the opening of F by

$$O(F) = \sqrt{(r_s-r_e)^2 + (c_s-c_e)^2},$$

which is the distance between the two end nodes.

(3) Depth of a Cavity

In a graph form, we assume a primitive cavity, F, is formed by the node list $\{n_1,n_2,\ldots,n_m\}$. We define the depth of cavity of F

$$D(F) = \underset{1\le i\le m}{\mathrm{Max}} \{d_i\}$$

where d_i is the perpendicular distance of n_i from the line joining the end nodes n_1 and n_m given by :

$$d_i = \frac{c_i(c_m-c_1) - r_i(r_m-r_1) + r_1c_m - r_mc_1}{\sqrt{(c_m-c_1)^2 + (r_m-r_1)^2}},$$

where r_i and c_i denote the coordinates of any node n_i.

(4) Shape of a Cavity

The direction link between nodes provides the shape of a specific portion of the cavity. In particular, to detect if the bottom part of a cavity is flat or not, we observe the direction change of the last two or three nodes.

3.5 Concluding Remarks

A graph encoding convention has been developed to extract primitives and define their structural relationship. The use of topological features has the advantages to contribute maximum interpretation of the character and simplify the pattern descriptive schema. With the encoding, the boundary of a character can be traced to detect these topological features. The algorithm for the detection and determination of primitives shows the effectiveness of the extraction of high-level primitives from a graph representable form. With these high-level primitives, semantic information of various types of primitives such as loop location and characteristic measurements of a primitive can be evaluated. In the next chapter, we shall discuss the classification scheme based on the strings of primitives which provide the structural and semantic information of the characters in each character class.

CHAPTER FOUR

FUNDAMENTALS OF SYNTAX - SEMANTIC DIRECTED CLASSIFICATION SCHEME

4.1 Introduction

The main purpose of a pattern recognition system is to classify the unknown patterns into predefined distinctive classes. To achieve this goal, relevant structural features and characteristic measurements in each pattern are considered in the decision process. Two major approaches, namely, decision-theoretic and syntactic approaches are commonly used in the decision-making process [9]. The former is a classical approach which is based on the formation of a feature vector and the utilization of Bayes classification rule or discriminant functions to partition the feature space [36]. Though this approach plays a significant part in the field of pattern recognition, it is not capable of formulating the pattern structures and their relationships. Thus, in structural-based problems, the expression of a pattern in terms of the relationships of its structures contributes more information and is appropriate for syntactic analysis. In this work, to test the capability of a syntax-directed classifier is the motivation of using the syntactic approach.

The syntactic approach is based on the theory of formal languages. Through decompositions of a pattern into subpatterns and primitives, a pattern language can be established for a given class of patterns analogous to the formal languages, which is defined in terms of a vocabulary and grammar rules. A pattern language has its primitives and subpatterns as the vocabulary and their structural relationships define its grammar. To accomplish the categorization of patterns, syntactic analysis or 'parsing' is performed to check the grammatical correctness of pattern sentences (string).

4.2. Pattern Description Language

The language which provides the structural description of a collection of patterns is called 'pattern description language'. According to Narasimhan [23], 'pattern description language' is defined as the descriptive schema having the following properties :

a) it must assign structures to the character pattern in terms of occurrences of subpatterns and their spatial dispositions; and structures to the subpattern in terms of occurrences of primitives and their compositions.

b) it must assign attributes with computed values to the subparts.

Thus, the descriptions generated provide information of pattern components with such and such properties occurring in such and such configurations. Based on this concept, the transformation of a character into a string of primitives discussed in the last chapter is to build up a 'pattern description language' for a particular character set .

To perform syntactic analysis of the 'pattern description language' generated, we made use of some of the concepts in the formal language theory. Some basic terminology and notations are presented in the following section to facilitate further discussion of the syntactic description schema used in the experiments of this work.

4.2.1 Basic Definitions [13,17]

An alphabet is any finite set of symbols

A sentence(or string) over an alphabet is any string of finite length of symbols from the alphabet.

A string with no symbols is called an empty string.

For any alphabet V, the countable infinite set of all sentences over V, including the empty string, is the closure of V, denoted by V^*. The positive closure of V is the set

$V^+ = V^* - \{z\}$ where z is the empty string.

A context-free grammar is defined formally as a 4-tuple :

$G = (V_N, V_T, P, S)$

where V_N is a finite set of non-terminals or variables ;

V_T is a finite set of terminals or constants ;

P is a finite set of productions or rewriting rules in which each production is of the form $A \dashrightarrow \alpha$ for A in V_N and α in $(V_T \cup V_N)^*$;

S in V_N is the start symbol .

A <u>regular</u> <u>grammar</u> has production of the form $A \dashrightarrow \alpha B$ or $A \dashrightarrow \alpha$ for A, B in V_N and α in V_T.

The <u>language</u> <u>generated</u> <u>by</u> <u>a</u> <u>grammar</u> G, denoted by L(G), is the set {x / x is in V_T and x can be derived from the start symbol by applying zero or more productions from G, denoted by $S \overset{*}{\underset{G}{\Longrightarrow}} x$}.

If a lanugage is generated by a regular grammar, the recognition process can be modelled by a finite automaton.

A <u>finite</u> <u>automaton</u> is a system specified as a 5-tuple,

FA = (K, V, E, S, F)

where K is a finite set of states ;

V is a finite input alphabet ;

E is a mapping from K x V into 2^K, the collection of all subsets of K ;

S in K is the starting state and

F is a set of 'final' or 'accepting' states, a subset of K.

A string x is accepted by a finite automaton if there is a path from the start state to a final state such that the labels along the path spell out x.

4.2.2 Principle of Classification

To apply the above idea for the classification process in a pattern recognition system, we have a recognizer illustrated in Fig.4.1.

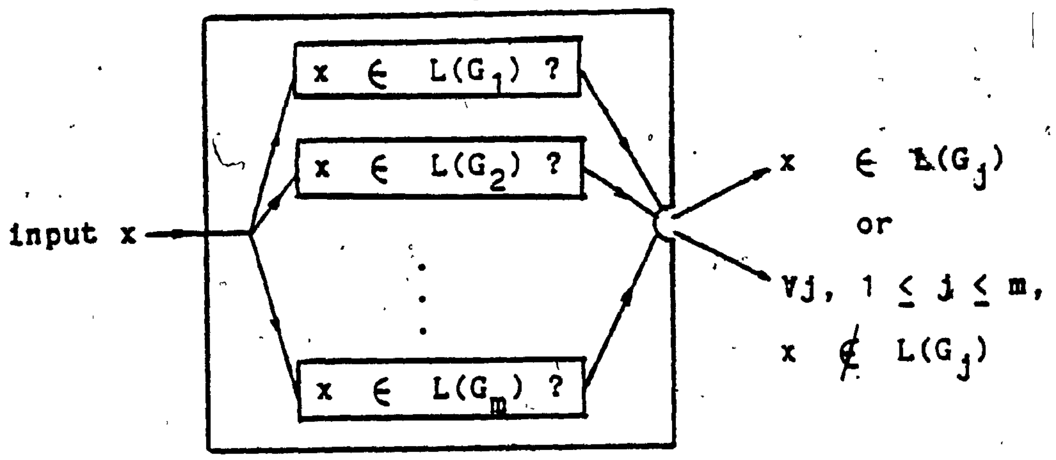


Figure 4.1 Block Diagram of a Syntactic Recognizer.

Assume we have m classes of patterns, $C_1, C_2, \ldots, C_m$ with grammars $G_1, G_2, \ldots, G_m$ for the m classes respectively. Obviously, there should not be any ambiguity between the grammars, that is,

$$\forall\ i,\ j,\ L(G_i) \cap L(G_j) = \emptyset \text{ if } i \neq j.$$

Thus, for an unknown pattern string x, its identity can be determined by testing if x is in $L(G_i)$, $i = 1,\ldots,m$. If for some j, $1 \le j \le m$, x is in $L(G_j)$, then x is recognized from class C_j. If x is not in any of the $L(G_i)$, then x is rejected.

4.2.3 Attributed Grammar Approach

It was found that in handling some types of patterns, syntactic information is insufficient to describe or discriminate the pattern. The introduction of semantic information can supplement the syntactic information to accomplish the description and recognition task. It was introduced by Knuth [19] and Narasimhan[23]. The attributed grammar approach evaluates both syntactic and semantic information. In addition to the syntactic rules in a conventional grammar, an attributed grammar has a set of semantic rules associated with the productions. For a specific primitive, a set of attributes is assigned to the primitive, thus a primitive can be expressed in terms of the values of its attributes to reflect its properties.

There are two kinds of attributes, namely, inherited and synthesized attributes [9]. An inherited attribute is derived from the context of a string while the synthesized attribute is built from within the primitive. For example, a curve segment when defined as a primitive, its curve structure is an inherited attribute but the radius of curvature or the length of the curve segment is a synthesized attribute.

In brief, each of the primitives in an attributed grammar has a symbolic part for syntax parsing and a value part for semantic parsing. However, it is not always necessary to assign attributes to every primitive. In this work, attributes are introduced to solve the ambiguity between some grammars.

4.3 Classification Scheme of the System

The classification scheme in the system makes use of both syntactic and semantic analyses. There are two advantages in this scheme :

(1) Using simple syntactic description and little semantic information, a powerful classifier can be developed.

(2) The syntax analysis and semantic analysis can be carried out in parallel.

In this work, regular grammars are used. A finite automaton can thus be built up for this type of grammar as the recognizer. In practical implementation, the semantic rules are embedded in the syntactic rules so that the parsing of the symbolic part leads to the testing of attributes also. An unknown primitive string is parsed by the grammar of each class. If the unknown input is successfully parsed by any of the grammars, then it is accepted, otherwise it is rejected. The attributed grammar for each class of numerals is listed in the appendix. The grammars are listed in their context-free form to shorten the list.

4.3.1 Syntactic Descriptive Schema

With the encoding scheme discussed in the previous chapter, patterns are described in terms of the topological features in their order of appearance during the encoding process. According to Narasimhan's descriptive schema [23], this encoding of graphic representation of a character describes the occurrence of specific structure in a specific order and thus produces a string of primitives. The order in which primitives occur reflects the inherent relationship between them. Consequently, this order of occurrences of primitives can serve to characterize the structural shape of individual characters. To inference the grammar for each character class, strings of primitives generated from the training samples were considered. In each class, the strings which correctly described the structure of the corresponding characters were used to form the set of typical references, called prototypes for that class. The grammar for a character class was inferenced heuristically by its prototypes. Given a primitive string, f, it is parsed through the grammar of each class. If f is successfully parsed by G_i, $0 \leq i \leq 9$, then f is classified into class i.

Let us denote the grammar for each class by G_i, $0 \leq i \leq 9$. We define the set of terminals by $V_T = \{a,b,c,\dots,m\} \cup \{n,o\}$ as illustrated in Fig.3.3 and section 3.4.

For $G_i = \{V_{i_T}, V_{i_N}, P_i, S_i\}$, $0 \leq i \leq 9$, $\bigcup_{i=0}^{9} V_{i_T} = V_T$.

Consider the grammar of class '2',

$$G_2 = \{V_{2_T},\ V_{2_N},\ P_2,\ \langle TWO \rangle\} \text{ with}$$

$$V_{2_T} = \{c,f,g,h\},\ V_{2_N} = \{\langle GC \rangle,\ \langle HA \rangle\}$$

P_2 :

```
1,2,3)   <TWO> ::= <GC> <HA> / gfh <GC> / ghf
  4,5)   <GC> ::= g / c
  6,7)   <HA> ::= h / a
```

We can illustrate the finite automaton recognizing the language of G_2 in Fig. 4.2.

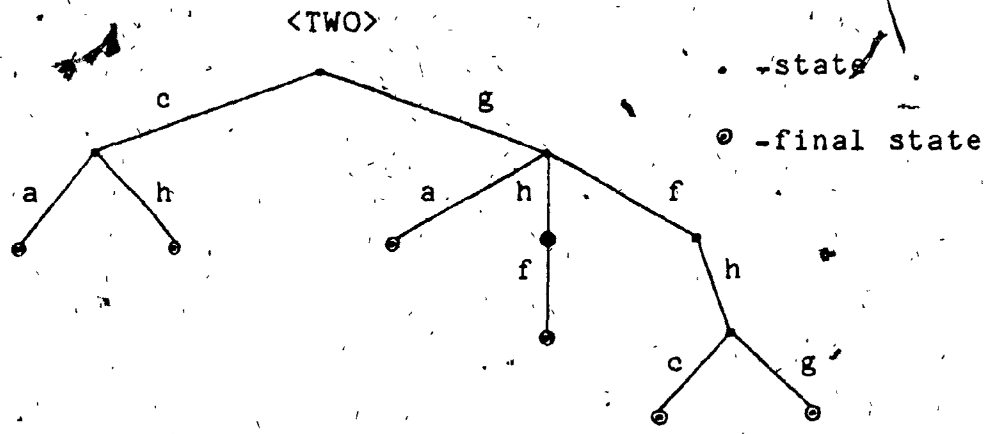


Figure 4.2. The Recognizer of Class '2'.

For example, we try to parse an unknown pattern shown in Fig.4.4 with its primitive string, x = gfhc to G_2, the following production rules will be applied :

```
        2)   <TWO> ----> gfh <GC>
        5)   <GC> ----> c .
```

In this case, x is successfully parsed according to G_2 and thus recognized as '2'.

Let us now consider the grammar of class '4',

$$G_4 = \{V_{4_T}, V_{4_N}, P_4, \langle FOUR\rangle\}$$

$$V_{4_T} = \{a,b,c,d,e,g,h,k,n\}$$

$$V_{4_N} = \{\langle GC\rangle, \langle HA\rangle, \langle DH\rangle, \langle GA\rangle, \langle EAB\rangle\}$$

P_4 :

```
       1)   <FOUR> ::= <GC> <HD> <HA> <EAB> /
       2)              <HA> <GC> <HD> <HA> <EAB> /
     3,4)              achh <GA> e / a <GC> k <EA> /
       5)              anc <HD> n /
     6,7)   <GC> ::= g / c
     8,9)   <HD> ::= h / d
   10,11)   <HA> ::= h / a
   12,13)   <GA> ::= g / a
   14,15)   <EA> ::= e / a
16,17,18)   <EAB> ::= e / a / b .
```

Fig. 4.3 illustrates the recognizer of this grammar.

Given a pattern shown in Fig.4.5 with its primitive string, y = agdae, it is successfully parsed according to G_4 by the following production rules :

```
        2)  <FOUR> -----> <HA> <GC> <HD> <HA> <EAB>
       11)  <HA> -----> a
        6)  <GC> -----> g
        9)  <HD> -----> d
       11)  <HA> -----> a
       16)  <EAB> -----> e .
```

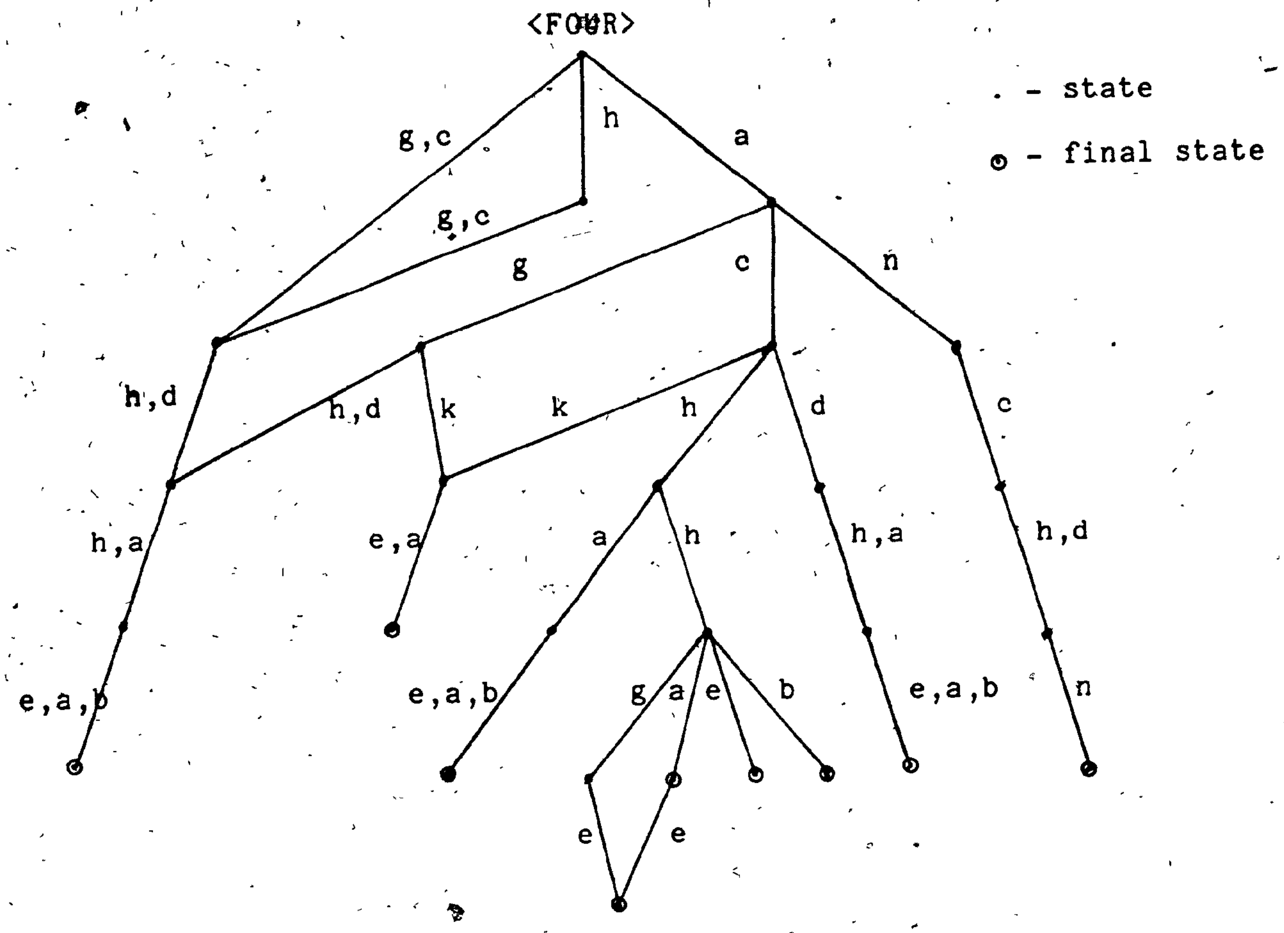
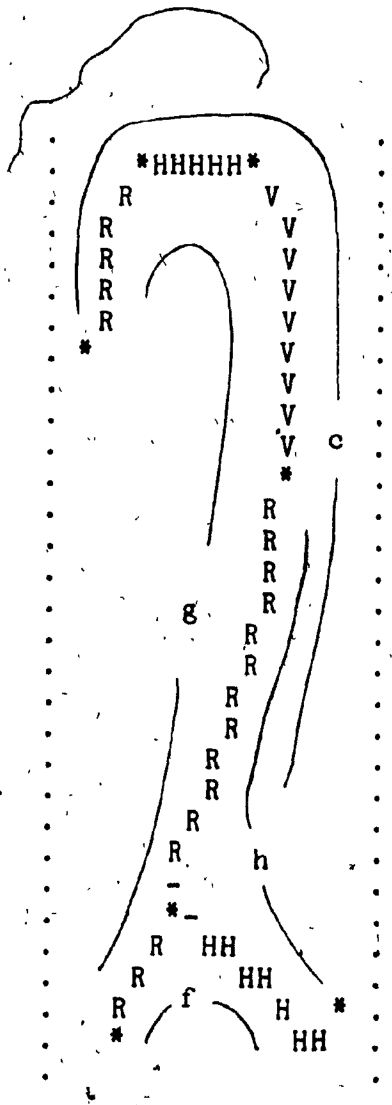


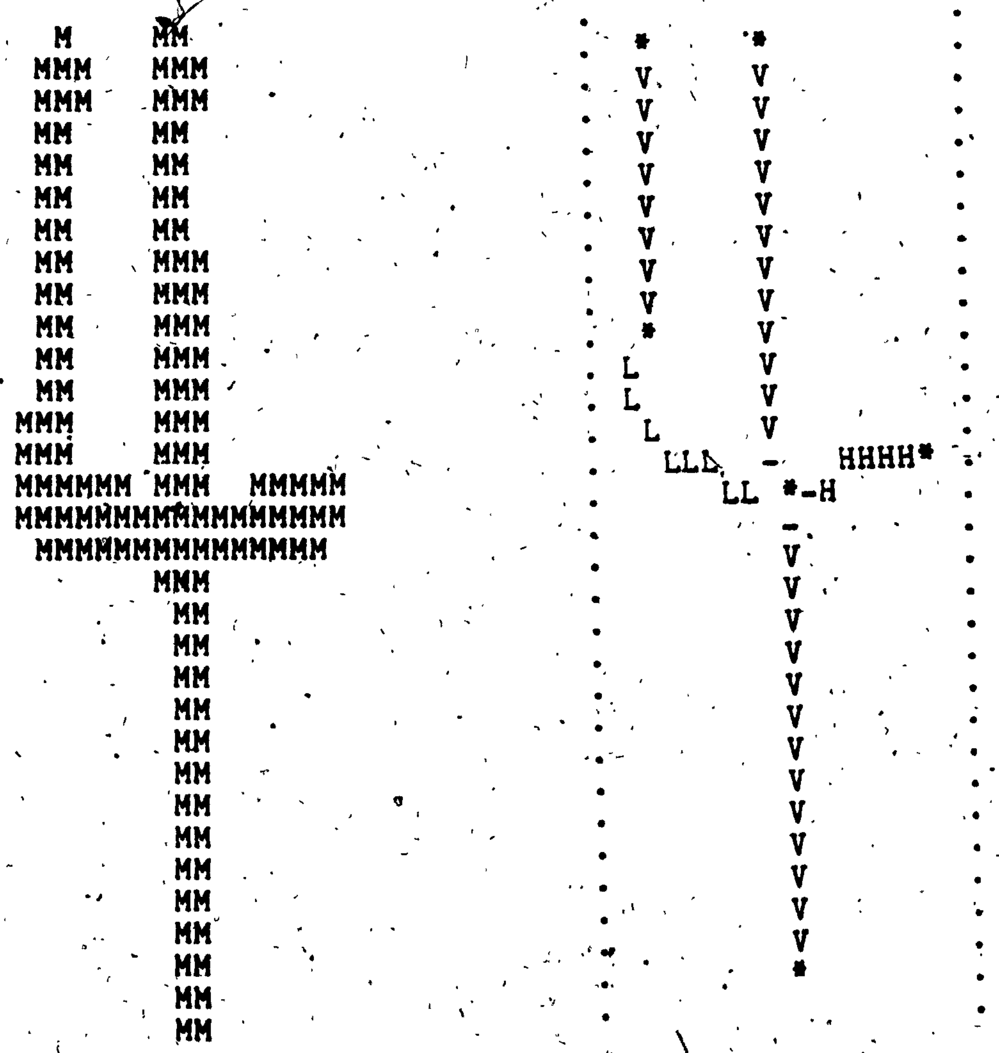
Figure 4.3. The Recognizer of Class '4'.

```
     MMMMMM
   MMMMMMMM
  MMMMMMMMMM
 MMMM     MMM
 MMM      MMM
 MMM       MM
 MMM       MM
MMMM       MM
MMMM       MM
MMM        MM
 MM        MM
           MM
           MM
          MMM
          MMM
          MMM
          MM
         MMM
         MMM
        MMM
        MMM
       MMM
      MMMM
      MMM
     MMM
    MMMM
    MMMMMMM
  MMMMMMMMMM
  MMMMM MMMMMM M
 MMMM     MMMMMM
 MMMM       MMMM
  MM         MMM
```



Primitive string = 'gfhc'

Figure 4.4. A Sample Numeral '2' for Parsing.



Primitive string = 'agdae'

Figure 4.5. A Sample Numeral '4' for Parsing.

The grammars of the rest of the classes are given in the Appendix.

4.3.2 Semantic Descriptive Schema

Since high-level primitives have been used in the description language to reduce the complexity of syntactic parsing, the evaluation of semantic information becomes important to resolve any ambiguity between grammars of two or more distinct character classes, In the classification scheme of this system, only the necessary synthetic attributes were evaluated for certain kinds of primitives. The reason for doing this is obvious, since evaluating values for attributes just sufficient for classification can reduce the computing time. Through experiments, the semantic information needed in the classification of numerals involves the properties of cavities. In particular, the most common cavities which lead to confusion are) (g) and ((h). There are mainly two types of attributes concerned, one is the depth of cavity and the other is the opening of the cavity. A summary of each semantic rule is given below.

(1) To distinguish a broken '0' from a '6' with an unclosed loop as shown in Fig.4.6 (a), an attribute indicating the opening (distance between the two end nodes) is simple and powerful.

Let us denote this attribute by O(h). Referring to section 3.4, we have

<ZERO> ::= h $\qquad\qquad$ $O(h) < t_r . T_R$

<SIX> ::= h $\qquad\qquad$ $O(h) \geq t_r . T_R$

where T_R is the total number of rows of the image matrix;

t_r is the threshold indicating the fraction of image length to discriminate the two classes.

(2) To distinguish between '2' and '7' both of which generate the same primitive string, 'gh', as shown in Fig.4.6(b), the depth of cavity of 'h' can serve as an attribute. When it is large enough, it favors class '2' rather than class '7' and vice versa.

Let us denote the attribute by D(h) and refer to section 3.4, we have

<TWO> ::= gh $\qquad\qquad$ $D(h) \geq t_c . T_c$

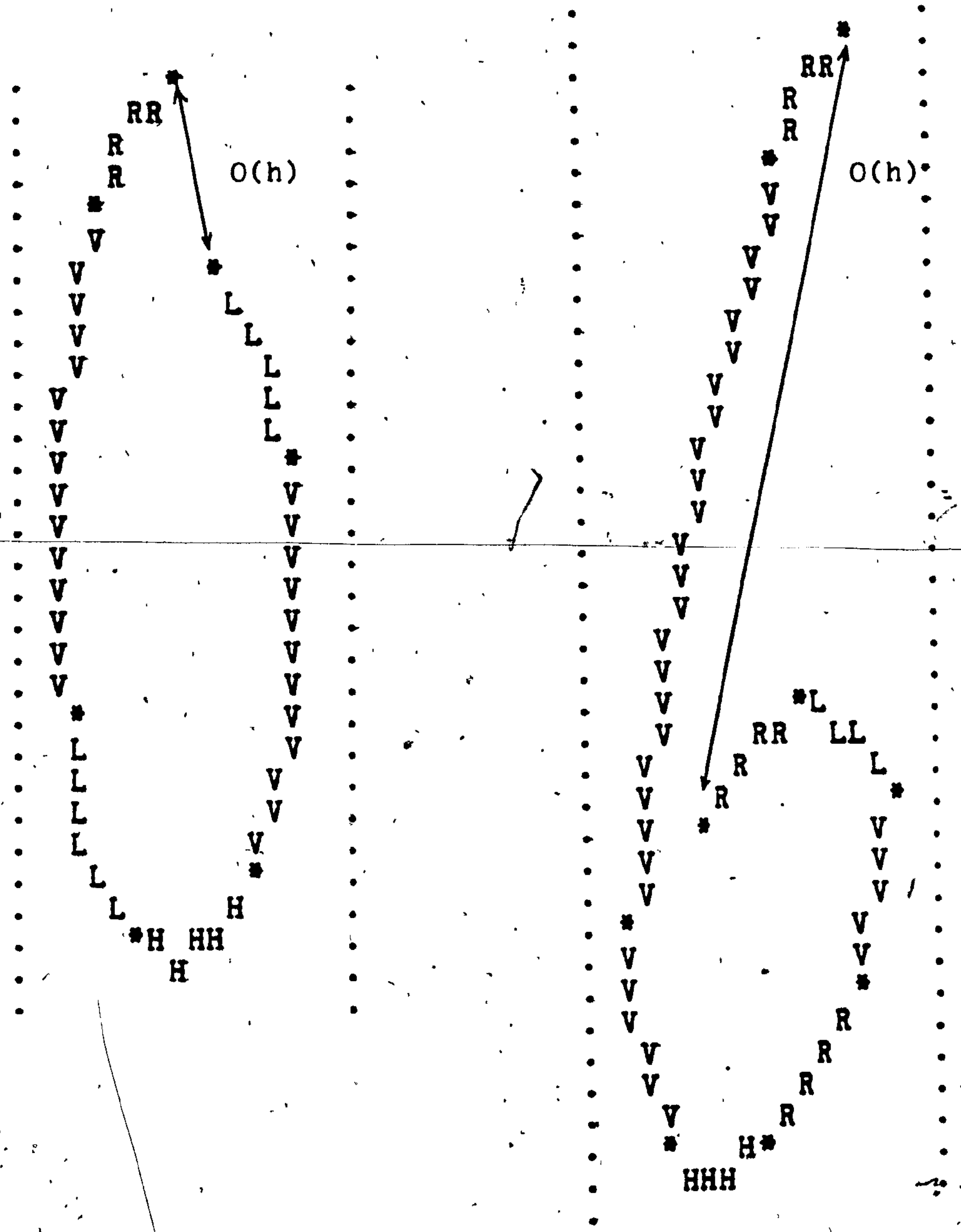
<SEVEN> ::= gh $\qquad\qquad$ $D(h) < t_c . T_c$

where T_c is the total number of columns of the image matrix;

t_c is the threshold indicating the fraction of image width.

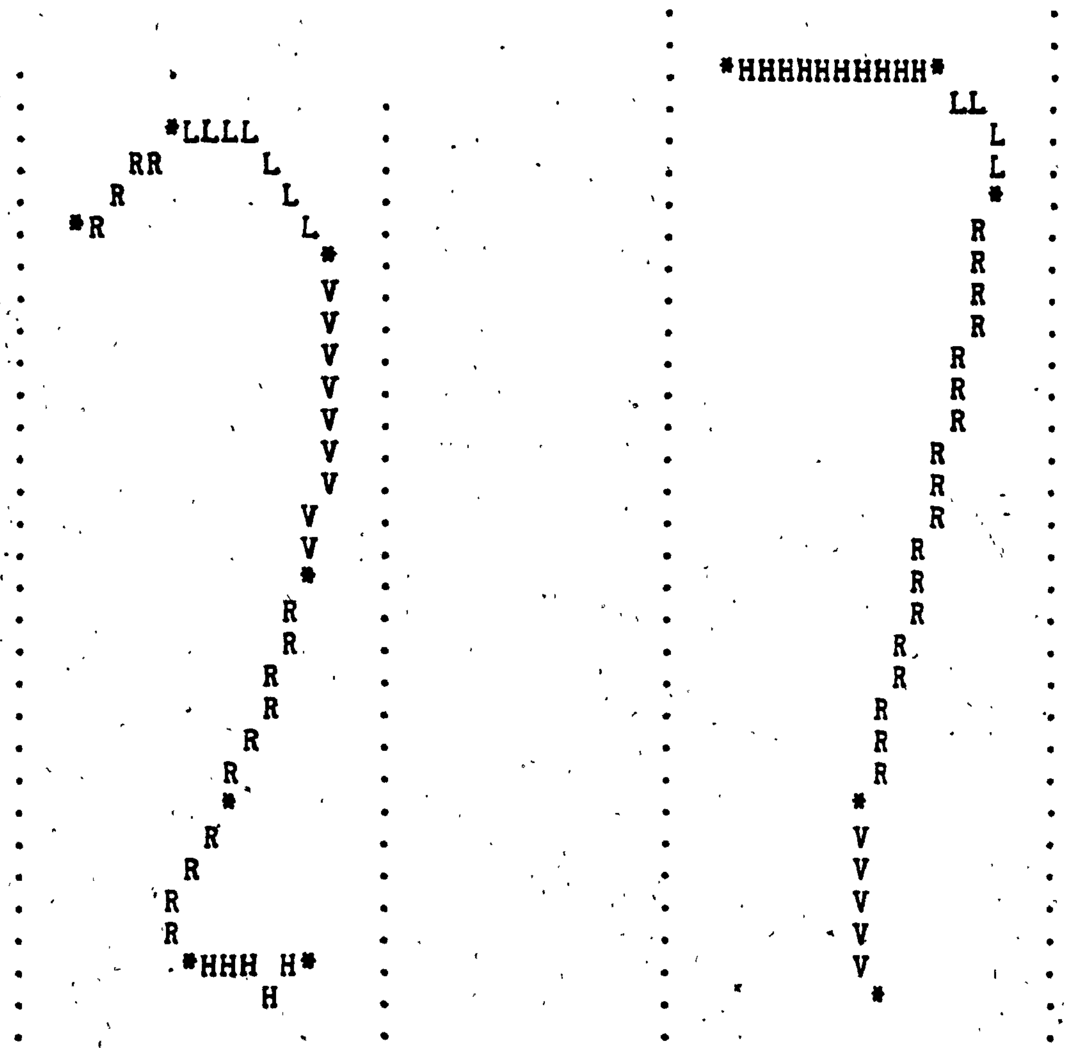
(3) In the case of class '1', some samples have a weak cavity and they may be confused with '6' or '7' as shown in Fig.4.6(c). So, the depth of cavity has to be considered. It should not be compared with the width of the character image, but rather a fixed value, because a small bent in '1' can still be as large as its width.

We have,



Primitive string = 'h'

Figure 4.6(a) Samples of Confused Numerals '0' and '6'.



Primitive string = 'gh'

Figure 4.6(b) Samples of Confused Numerals '2' and '7'.

<ONE> ::= g	$D(g) < F_c$
<SEVEN> ::= g	$D(g) \geq F_c$

and

<ONE> ::= h	$D(h) < F_c$
<SIX> ::= h	$D(h) \geq F_c$

where F_c is a threshold, which is assigned a fixed value, instead of being expressed as a fraction of the size of the image matrix.

(4) In the case of '3', the horizontal stroke of some samples is degenerated in the middle and it becomes '7' as shown in Fig.4.6(d). To detect the shape of the bottom part of the cavity is a means of distinguishing these two classes. The direction link to the last node of the primitive 'g', denoted by B, can be examined to discriminate the two classes. We have,

<THREE> ::= g	$B(g) \in \{ 2,3,4 \}$
<SEVEN> ::= g	$B(g) \in \{ 5,6 \}$.

4.6 Concluding Remarks

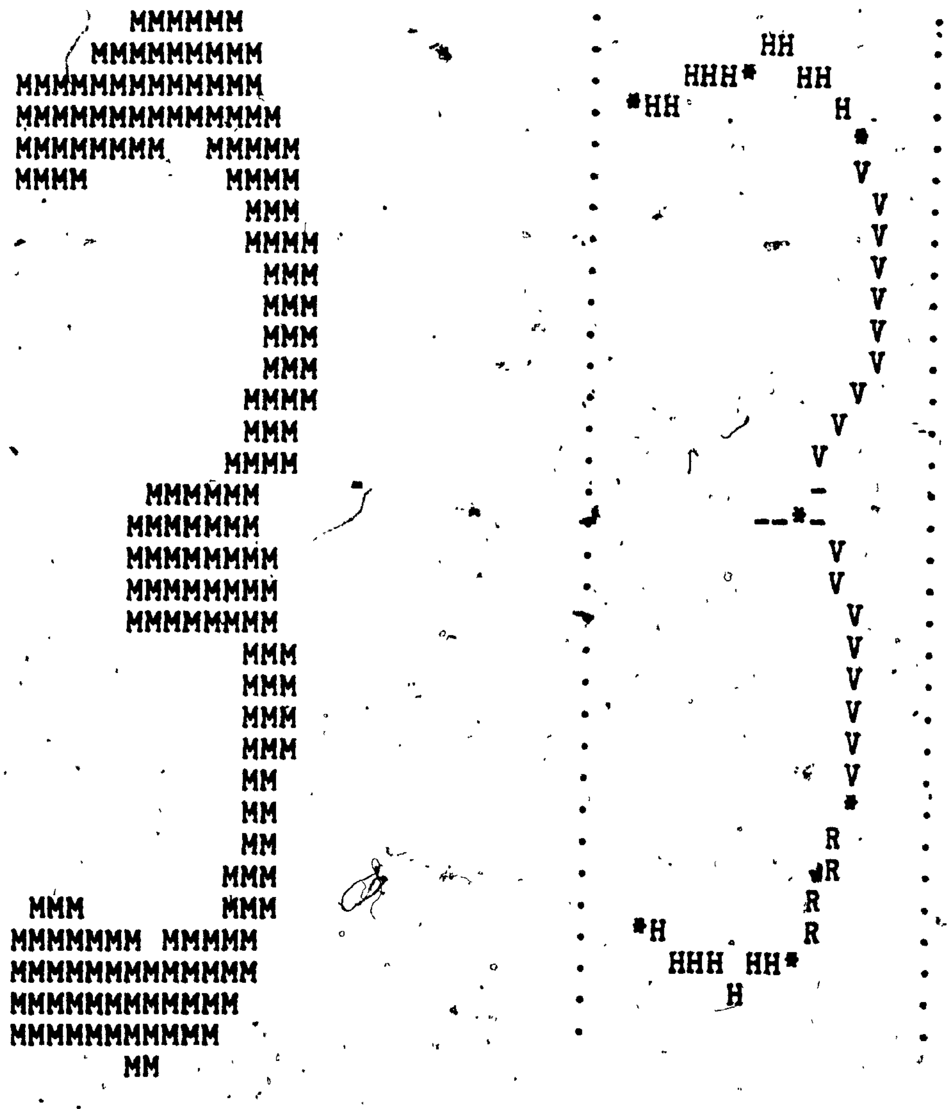
In this chapter, the foundation of formal language theory applied to syntactic recognition was reviewed. The introduction of semantic information in the recognition process also showed how they can compensate the weakness of the classical syntactic approach in handling significant numerical features. In the above classification scheme, the advantages of using a simple description language and a

```
   MM
MMMMMM
M  MMM         . HH    .
M  MM          .H   H  .
   MM          .*    * .
   MM          .     V .
   MM          .     V .
  MMM          .     V .
  MMM          .    V  .
   MM          .    V  .
  MMM          .     V .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MMM          .    V  .
  MM           .    *  .
```

```
  MMMM         .
 MMMMMM        .   RRR* .
 MM            .  R     .
 MM            .  R     .
 MM            .  R     .
 MM            .  R     .
MMM            . R      .
MMM            . *      .
MMM            .  V     .
 MM            .  V     .
 MM            .  V     .
 MM            .  V     .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            . V      .
MMM            .  *     .
               .        .
```

Primitive string = 'g'   Primitive string = 'h'

Figure 4.6(c) Samples Numeral '1' with weak cavity.



Primitive string = 'g'

Figure 4.6(d) Sample Numeral '3' with Degenerated Stroke.

small but sufficient set of attributes were pointed out. The principle of the classification scheme is to accept only those primitive strings which can be parsed by the grammars obtained from the training stage but reject all others. The training results on some experimental data and testing of the classification scheme will be discussed in the next chapter.

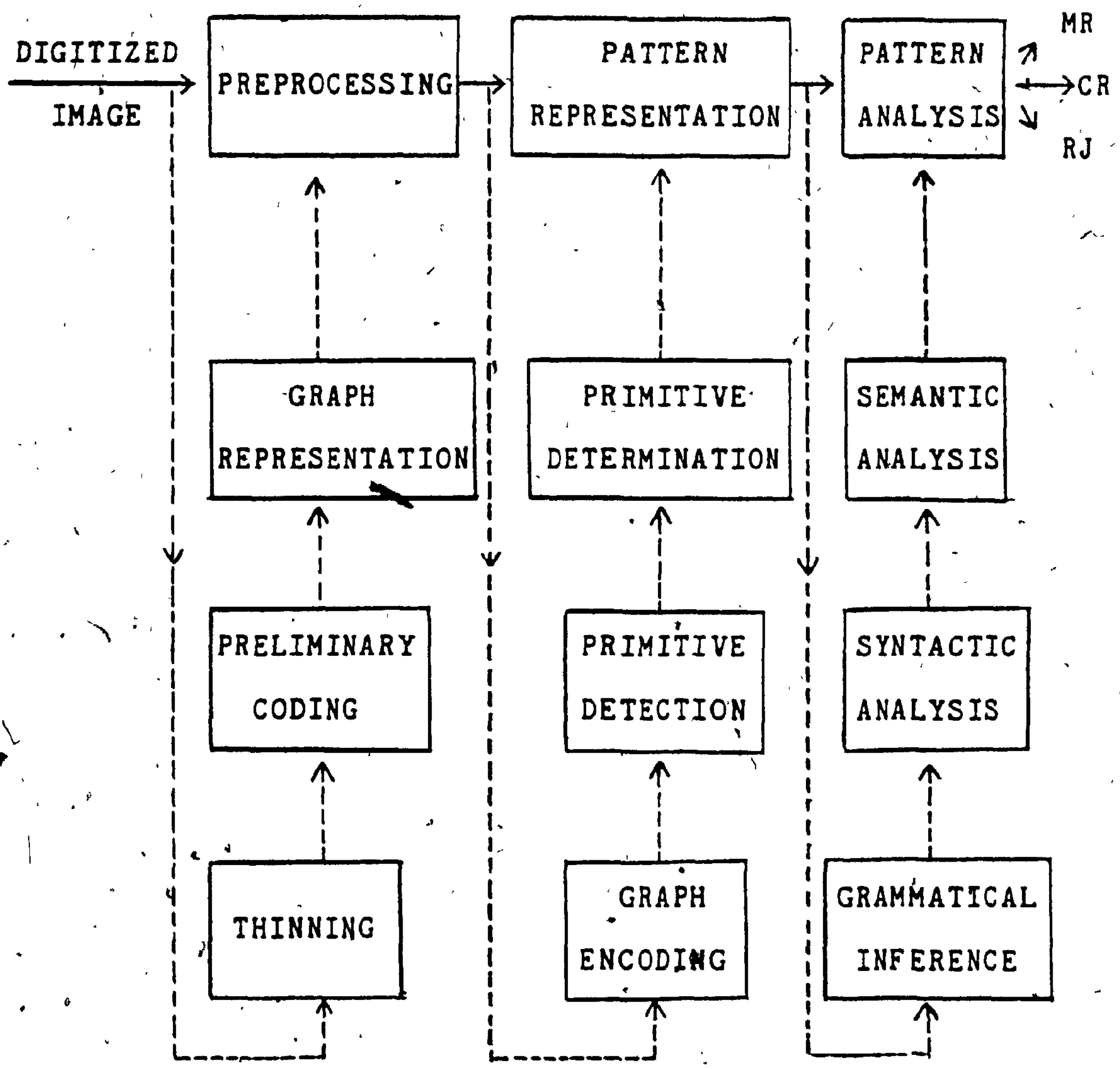
CHAPTER FIVE

EXPERIMENTAL RESULTS

5.1 Experimental Data and Environment

The algorithm for the extraction of primitives and syntactic-semantic classification scheme developed in Chapters 3 and 4 were tested on handwritten numerals. The detailed structure of the proposed character recognition system is shown in Fig.5.1. The extraction of primitives and classification scheme was implemented on the CDC CYBER 172 computer at Concordia University using the PASCAL language.

The data base for our investigation was prepared by Dr. C.Y. Suen of Concordia University. Suen's database consists of 100,000 alphanumeric characters including 174 models [30]. Among these characters, 6000 numerals with 600 samples per class of the ANSI shapes were used. The data samples were written by 30 different authors of which 15 were right-handed and 15 were left-handed with each author writing 20 specimens of each class. The subjects were given brief writing instructions on the size and models of the character and asked to write as quickly and carefully as possible. The frame of a digitized image is 64x32 and a typical image occupies 35 rows by 20 columns. Typical



CR - Correctly Recognized

MR - MisRecognized

RJ - ReJected

Figure.5.1. Block Diagram of the Structure of the Proposed Character Recognition System.

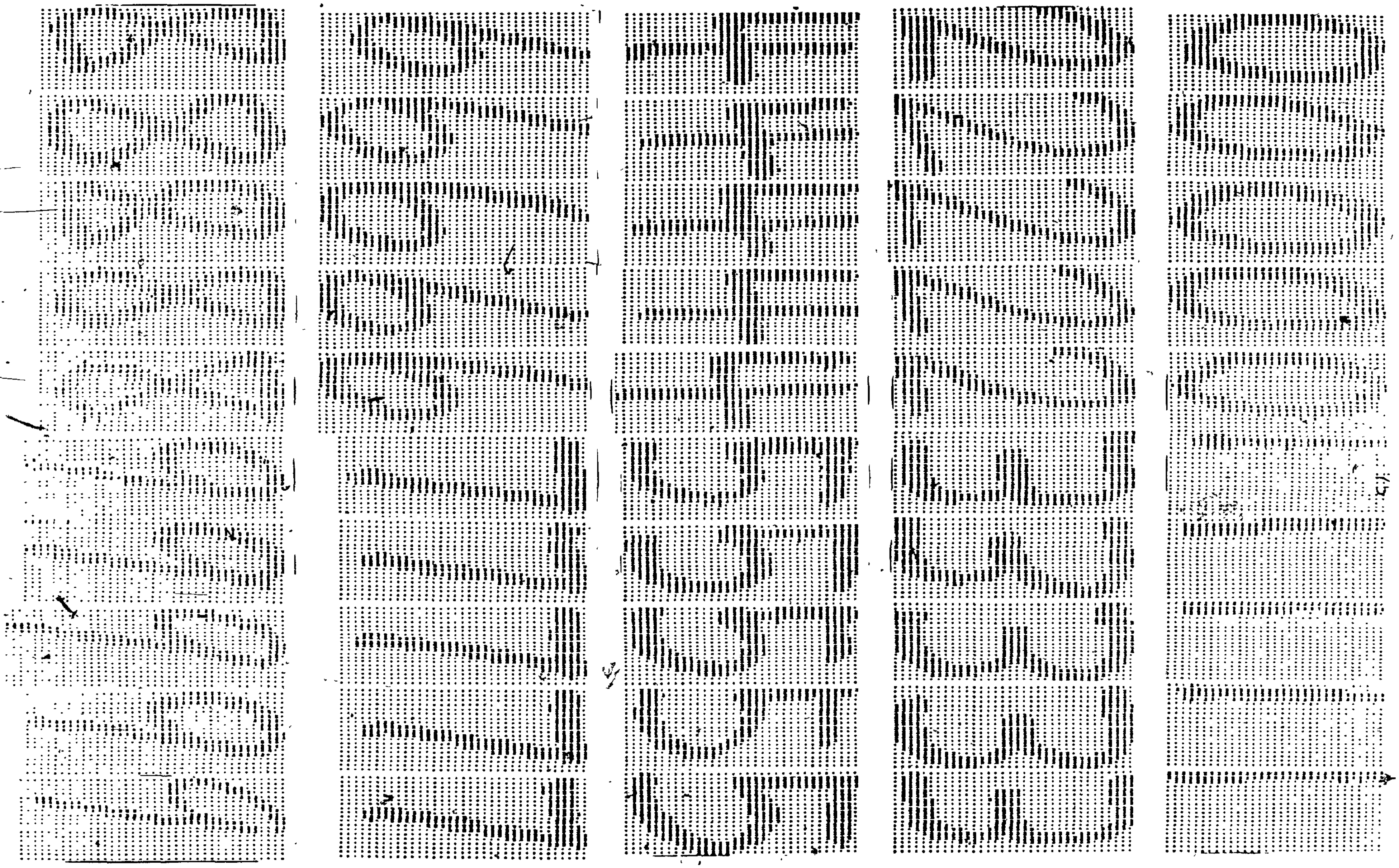


Figure 5.2. Typical Samples in the Experimental Data Base.

samples are shown in Fig.5.2.

The 6000 samples were divided into two sets arbitrarily with 300 samples per class. They are referred as Data Set II and Data Set III below. From Data Set II, 60 samples per class were taken out arbitrarily as the basic training set, which is referred as Data Set I. Three experiments were carried out on these three data sets.

5.2 Results of Recognition Experiments

Experiment I

Data Set I was used as the training data to obtain the grammar for each individual class. All the strings of primitives which define their corresponding character correctly were used in the grammatical inference. Two tests were performed with Data Set I.

In Test 1.1, only structural information was used in the grammar with no semantic information. A misrecognition rate of 0.67% was obtained. The confusion matrix of this test is shown in Table 5.1(a). The reason for the confusion can be referred to section 4.3.2 of the previous chapter which introduced the function of each type of semantic information.

In Test 1.2, semantic rules were introduced in the grammar based on the confusion obtained in Test 1. No sample was misrecognized but a 0.33% of rejection rate was recorded. The confusion matrix of this experiment is shown in Table

I\O	0	1	2	3	4	5	6	7	8	9	REJECT
0	59						1				
1		60									
2			60								
3				59				1			
4					59						1
5						60					
6							59				1
7			1					59			
8									60		
9							1			59	

Recognition Rate	= 99.00%	(594 / 600)
Misrecognition Rate	= 0.67%	(4 / 600)
Rejection Rate	= 0.33%	(2 / 600)

Table 5.1(a). Confusion matrix and experimental results obained from Data Set I (600 training samples) with no semantic rules.

I\O	0	1	2	3	4	5	6	7	8	9	REJECT
0	60										
1		60									
2			60								
3				60							
4					59						1
5						60					
6							59				1
7								60			
8									60		
9										60	

Recognition Rate	=	99.67%	(598 / 600)
Misrecognition Rate	=	0%	(0 / 600)
Rejection Rate	=	0.33%	(2 / 600)

Table 5.1(b). Confusion matrix and experimental results obtained from Data Set I (600 training samples).

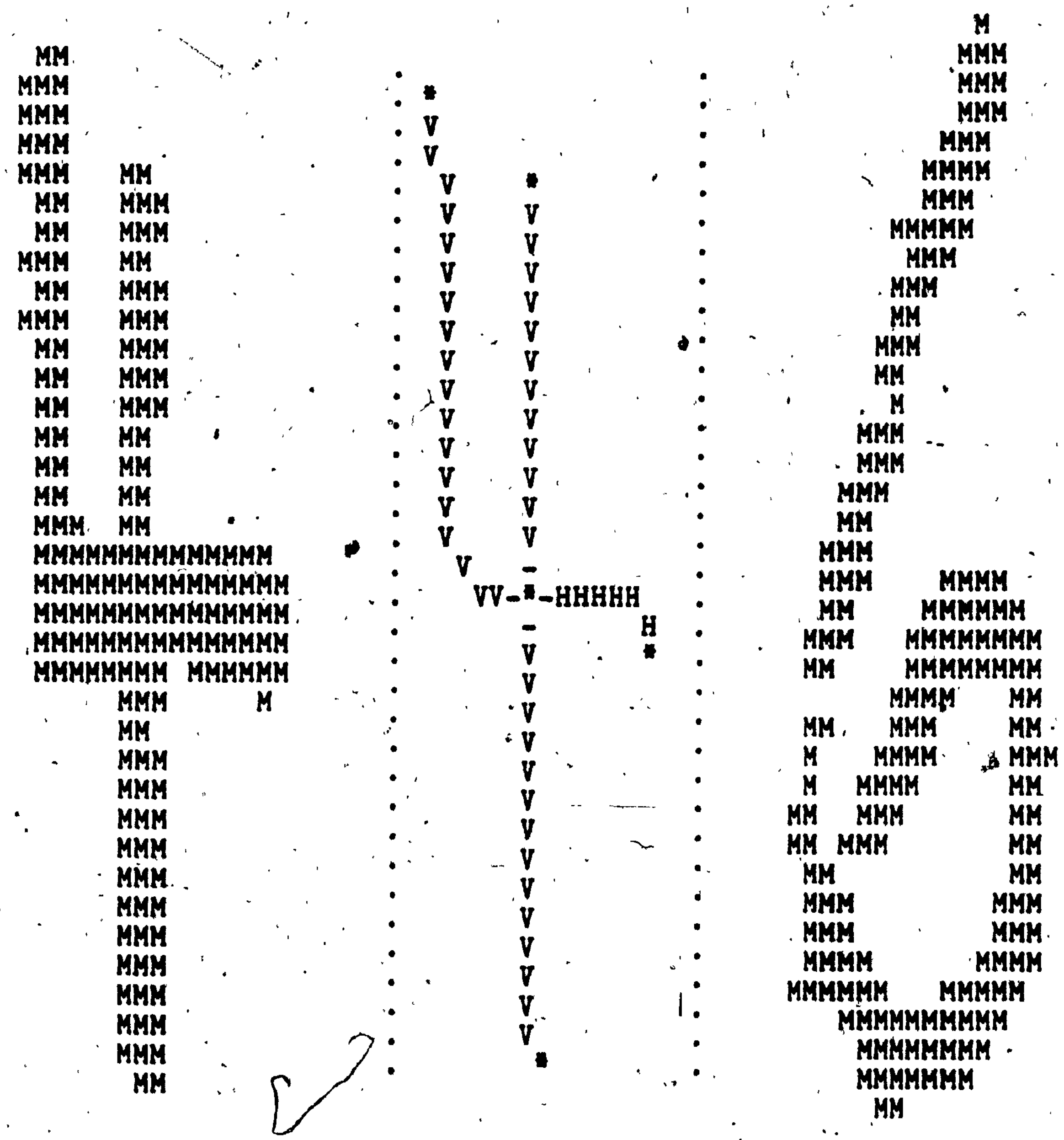


Figure 5.3. Rejected Samples from Data Set I.

5.1(b). The original and thinned images of the rejected samples are shown in Fig.5.3. The sample numeral '4' was rejected because the turning node and the junction node are too close to be detected. In order to maintain correct interpretation of a string of primitives to its corresponding character, our system rejected this sample though this kind of corrupted image also has a unique primitive string. The numeral '6' was rejected because it was broken. All numerals which have more than one connected components will be rejected.

Experiment II

Based on the grammars inferenced in Experiment I, two tests were performed on Data Set II.

In Test 2.1, the classifier developed in Experiment I with both syntactic and semantic analysis was tested on Data Set II. An error rate of only 0.03% was resulted. The misrecognized sample was an '8' with the upper loop degenerated into a line after the thinning process. Fig.5.4 illustrates the misrecognized sample. For the rejections, all of them were due to the lack of their prototypes in Data Set I. Table 5.2(a) illustrates the confusion matrix.

In Test 2.2, the samples rejected in Test 2.1 above due to the lack of prototypes were introduced into the classifier as new prototypes. This new classifier was then tested on Data Set II. The misrecognition rate remained 0.03% since

I\O	0	1	2	3	4	5	6	7	8	9	REJECT
0	294										6
1		300									
2			287								13
3				296							4
4					295						5
5						293					7
6							289				11
7								300			
8							1		283		16
9										288	12

Recognition Rate = 97.50% (2925 / 3000)

Misrecognition Rate = 0.03% (1 / 3000)

Rejection Rate = 2.47% (74 / 3000)

Table 5.2(a). Confusion matrix and experimental results obtained from Data Set II (600 training samples).

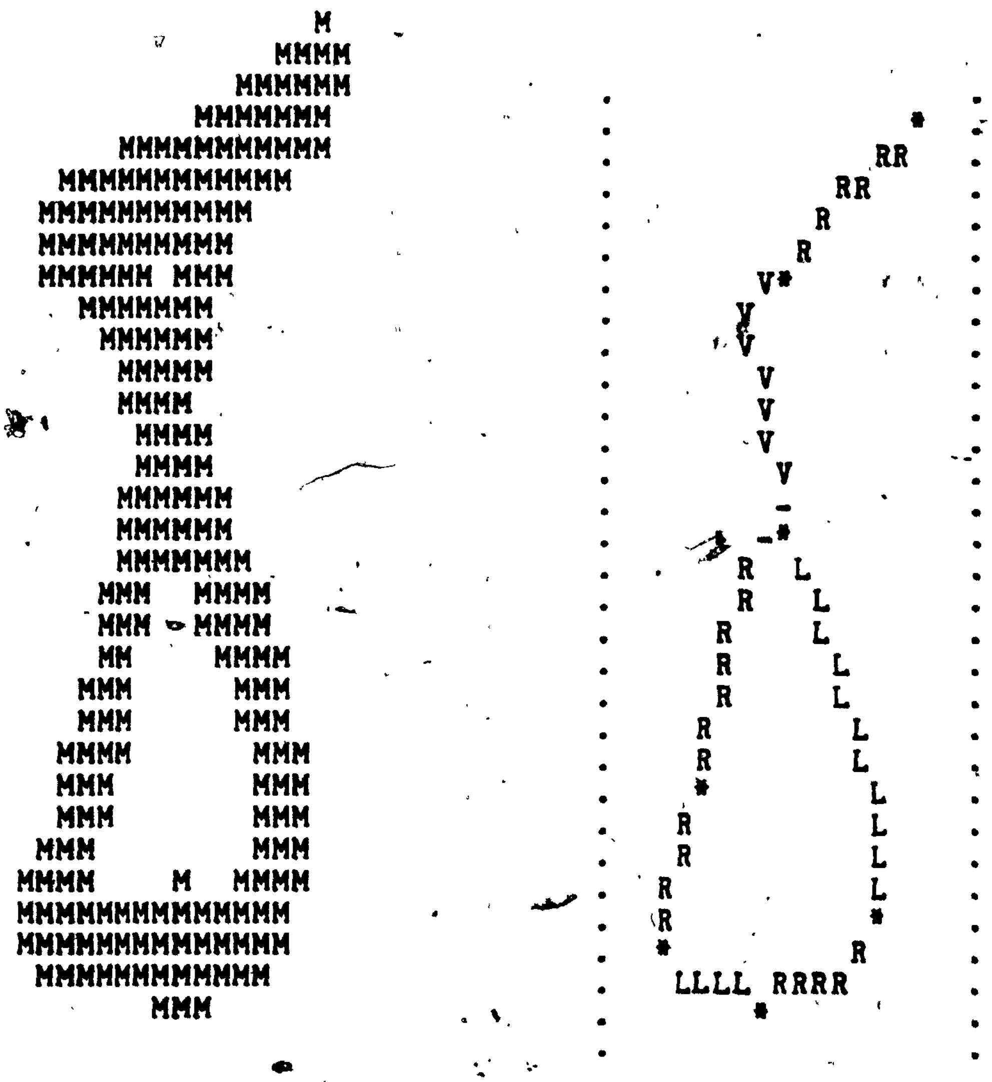


Figure 5.4. Misrecognized Sample from Data Set II.

I\O	0	1	2	3	4	5	6	7	8	9	REJECT
0	300										
1		300									
2			300								
3				300							
4					298						2
5						300					
6							295				5
7								300			
8							1		299		
9										299	1

Recognition Rate	=	99.70%	(2991 / 3000)
Misrecognition Rate	=	0.03%	(1 / 3000)
Rejection Rate	=	0.27%	(8 / 3000)

Table 5.2(b). Confusion matrix and experimental results obtained from Data Set II (training).

the numeral '8' cannot be improved. Meanwhile, a rejection rate of 0.27% was obtained. Table 5.2(b) shows the confusion matrix. The rejected '4's are similar to the case in Experiment I. Five samples of numeral '6' were rejected because they were all broken characters. The sample numeral '9' was rejected due to the corrupted quantization caused by the slant of the image character. Illustrations are given in Fig.5.5.

Experiment III

After revising the set of grammars trained from Data Set II in Experiment II, the classifier was tested on Data Set III to examine the result. A misrecognition rate of 0.07% was obtained, it was caused by the two '0's with a big opening to the right. Their images are shown in Fig.5.6. A rejection rate of 1.3% was obtained and the rejections of most samples were due to the lack of prototypes except those of '6's which were broken. Fig.5.7 shows some reasonable images which have been rejected due to the lack of prototypes in the classifier. In particular, the great variations of numeral '8' generated a lot of new prototypes not included in the grammars. The result of Data Set II and Data Set III using the former set as the taining set is shown in Table 5.3.

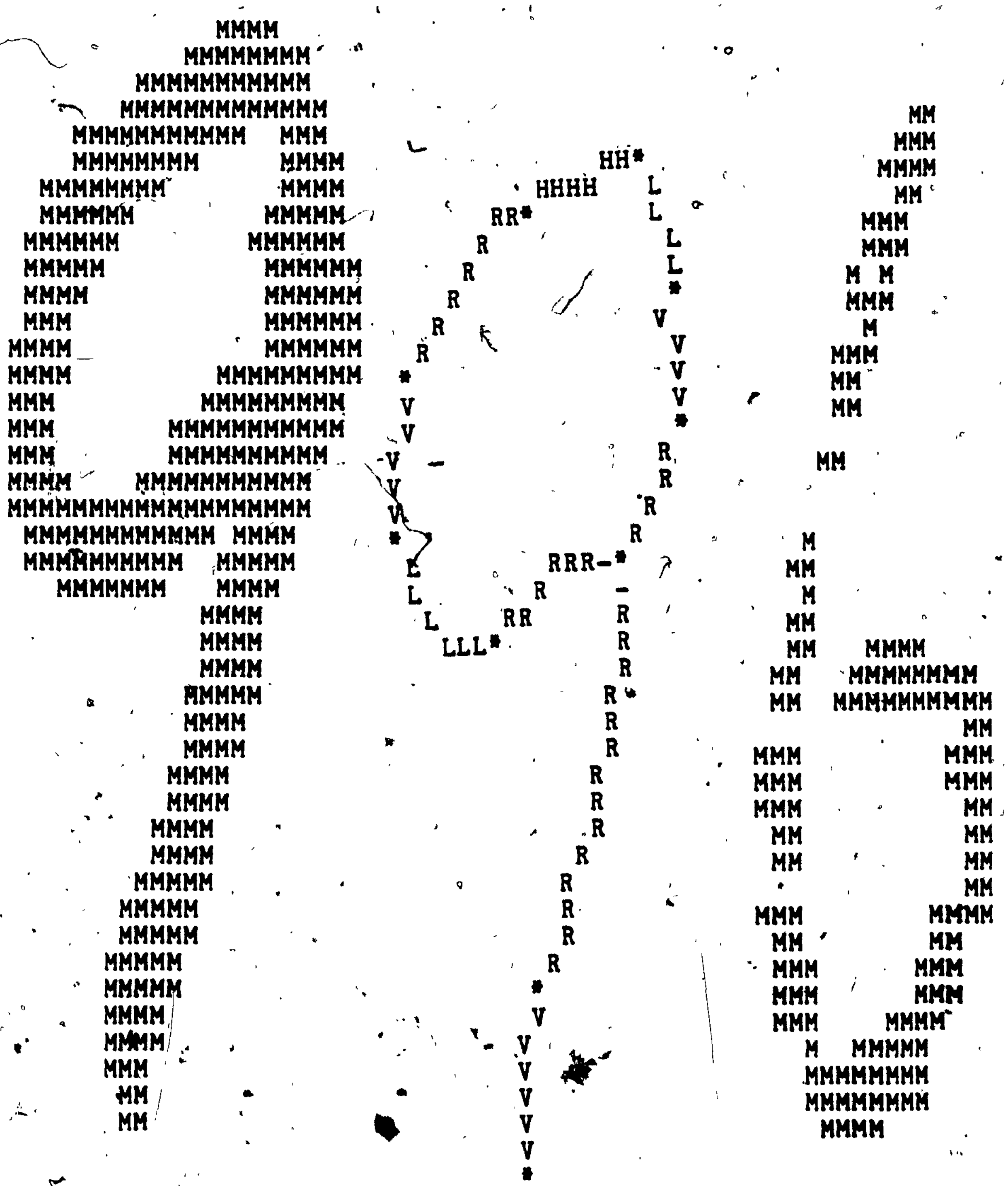


Figure 5.5. Rejected Sample from Data Set II.

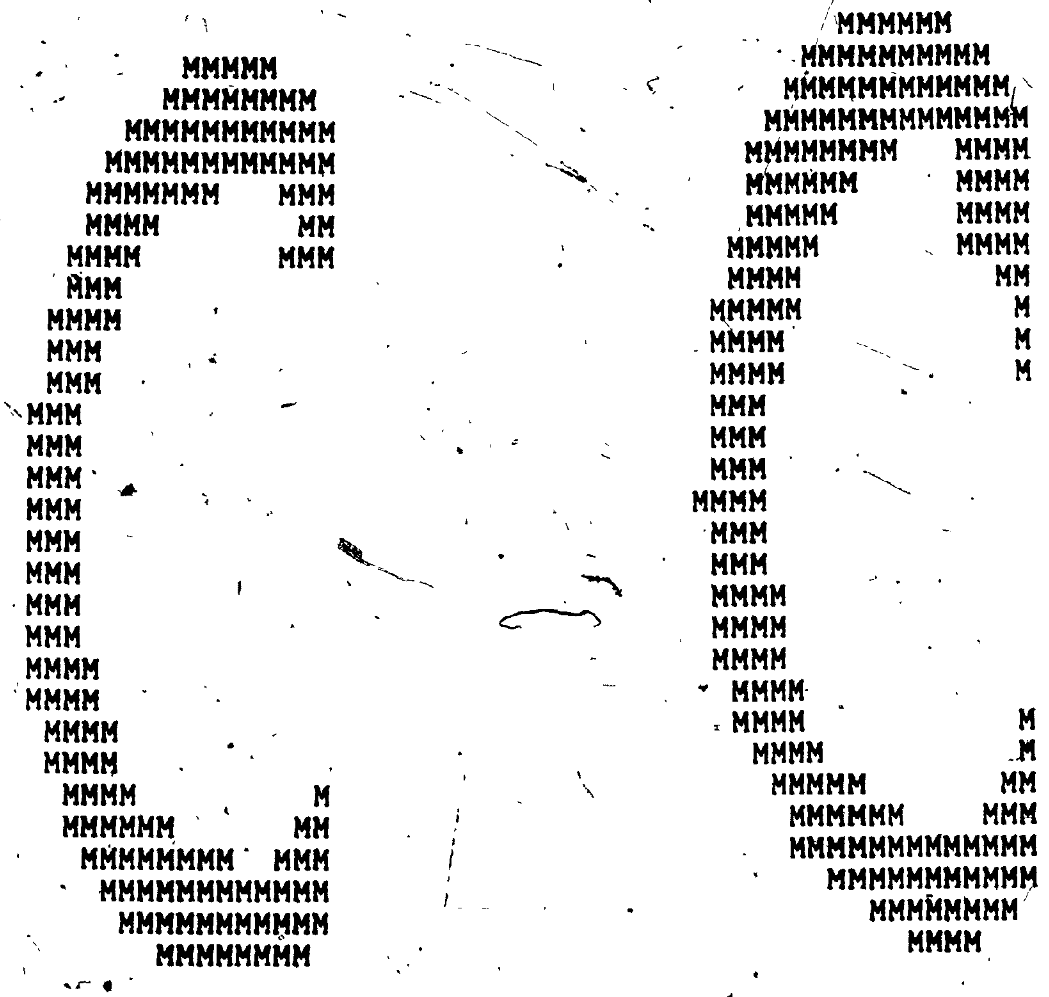


Figure 5.6. Misrecognized Samples from Data Set III.

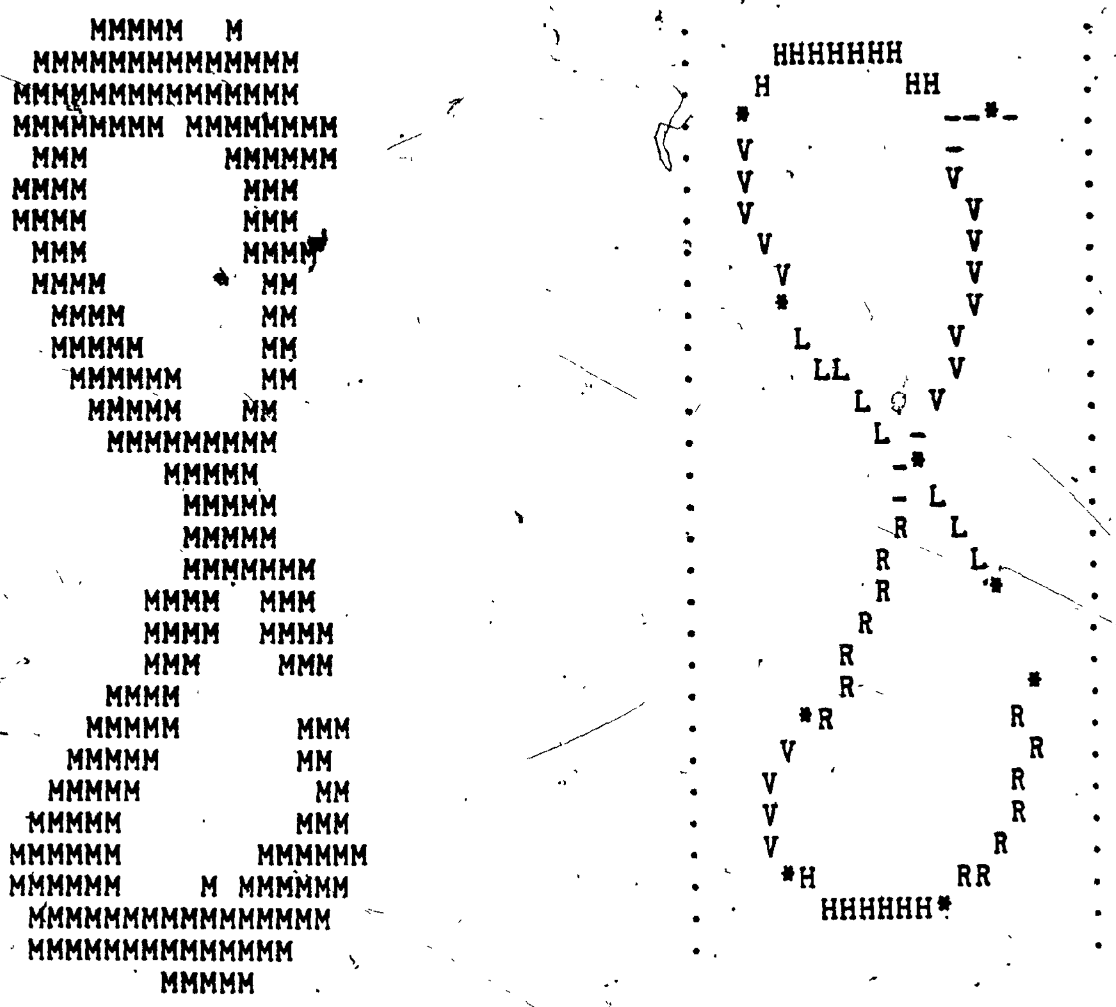


Figure 5.7(a) A Rejected Sample Numeral '8' from Data Set III.

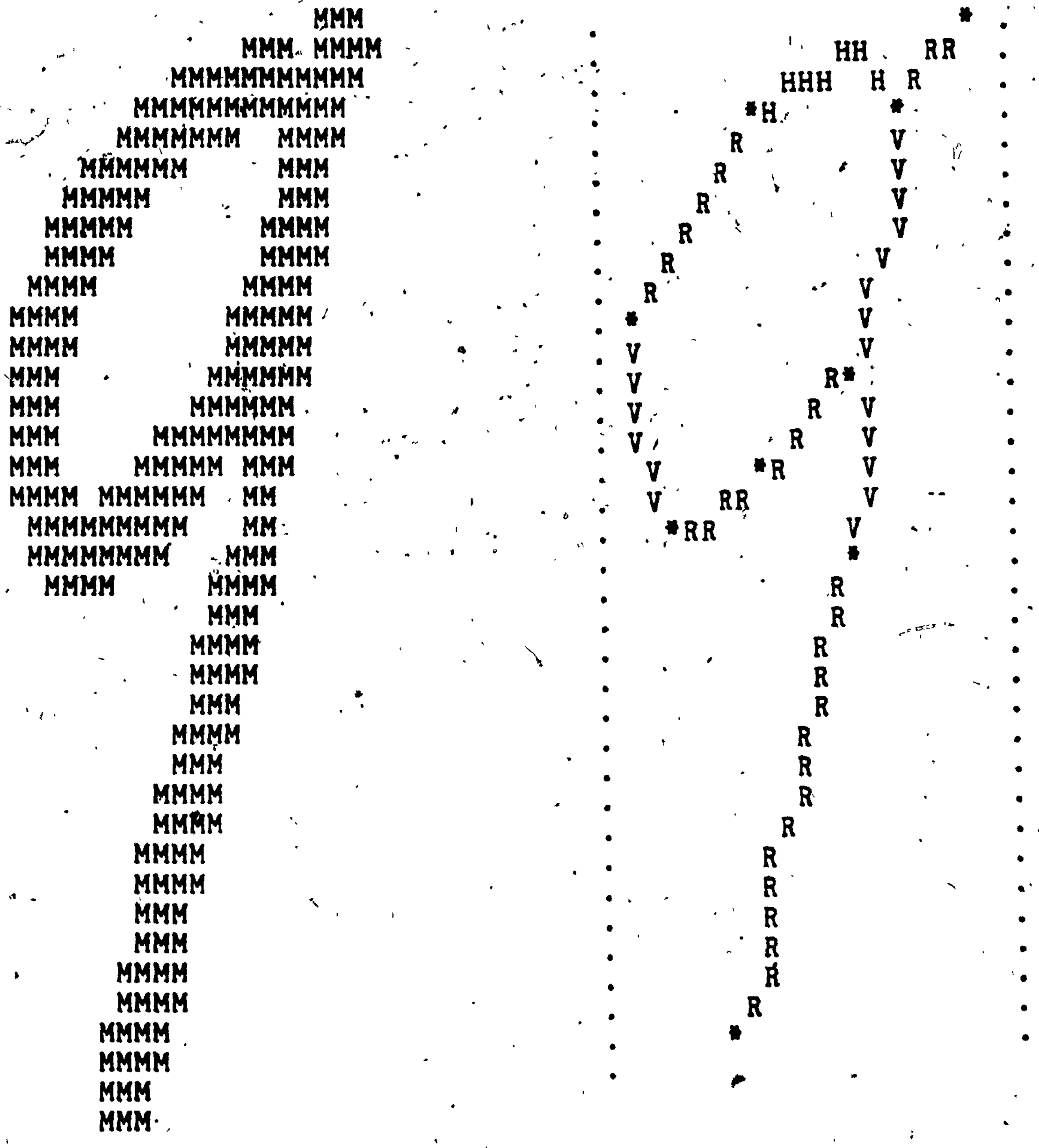


Figure 5.7(b) A Rejected Sample Numeral '9' from Data Set III.

I\O	0	1	2	3	4	5	6	7	8	9	REJECT
0	594						2				4
1		600									
2			600								
3				598							2
4					594						6
5						598					2
6							589				11
7								600			
8							1		581		18
9										595	5

Recognition Rate	= 99.15%	(5949 / 6000)
Misrecognition Rate	= 0.05%	(3 / 6000)
Rejection Rate	= 0.80%	(48 / 6000)

Table 5.3. Confusion matrix and overall experimental results obtained from Data Sets II and III.

The overall performance of the system is as follows :

Recognition Rate	=	99.15%
Misrecognition Rate	=	0.05%
Rejection Rate	=	0.80%

5.3 Concluding Remarks

The experimental results presented above support the approach used in this system. The speed of recognition is approximately 10 characters per second (including extraction of primitives and classification process). The most time-consuming process of the system is the thinning process which takes approimately 0.2 CPU second in average to thin a character. Though this reduces tremendously the speed of the whole system, it is considered as a less relevant problem to the using of topological features as pattern primitives since the hardware for thinning is available.

Though no complete recognition was obtained from the experiments, the performance of this recognition system compares favourably with those obtained by other researchers using the same or similar ANSI data [20,41]. Kwan et al [20] reported a higher misrecognition rate of 0.97% using geometrical features and discrimination functions. Using similar type of numerals, Mori et.al. [41] obtained a 80.3% of recognition with the correlation method. In particular, our system performed with a very low misrecognition rate due to the uniqueness of the grammars in each class. A slightly high rejection rate is observed mainly because of insufficient prototypes defined in the training process. With these characteristics, the system is more reliable because rejected samples can be re-examined and improvements or a secondary recognition process are allowed.

CHAPTER SIX

CONCLUSIONS

The basic objective of this work is to justify the use of geometrical structures as pattern primitives in a syntactic recognition system. These high-level primitives reduce the complexity of the syntactic recognizer, and motivate the use of attributed grammars. The design of the system is mainly based on the reduction of a character image to a graph with quantized vertices. A primitive string is established by the order in which the geometrical structures are extracted as pattern primitives. An automatic character recognition system has been implemented and tested on several data bases of handprinted numeric characters. The performance of the system has been established experimentally by the high recognition rates.

In the proposed system, there are some points needed to be improved and further developed. Some suggestions are given below.

1. In the classification scheme, a primitive string has to be in the defined set of prototypes in order to be recognized. This rigidity can be resolved by introducing substitution and deletion rules in the grammars or the use of stochastic grammars.

2. The idea employed in this system can be extended to recognize characters other than the numerics.
3. For those samples which were rejected because of broken strokes, each connectd part can be considered individually as a subgraph. By defining the relations between subgraphs in terms of their position, a meaningful string representation can be built up to characterize its character class.
4. For those samples which were rejected because of corrupted quantization, the eight directions convention can be further refined as sixteen directions to give more accurate graph representation.
5. The rejection of samples due to the lack of prototypes can be improved further by training the system with more characters.
6. When introducing more character models, more semantic information should be extracted if required.
7. In building the syntactic-semantic recognizer, the grammatical inference has not been formalized, so the necessity to develope efficient grammar and parser generators to facilitate more complex syntactic analysis is urgent.
8. During the process of extraction of primitives, the adjacency matrix of the graph form allows the reconstruction of the character. This reconstruction can be used in graphic display and data communication.
9. The implementation of this system in primitive

extraction and parsing can be refined to augment the efficiency of the system.

10. In the system, time is mainly consumed in the thinning process, parallel thinning synchronized with the extractor of primitives and recognizer can speed up the system considerably.

REFERENCES

1. Ali, F. and Pavlidis, T., "Syntactic Recognition of Handwritten Numerals", IEEE Trans. SMC. Vol.7, pp.537-541, July 1977.
2. Alt, F.L., "Digital Pattern Recognition by Moments" in Optical Character Recognition, Ed. by Fischer, G.L. and Pollock, D.K. et al., McGregor & Werner, Washington D.C., pp.153-180, 1962.
3. Beun, M., "A Flexible Method for Automatic Reading of Handwritten Numerals", Philips Tech. Rev., vol.33, pp.89-101,130-137, 1973.
4. Caskey, D.L. and Coates, C.L., "Machine Recognition of Handprinted Characters", Proc. 1st Int. Jt. Conf. Pattern Recognition, pp.41-49, 1973.
5. Doyle, W., "Recognition of Sloppy, Hand-printed Characters", Proc. West. Jt. Computer Conf., vol.17, pp.133-142, May 1960.
6. Fedar, J., "Plex Languages", Information Science 3, pp.225-241, 1971.
7. Focht, L.R. and Burger, A., "A Numeric Script Recognition Processor for Postal Zip Code Application", Proc. Int. Conf. Cybernetics and Society, pp.489-492, Nov. 1976.
8. Freemen, H., "On the Encoding of Arbitrary Geometric Configurations", IRE Trans. on Electronic Computers,

pp.260-268, June 1961.

9. Fu, K.S., Syntactic Pattern Recognition and Applications. Prentice-Hall, Englewood Cliffs, N.J., 1982.

10. Fu, K.S. and Lu, S.Y., "A Clustering Procedure for Syntactic Pattern", IEEE Trans. SMC. Vol.SMC-7, pp.734-742, Oct. 1977.

11. Genchi, H. and Mori, K.I. et al., "Recognition of Handwritten Numeral Characters for Automatic Letter Sorting", Proc. IEEE vol.5-6, pp.1292-1301, Aug. 1968.

12. Genchi, H. and Tamada, M. et al., "Automatic Reader-Sorter for Mail with Handwritten or Printed Postal Code Numbers", Toshiba Rev., pp.7-11, July-Aug. 1970.

13. Gonzalez, R.C. and Thomason, M.G., Syntactic Pattern Recognition- An Introduction. Addison-Wesley, Reading, Mass., 1978.

14. Gonzalez, R.C. and Wintz, P., Digital Image Processing. Addison-Wesley Pub., Reading, Mass., 1977.

15. Haralick, R.M. et al., "Texture Features for Image Classification", Int. Jt. of Computer and Info. Sci., vol.9, pp.1-13, 1980.

16. Hilditch, C.J., "Linear Skeletons from Square Cupboards", Machine Intelligence, vol. 4, pp.403-422, American Elsevier, NY, 1969.

17. Hopcroft, J.E. and Ullman, J.D., Formal Languages and their Relation to Automata. Addison-Wesley Pub.,

Reading, Mass., 1969.

18. Hu, C.H. et al., "A Handwritten Numeral Recognition Machine for Automatic Mail-Sorting", Proc. 1st European Conf. Signal Processing, pp.507-510, Sept. 1980.

19. Knuth, D.E., "Semantic of Context-Free Languages", J. Math. Syst. Theory 2., pp.127-146, 1968.

20. Kwan, C.C., Pang, L. and Suen, C.Y., "A Comparative Study of some Recognition Algorithms in Character Recognition", Proc. Int. Conf. on Cybernatics and Society, pp.530-535, 1979.

21. Lewis II, R.M., Rosenkrantz, D.H. and Stearns, R.E., Computer Design Theory. Addison-Wesley, Reading, Mass., 1976.

22. Ma, H. and Yudin, J.,"Skeletonization of Pattern", Term Report, Dept. Of Computer Science, Concordia University, Aug. 1979. (Available from Prof. C.Y. Suen)

23. Narasimhan, R., "On the Description, Generation, and Recognition of Classes of Pictures" in Automatic Interpretation and Classification of Images, Ed. by Grasselli, A., Acadamic Press, NY, 1969.

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

25. Ogawa, H. and Taniguchi, K., "Preprocessing for Chinese Character Recognition and Global Classification of Handwritten Chinese Characters", Pattern Recognition, vol.11, pp.1-7, 1979.

26. Pavlidis, T., "Syntactic Feature Extraction for Shape Recognition", Proc. 3rd Int. Jt. Conf. on Pattern Recognition, pp.95-99, 1976.

27. Persoon, E. and Fu, K.S., "Shape Discrimination using Fourier Descriptors", IEEE Trans. Syst., Man, Cybern., vol.7, pp.170-179, Mar. 1977.

28. Shaw, A., "A Formal Picture Description Scheme as a Basis for Picture Processing System", Information and Control 14, pp.9-52, 1963.

29. Sherman, H., "A Quasi-Topological Method for the Recognition of Line Patterns", Proc. Int. Conf. on Information Processing, pp.232-238, 1959.

30. Shinghal, R. and Suen, C.Y., "A Method for Selecting Constrained Hand-printed Character Shapes for Machine Recognition", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.PAMI-4, No.1, pp.74-78, Jan. 1982.

31. Spanjersberg, A.A.,"Experiments with Automatic Input of Handwritten Numeral Data into a Large Administrative System", IEEE Trans. Syst., Man, Cybern., vol.8, pp.286-288, April 1978.

32. Stallings, W., "Recognition of Printed Chinese Characters by Automatic Pattern Analysis", Computer Graphics and Image Processing, vol.1, pp.47-65, 1972.

33. Suen, C.Y., "Recent Advances in Computer Vision and Computer Aids for the Visually-Handicapped", IEEE, Computers in Opthalmology, pp.259-266, April 1979.

34. Suen, C.Y., Berthod, M. and Mori, S., "Automatic Recognition of Handprinted Characters - The State of the Art", Proc. IEEE, vol.68, No.4, pp.469-487, April 1980.

35. Tou, J.T., "An Approach to Understanding Geometical Configuration by Computer", Int. Jt. of Computer and Infor. Sci., vol.9, No.1, pp.1-13, 1980.

36. Tou, J.T. and Gonzalez, R.C., Pattern Recognition Principles. Addison-Wesley, Reading, Mass., 1974.

37. Tou, J.T. and Gonzalez, R.C., "Recongnition of Handwritten Characters by Topological Feature Extraction and Multilevel Categorization", IEEE Trans. Comput., pp.776-785, July 1972.

38. Tucker, N.D. and Evans, F.C., "A Two-Step Strategy for Character Recognition Using Geometrical Moments", IEEE 2nd Int. Jt.. Conf. on Pattern Recognition, pp.223-225, 1974.

39. Unger, S.H., "Pattern Detection and Recognition", Proc. Of IRE, pp.1737-1752, Oct. 1959.

40. Watt, A.H. and Beurle, R.L., "Recognition of Hand-printed Numerals Reduced to Graph-Representable Form", Proc. 2nd Int. Jt. Conf. Artificial Intelligence, pp.322-332, Sept. 1971.

41. Yamada, H., Saito, T. and Mori, S., "An Improvement of Correlation Method-Locally Maximized Correlation", Trans. IECE, No.10, pp.970-973, 1981.

APPENDIX

List of Thresholds

			VALUE
L_t	:	the normal length of a straight line segment	: 6 pixels
M_l	:	the minimum length of a straight Line segment	: 5 pixels
ϕ_t	:	the angular difference between two segments to determine the corner node	: 22.5°
θ_t	:	the quantization angle of graph vertices	: 20°
θ_f	:	the quantization angle for primitive determination.	: 20°
t_r	:	the fraction of image length to measure the opening of a cavity.	: 1/3
t_c	:	the fraction of image width to measure the depth of a cavity.	: 1/4
F_c	:	the minimum depth to support a cavity	: 3 pixels

Attributed Grammars trained from Data Sets I & II

NOTE. Except the starting symbols, the name of a non-terminal indicates the alternatives of the primitives from it.

Example : <GC> -----> g / c

<ZERO>	::=	i /	
		h /	$O(h) < t_r.T_R$
		gb /	
		a /	
		ii /	
		gfi	
<ONE>	::=	k /	
		g /	$D(g) < F_c$
		h	$D(h) < F_c$
<TWO>	::=	<GC> <HA> /	$D(h) \geq t_c.T_c$
		gfh <GC> /	
		ghf	
<THREE>	::=	<GC> <GB> <GB> <HAD> <GC> /	
		<GC> <CB> g /	
		<GC> <HD> <GB> /	
		g	$4 \leq B(g) \leq 2$
<FOUR>	::=	a <GC> k<EA> /	
		achh <GA> e /	
		<GC> <HD> <HA> <EAB> /	
		<HA> <GC> <HD> <HA> <EAB> /	
		anc <HD> n	
<FIVE>	::=	<HD> <GBC> /	
		dgg <HD> /	
		dg <GC> g <HD> /	
		<GC> agg <HD> <EJ> /	
		ggdd /	
		edgghe /	

eag /
agg <HD> e

<SIX> ::= <HD> <HA> o /
<HA> <GB> oho /
hg <DF> <GO> ho /
g <FC> <GB> ho /
<HD> o /
hod /
h $D(h) \geq F_c$; $O(h) \geq t_r \cdot T_R$

<SEVEN> ::= g / $4 < B(g) < 7$
c /
gh / $D(h) < t_c \cdot T_c$
g <HD> <JE> /
eg

<EIGHT> ::= n <HA> g <AO> <HA> <GC> o /
ng <AO> <HA> <GC> o /
nhgohgoo /
hgoh <GC> no /
agoh <GC> no /
goh <GC> no /
edgoh <HAN> o /
dgoh <HAN> o /
dgohno /
<HA> gan /
dhno /
ddgoho /
cdahgohoo

<NINE> ::= <HD> hn /
hhgn /
hg <KH> <HANE> /
hd <GC> <HD> /
dghn /
ddghg /
ndghgn /
g <GC> /
nnkk

where T_R is the total number of rows of the image matrix;
T_C is the total number of columns of the image matrix.