Communication and Cybernetics 14

Syntactic Pattern Recognition, Applications

Edited by K. S. Fu

Springer-Verlag Berlin Heidelberg New York

Communication and Cybernetics 14

Editors: K. S. Fu W. D. Keidel H. Wolter

Syntactic Pattern Recognition, Applications

Edited by K. S. Fu

With Contributions by

J. E. Albus R. H. Anderson J. M. Brayer R. DeMori
H.-Y. F. Feng K. S. Fu S. L. Horowitz B. Moayer
T. Pavlidis W. W. Stallings P. H. Swain T. Vámos

With 135 Figures

Springer-Verlag
Berlin Heidelberg New York 1977

Professor KING SUN FU, PhD

Purdue University, School of Electrical Engineering
West Lafayette, IN 47907, USA

Professor Dr. WOLF DIETER KEIDEL

I. Physiologisches Institut der Universität Erlangen-Nürnberg
D-8520 Erlangen, Fed. Rep. of Germany

Professor Dr. HANS WOLTER

Institut für Angewandte Physik der Universität,
D-3550 Marburg/Lahn, Fed. Rep. of Germany

ISBN-13: 978-3-642-66440-3 e-ISBN-13: 978-3-642-66438-0
DOI: 10.1007/978-3-642-66438-0

Library of Congress Cataloging in Publication Data. Main entry under title: Syntactic pattern recognition. (Communication and cybernetics; 14). Bibliography: p. . Includes index. 1. Pattern perception-Addresses, essays, lectures. 2. Pattern recognition systems-Addresses, essays, lectures. I. Albus, J. E. II. Fu, King Sun, 1930—. Q327.S93. 001.53'4. 76-42196. ISBN 0-387-07841-X

This work is subject to copyright. All rights are reserved, whether the whole or part of material is concerned, specifically those of translation, reprinting, re-use of illustrations. broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© by Springer-Verlag Berlin Heidelberg 1977
Softcover reprint of the hardcover 1st edition 1977

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Monophoto typesetting, offset printing, and book binding: Brühlsche Universitätsdruckerei, Giessen

Preface

The many different mathematical techniques used to solve pattern recognition problems may be grouped into two general approaches: the decision-theoretic (or discriminant) approach and the syntactic (or structural) approach. In the decision-theoretic approach, a set of characteristic measurements, called features, are extracted from the patterns. Each pattern is represented by a feature vector, and the recognition of each pattern is usually made by partitioning the feature space. Applications of decision-theoretic approach include character recognition, medical diagnosis, remote sensing, reliability and socio-economics. A relatively new approach is the syntactic approach. In the syntactic approach, each pattern is expressed in terms of a composition of its components. The recognition of a pattern is usually made by analyzing the pattern structure according to a given set of rules. Earlier applications of the syntactic approach include chromosome classification, English character recognition and identification of bubble and spark chamber events. The purpose of this monograph is to provide a summary of the major recent applications of syntactic pattern recognition.

After a brief introduction of syntactic pattern recognition in Chapter 1, the nine main chapters (Chapters 2–10) can be divided into three parts. The first three chapters concern with the analysis of waveforms using syntactic methods. Specific application examples include peak detection and interpretation of electrocardiograms and the recognition of speech patterns. The next five chapters deal with the syntactic recognition of two-dimensional pictorial patterns. Applications examples consist of Chinese character recognition, recognition of geometric figures and two-dimensional mathematical expressions, classification of fingerprint patterns and interpretation of Earth Resources Satellite data. The last chapter treats the problem of recognition of three-dimensional objects, namely, machine parts and industrial objects.

It is the authors of the individual chapters whose contributions made this volume possible. The editor wishes to express his heartfelt appreciation to the authors for their cooperation in its rapid completion.

West Lafayette, Indiana
Winter 1976/77

K. S. Fu

Contents

1. Introduction to Syntactic Pattern Recognition. By K.S.Fu (With 10 Figures)

1.1 Syntactic (Structural) Approach to Pattern Recognition 1
1.2 Syntactic Pattern Recognition System 5
1.3 Selection of Pattern Primitives 7
1.3.1 Primitive Selection Emphasizing Boundaries or Skeletons . . 10
1.3.2 Pattern Primitives in Terms of Regions 11
1.4 Pattern Grammar 13
1.5 High-Dimensional Pattern Grammars 16
1.6 Syntax Analysis as Recognition Procedure 23
1.6.1 Recognition of Finite-State Languages 23
1.6.2 Syntax Analysis of Context-Free Languages 25
1.7 Concluding Remarks 26
References 28

2. Peak Recognition in Waveforms. By S.L.Horowitz (With 4 Figures)

2.1 What is a Peak? 32
2.1.1 Mathematical Definition 32
2.1.2 Syntactical Definition 33
2.2 Practical Peak Recognition 36
2.2.1 Signal and Noise 36
2.2.2 Piecewise Linear Approximation Preprocessor 37
2.2.3 Peak Recognition Algorithms 39
2.3 Application to Electrocardiography 42
2.4 Concluding Remarks 47
References 48

3. Electrocardiogram Interpretation Using a Stochastic Finite State Model.
By J.E.Albus (With 8 Figures)

3.1 The Interpretation Problem 53
3.2 A Solution to the Interpretation Problem 54
3.2.1 The Tree Structure 55
3.2.2 The Yield of a Node 55
3.2.3 The Probability of a Node 55
3.2.4 The Tree Search 55
3.3 A Simple Example 56
3.4 Application to Analog Signal Processing 57

3.5 An Example Application 58
3.5.1 The Clock Model 58
3.5.2 The Clock Pulse Detector 59
3.5.3 The Total System 60
3.6 The Solution to the Clock Signal Interpretation Problem 60
3.6.1 The Search Tree 60
3.6.2 The Abbreviated Tree Structure 61
3.6.3 The Probability of a Node 61
3.6.4 Summary of the Tree Structure 62
3.7 A Sample Clock Signal Search Tree 62
3.8 Conclusion 64
References 64

4. Syntactic Recognition of Speech Patterns. By R. DeMori (With 9 Figures)
4.1 Background 65
4.2 Preprocessing and Feature Extraction 67
4.2.1 Signal Processing 67
4.2.2 Trajectories of Speech Parameters 68
4.2.3 Distinctive Features 69
4.3 Syntax-Controlled Segmentation of Continuous Speech 70
4.4 Syntactic Recognition of Syllables and Words 79
4.5 Linguistic Interpretation of Speech Patterns 84
4.5.1 Evaluation of Hypotheses 84
4.5.2 Preselection Rules 87
4.5.3 Verification of Hypotheses 87
4.6 Automatic Learning of Speech Patterns 89
4.7 Conclusions 92
References 92

5. Chinese Character Recognition. By W. W. Stallings (With 17 Figures)
5.1 Historical 95
5.1.1 An Approach to Pattern Recognition 95
5.1.2 A Chinese Reading Machine 95
5.1.3 Chinese Characters 96
5.1.4 Preview 98
5.2 The Morphology of Chinese Characters 98
5.2.1 Models 98
5.2.2 Some Applications 102
5.3 Recognition of Chinese Characters 105
5.3.1 The Model 105
5.3.2 Input 107
5.3.3 Analysis of Components 107
5.3.4 Analysis of Characters 112
5.3.5 Encoding of Components 114

5.3.6 Encoding of Characters 116
5.3.7 Results 117
5.3.8 Conclusions 118
5.3.9 Other Approaches 118
References 121

6. Shape Discrimination. By TH. PAVLIDIS and H.-Y. F. FENG (With 18 Figures)
6.1 Basic Considerations 125
6.2 Description of Contours in Terms of Polygons 126
6.3 Description of the Shape of Polygons 127
6.4 Fundamentals of Decomposition 128
6.5 Further Decomposition and Shape Description 131
6.6 Decomposition of Polygons with Holes 132
6.7 Implementation of the Decomposition Algorithm 133
6.8 Discussion of the Methodology 144
References 144

7. Two-Dimensional Mathematical Notation. By R. H. ANDERSON (With 7 Figures)
7.1 Mathematics Notation 147
7.2 Coordinate Grammars 147
7.3 A Syntax-Directed Recognition Algorithm 148
7.3.1 Characters 149
7.3.2 Syntactic Units 150
7.3.3 Coordinate Grammar Rules for Two-Dimensional Character Configurations 151
7.4 Scope of Recognition Capability 156
7.5 Implementation and Efficiency 158
7.6 Summary 159
Appendix 7.A 159
Appendix 7.B 167
References 177

8. Fingerprint Classification. By B. MOAYER and K. S. FU (With 17 Figures)
8.1 Historical Background 179
8.1.1 Fingerprint Pattern 179
8.1.2 Automatic Fingerprint Identification 182
8.2 Syntactic Approach 183
8.2.1 Digitizer 184
8.2.2 Preprocessing 184
8.2.3 Feature Extractor 185
8.2.4 Fingerprint Classifier 187
8.3 Tree Grammar Approach to Fingerprint Pattern Recognition . . . 198
8.3.1 Tree Classifier 199

8.3.2 Tree Grammar 203
8.3.3 Grammatical Inference 205
8.3.4 Computer Simulation 207
8.4 Summary 213
References 213

9. Modeling of Earth Resources Satellite Data. By J. M. BRAYER, P. H. SWAIN, and K. S. FU (With 25 Figures)

9.1 The Satellite Data 215
9.2 The Model 223
9.3 Details of the Analysis 224
9.3.1 Clouds and Shadows 224
9.3.2 The Downtown Area 230
9.4 Inferring a Grammar for the Highways 232
9.5 Summary 242
References 242

10. Industrial Objects and Machine Parts Recognition. By T. VÁMOS (With 20 Figures)

10.1 A Short Review of Leading Ideas 244
10.2 Levels of Knowledge 247
10.3 Hardware 248
10.3.1 Input 248
10.3.2 Output 250
10.4 Software 250
10.4.1 The Dictionary 250
10.4.2 Software Tools 253
10.5 Pattern Recognition 254
10.5.1 Preprocessing 254
10.5.2 Feature Extraction 256
10.5.3 The Grammar 256
10.5.4 The Organization of the Grammar 258
10.5.5 Some Heuristics 261
10.6 Conclusions 264
References 265

Subject Index 268

Contributors

ALBUS, JOHN EDWARD
7921 Brimfield, St. Apt. 3 Van Nuys, CA 91402, USA

ANDERSON, ROBERT HELMS
Rand Corporation, Santa Monica, CA 90406, USA

BRAYER, JOHN MARVIN
Department of Electrical Engineering and Computer Sciences, University of New Mexico, Albuquerque, NM 87106, USA

DEMORI, RENATO
Politecnio di Torino, Istituto di Elettrotecnica, Corso Duca degli Abruzzi 24, I-10129 Torino

FENG, HOU-YUAN FRANK
Department of Electrical Engineering, Princeton University, Princeton, NJ 08540, USA

FU, KING SUN
Purdue University, School of Electrical Engineering, West Lafayette, IN 47907, USA

HOROWITZ, STEVEN LESLIE
Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA

MOAYER, BIJAN
The Teheran Polytechnic, Shiraz Avenue, Teheran, Iran

PAVLIDIS, THEODOSIOS
Department of Electrical Engineering, Princeton University, Princeton, NJ 08540, USA

STALLINGS, WILLIAM WALTER
Advanced Computer Techniques Corporation, 1501 Wilson Boulevard, Arlington, VA 22209, USA

SWAIN, PHILIP H.
Purdue University, School of Electrical Engineering, West Lafayette, IN 47907, USA

VÁMOS, TIBOR
Computer and Automation Institute, Hungarian Academy of Sciences, H-1502 Budapest POB 63

1. Introduction to Syntactic Pattern Recognition

K.S.Fu

With 10 Figures

1.1 Syntactic (Structural) Approach to Pattern Recognition

Most of the developments in pattern recognition research during the past decade deal with the decision-theoretic approach [1.1–11] and its applications. In some pattern recognition problems, the structural information which describes each pattern is important, and the recognition process includes not only the capability of assigning the pattern to a particular class (to classify it), but also the capacity to describe aspects of the pattern which make it ineligible for assignment to another class. A typical example of this class of recognition problem is picture recognition, or more generally speaking, scene analysis. In this class of recogniton problems, the patterns under consideration are usually quite complex and the number of features required is often very large which makes the idea of describing a complex pattern in terms of a (hierarchical) composition of simpler subpatterns very attractive. Also, when the patterns are complex and the number of possible descriptions is very large, it is impractical to regard each description as defining a class (for example, in fingerprint and face identification problems, recognition of continuous speech, Chinese characters, etc.). Consequently, the requirement of recognition can be satisfied only by a description for each pattern rather than the simple task of classification.

Example 1.1: The pictorial patterns shown in Fig. 1.1a can be described in terms of the hierarchical structures shown in Fig. 1.1b.

In order to represent the hierarchical (tree-like) structural information of each pattern, that is, a pattern described in terms of simpler subpatterns and each simpler subpattern again be described in terms of even simpler subpatterns, etc., the syntactic or structural approach has been proposed [1.12–16]. This approach draws an analogy between the (hierarchical, tree-like) structure of patterns and the syntax of languages. Patterns are specified as building up out of subpatterns in various ways of composition just as phrases and sentences are built up by concatenating words and words are built up by concatenating characters. Evidently, for this approach to be advantageous, the simplest subpatterns selected, called "pattern primitives", should be much easier to recognize than the patterns themselves. The "language" which provides the structural description of patterns in terms of a set of pattern primitives and their composition operations, is sometimes called "pattern description language". The rules governing the composition of primitives into patterns are usually specified by the co-called "grammar" of the pattern description language. After each primitive within the pattern is identified, the recognition

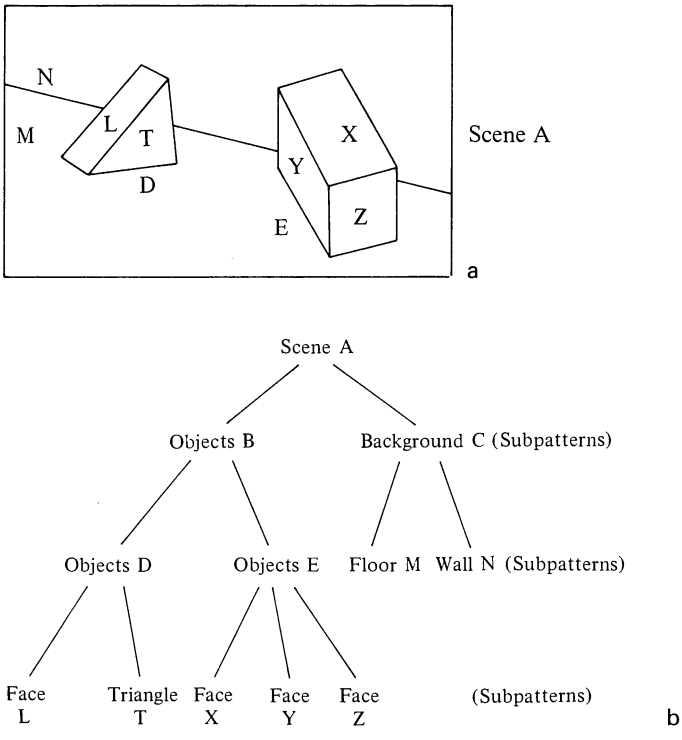


Fig. 1.1a and b. The pictorial pattern *A* and its hierarchical structural descriptions

process is accomplished by performing a syntax analysis or parsing of the "sentence" describing the given pattern to determine whether or not it is syntactically (or grammatically) correct with respect to the specified grammar. In the meantime, the syntax analysis also produces a structural description of the sentence representing the given pattern (usually in the form of a tree structure).

The syntactic approach to pattern recognition provides a capability for describing a large set of complex patterns using small sets of simple pattern primitives and of grammatical rules. The various relations or composition operations defined among subpatterns can usually be expressed in terms of logical and/or mathematical operations. As can be seen later, one of the most attractive aspects of this capability is the use of recursive nature of a grammar. A grammar (rewriting) rule can be applied any number of times, so it is possible to express in a very compact way some basic structural characteristics of an infinite set of sentences. Of course, the practical utility of such an approach depends upon our ability to recognize the simple pattern primitives and their relationships represented by the composition operations.

An alternative representation of the structural information of a pattern is to use a "relational graph". For example, a relational graph of Pattern A in Fig. 1.1a is shown in Fig. 1.2. Since there is a one-to-one corresponding

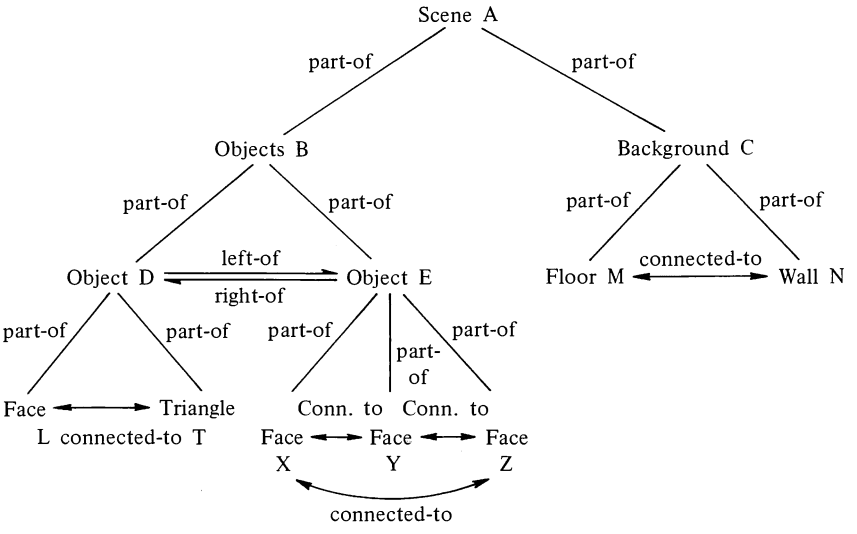


Fig. 1.2. A relational graph of scene A

relation between a linear graph and a matrix, a relational graph can certainly also be expressed as a "relational matrix". In using the relational graph for pattern description, we can broaden the class of allowed relations to include any relation that can be conveniently determined from the pattern. With this generalization, we may possibly express richer descriptions than we can with tree structures. However, the use of tree structures does provide us a direct channel to adapt the techniques of formal language theory to the problem of compactly representing and analyzing patterns containing significant structural information.

We briefly introduce some important definitions and notations in this section.

Definition 1.1: A (phrase-structure) grammar G is a four-triple

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

where

V_N is a finite set of nonterminals,

V_T is a fintie set of terminals,

$S \in V_N$ is the start symbol,

and P is a finite set of rewrite rules or productions denoted by

$$\alpha \rightarrow \beta . \tag{1.1}$$

α and β are strings over the union of V_N and V_T and with α involving at least one symbol of V_N.

The following notations are frequently used.

1) V^* is the set of all strings of symbols in V, including λ, the string of length 0, $V^+ = V^* - \{\lambda\}$.

2) If x is a string, x^n is x written n times.

3) $|x|$ is the length of the string x, or the number of symbols in string x.

4) $\eta \underset{G}{\Rightarrow} \gamma$, or a string η directly generates or derives a string γ if $\eta = \omega_1 \alpha \omega_2$, $\gamma = \omega_1 \beta \omega_2$, and $\alpha \to \beta$ is a production in P.

5) $\eta \overset{*}{\underset{G}{\Rightarrow}} \gamma$, or a string η generates or derives a string γ if there exists a sequence of strings $\zeta_1, \zeta_2, \ldots \zeta_n$ such that $\eta = \zeta_1$, $\gamma = \zeta_n$, $\zeta_i \Rightarrow \zeta_{i+1}$, $i = 1, 2, \ldots, n-1$. The sequence of strings $\zeta_1, \zeta_2, \ldots, \zeta_n$ is called a derivation of γ from η.

Definition 1.2: The language generated by grammar G is

$$L(G) = \{x | x \in V_T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} x\}. \tag{1.2}$$

That is, the language consists of all strings or sentences of terminals generated from the start symbol S.

Definition 1.3: In (1.1) if $|\alpha| \leqq |\beta|$, the grammar is called a type 1 or context-sensitive grammar. If $\alpha = A \in V_N$, the grammar is called a type 2 or context-free grammar. If, in addition to $\alpha = A$, $\beta = aB$ or $\beta = a$, where $a \in V_T$ and $B \in V_N$, the grammar is called a type 3, or finite-state, or regular grammar.

The languages generated by context-sensitive, context-free, and finite-state (regular) grammars are called context-sensitive, context-free, and finite-state (regular) languages, respectively.

Example 1.2: Consider the context-free grammar

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

where $V_N = \{S, A, B\}$, $V_T = \{a, b\}$, and P[1]:

1) $S \to aB$	5) $A \to a$
2) $S \to bA$	6) $B \to bS$
3) $A \to aS$	7) $B \to aBB$
4) $A \to bAA$	8) $B \to b$.

The language generated by G, $L(G)$, is the set of all sentences or strings in V_T^+ consisting of an equal number of a's and b's. Typical generations or derivations of sentences include

$$S \overset{(1)}{\Rightarrow} aB \overset{(8)}{\Rightarrow} ab$$
$$S \overset{(1)}{\Rightarrow} aB \overset{(6)}{\Rightarrow} abS \overset{(2)}{\Rightarrow} abbA \overset{(5)}{\Rightarrow} abba$$
$$S \overset{(2)}{\Rightarrow} bA \overset{(4)}{\Rightarrow} bbAA \overset{(4)}{\Rightarrow} bbbAAA \overset{(5)}{\Rightarrow} bbbaAA \overset{(5)}{\Rightarrow} bbbaaA \overset{(5)}{\Rightarrow} bbbaaa$$

where the parenthesized number indicates the production used.

[1] For convenience, we can also used the shorthand notation $S \to aB|bA$ for representing productions 1) and 2). Similarly, we can use $A \to aS|bAA|a$ for productions 3), 4), and 5), and use $B \to bS|aBB|b$ for productions 6), 7), and 8).

An alternative method for describing any derivation in a context-free grammar is the use of derivation or parse trees. A derivation tree for a context-free grammar can be constructed according to the following procedure:

1) Every node of the tree has a label, which is a symbol in V_N or V_T.

2) The root of the tree has the label S.

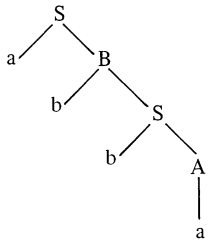
3) If a node has at least one descendant other than itself, and has the label A, then $A \in V_N$.

4) If nodes $n_1, n_2, \ldots n_k$ are the direct descendants of node n (with label A) in the order from left to right, with labels $A_1, A_2, \ldots, A_k$, respectively, then

$$A \rightarrow A_1 A_2, \ldots, A_k$$

must be a production in P.

For example, the derivation $S \overset{*}{\Rightarrow} abba$ in Example 1.2 can be described by the following derivation tree:



1.2 Syntactic Pattern Recognition System

A syntactic pattern recognition system can be considered as consisting of three major parts; namely, preprocessing, pattern description or representation, and syntax analysis[2]. A simple block diagram of the system is show in Fig. 1.3. The functions of preprocessing include i) pattern encoding and approximation, and ii) filtering, restoration and enhancement. An input pattern is first coded or approximated by some convenient form for further processing. For example, a black-and-white picture can be coded in terms of a grid (or a matrix) of 0's and 1's, or a waveform can be approximated by its time samples or a truncated Fourier series expansion. In order to make the processing in the later stages of the system more efficient, some sort of "data compression" is often applied at this stage. Then, techniques of filtering, restoration and/or enhancement will be used to clean the noise, to restore the degradation, and/or to improve the quality of the coded (or approximated) patterns. At the output of the preprocessor, presumably, we have patterns with reasonably "good quality". Each preprocessed pattern is then represented by a language-like structure (for example, a string or a graph). The operation of this pattern-

2 The division of three parts is for convenience rather than necessity. Usually, the term "linguistic pattern recognition" refers primarily to the pattern representation (or description) and the syntax analysis.

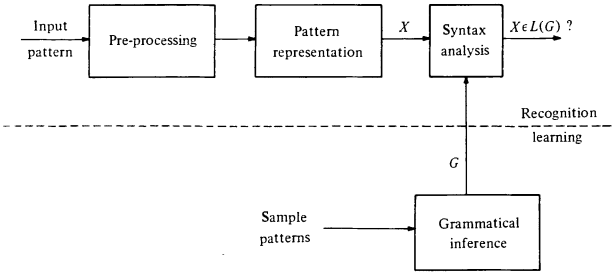


Fig. 1.3. Block diagram of a syntactic pattern recognition system

representation process consists of i) pattern segmentation, and ii) primitive (feature) extraction. In order to represent a pattern in terms of its subpatterns, we must segmentize the pattern and, in the meantime, identify (or extract) the primitives and relations in it. In other words, each preprocessed pattern is segmentized into subpatterns and pattern primitives based on prespecified syntactic or composition operations; and, in turn, each subpattern is identified with a given set of pattern primitives. Each pattern is now represented by a set of primitives with specified syntactic operations. For example, in terms of "concatenation" operation, each pattern is represented by a string of (concatenated) primitives. More sophisticated systems should also be able to detect various syntactic relations within the pattern. The decision on whether or not the representation (pattern) is syntactically correct (i.e., belongs to the class of patterns described by the given syntax or grammar) will be performed by the "syntax analyzer" or "parser". When performing the syntax analysis or parsing, the analyzer can usually produce a complete syntactic description, in terms of a parse or parsing tree, of the pattern provided it is syntactically correct. Otherwise, the pattern is either rejected or analyzed on the basis of other given grammars, which presumably describe other possible classes of patterns under consideration.

Conceptually, the simplest form of recognition is probably "template-matching". The string of primitives representing an input pattern is matched against strings of primitives representing each prototype or reference pattern. Based on a selected "matching" or "similarity" criterion, the input pattern is classified in the same class as the prototype pattern which is the "best" to match the input. The hierarchical structural information is essentially ignored. A complete parsing of the string representing an input pattern, on the other hand, explores the complete hierarchical structural description of the pattern. In between, there are a number of intermediate approaches. For example, a series of tests can be designed to test the occurrences or non-occurrence of certain subpatterns (or primitives) or certain combinations of subpatterns or primitives. The result of the tests (for example, through a table look-up, a decision tree, or a logical operation) is used for a classification decision. Notice that each test may be a template-matching scheme or a parsing for a subtree representing a subpattern. The selection of an appropriate approach for recognition usually depends upon the problem requirement. If a complete pattern description is required for recognition, parsing is necessary. Otherwise, a

complete parsing could be avoided by using other simpler approaches to improve the efficiency of the recognition process.

In order to have a grammar describing the structural information about the class of patterns under study, a grammatical inference machine is required which can infer a grammar from a given set of training patterns in language-like representations[3]. This is analogous to the "learning" process in a decision-theoretic pattern recognition system [1.1–11, 17–20]. The structural description of the class of patterns under study is learned from the actual sample patterns from that class. The learned description, in the form of a grammar, is then used for pattern description and syntax analysis (see Fig. 1.3). A more general form of learning might include the capability of learning the best set of primitives and the corresponding structural description for the class of patterns concerned.

1.3 Selection of Pattern Primitives

As was discussed in Section 1.1, the first step in formulating a linguistic model for pattern description is the determination of a set of primitives in terms of which the patterns of interest may be described. This will be largely influenced by the nature of the data, the specific application in question, and the technology available for implementing the system. There is no general solution for the primitive selection problem at this time. The following requirements usually serve as a guideline for selecting pattern primitives.

i) The primitives should serve as basic pattern elements to provide a compact but adequate description of the data in terms of the specified structural relations (e.g., the concatenation relation).

ii) The primitives should be easily extracted or recognized by existing non-linguistic methods, since they are considered to be simple and compact patterns and their structural information not important.

For example, for speech patterns, phonemes are naturally considered as a "good" set of primitives with the concatenation relation[4]. Similarly, strokes have been suggested as primitives in describing handwriting. However, for general pictorial patterns, there is no such "universal picture element" analogous to phonemes in speech or strokes in handwriting[5]. Sometimes, in order to provide an adequate description of the patterns, the primitives should contain the information which is important to the specific application in question. For example, If the size (or shape or location) is important in the recognition problem, then the primitives should contain information relating to size (or shape or location) so that patterns from different classes are distinguishable by whatever method

[3] At present, this part is performed primarily by the designer.

[4] The view of continuous speech as composed of one sound segment for each successive phoneme is, of course, a simplification of facts.

[5] It is also interesting to see that the extraction of phonemes in continuous speech and that of strokes in handwriting is not a very easy task with respect to the requirement ii) specified above.

is to be applied to analyze the descriptions. This requirement often results in a need for semantic information in describing primitives [1.12].

Requirement ii) may sometimes conflict with requirement i) due to the fact that the primitives selected according to requirement i) may not be easy to recognize using existing techniques. On the other hand, requirement ii) could allow the selection of quite complex primitives as long as they can be recognized. With more complex primitives, simpler structural descriptions (e.g., simple grammars) of the patterns could be used. This tradeoff may become quite important in the implementation of the recognition system. An example is the recognition of two-dimensional mathematical expressions in which characters and mathematical notations are primitives. However, if we consider the characters as subpatterns and describe them in terms of simpler primitives (e.g., strokes or line segments), the structural descriptions of mathematical expressions would be more complex than the case of using characters directly as primitives.

One of the earliest papers describing the decomposition of pictorial patterns into primitives [1.20a] presented a conceptually appealing method which allows the recognition system to (heuristically) determine the primitives by inspection of training samples. A pattern is first examined by a programmed scan. The result of the scan is to produce descriptions of segments of the picture (subpictures) which are divisions conveniently produced by the scanning process, and not necessarily true divisions. The scanning process also includes pre-processing routines for noise-cleaning, gap-filling, and curve-following. The subpictures obtained in the scan are analyzed and connected, when appropriate, into true picture parts; a description is given in terms of the length and slope of straight-line segments and the length and curvature of curved segments. The structural relations among various segments (primitives) of a picture are expressed in terms of a connection table (Table of Joins). The assembly program produces a "statement" which gives a complete description of the pattern. The description is independent of the orientation and the size of the picture, the lengths of the various parts being given relative to one another. It is, in effect, a coded representation of the pattern and may be regarded as a one-dimensional string consisting of symbols chosen from a specified alphabet. The coded representation gives the length, slope and curvature of each primitive, together with details of the ends and joins to other primitives. No explicit consideration is given to formalizing the pattern syntax.

A formal model for the abstract description of English cursive script has been proposed by EDEN and HALLE [1.21]. The primitives are four distinct line segments in the form of a triple

$$\sigma_j = [(x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2}), \theta_j] \tag{1.3}$$

where (x_j, y_j)'s represent the approximate location of the end points of the line segment, and θ_j refers to the sense of rotation from the first to the second end point. θ_j is positive if the sense of rotation is clockwise and negative

if counterclockwise. The four primitives are

$\sigma_1 = [(1, 0), (0, 0), +]$ "bar"

$\sigma_2 = [(1, 1), (0, 0), +]$ "hook"

$\sigma_3 = [(0, 0), (0, 1), +]$ "arch"

$\sigma_4 = [(1, \varepsilon), (0, 0), +], \quad 0 < \varepsilon < 1,$ "loop"

They can be transformed by changing the sign of θ or by reflection about the horizontal or vertical axis. These transformations generate 28 strokes (because of symmetry, the arch generates only four strokes), but only nine of them are of interest in the English script commonly used.

A word is completely specified by the stroke sequence comprising its letters. A word is represented by the image of a mapping of a finite sequence of strokes into the set of continuous functions, the mapping being specified by concatenation and tracing rules applied in specific order. Only two concatenation rules are required. The first specifies stroke locations within a letter. The rule prescribes that two consecutive strokes are concatenated by identifying the abscissa of the terminal endpoint of the first stroke with that of the initial endpoint of the second stroke. The second rule states that across a letter boundary, the leftmost end point of the stroke following the boundary is placed so as to be to the right of the rightmost end point of the penultimate stroke before the letter boundary. The simple cursive strokes of the word "globe" are shown in Fig. 1.4a and their concatenation in Fig. 1.4b. These concatenation rules are not sufficient to specify all sequences of English letters unambiguously. Nevertheless, the ambiguities are intrinsic to the writing system, even in careful handwriting.

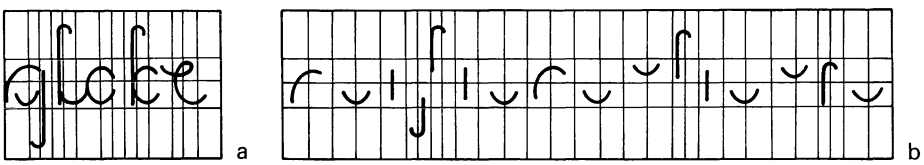


Fig. 1.4. (a) Cursive strokes of "globe"; (b) Stroke sequence representation of the word "globe"

No formal syntax was attempted for the description of handwriting. Interesting experimental results on the recognition of cursive writing were obtained by EARNEST [1.22] and MERMELSTEIN and EDEN [1.23] using a dictionary and rather heuristic recognition criteria. In addition, the dynamics of the trajectory (in space and time) that the point of the pen traces out as it moves across the paper has also been studied [1.24]. The motion of the pen is assumed to be controlled by a pair of orthogonal forces, as if one pair of muscles controls the vertical displacement and another the horizontal.

More general methods for primitive selection may be grouped roughly into methods emphasizing boundaries and methods emphasizing regions. These methods are discussed in the following.

1.3.1 Primitive Selection Emphasizing Boundaries or Skeletons

A set of primitives commonly used to describe boundaries or skeletons is the chain code given by Freeman [1.25, 25a]. Under this scheme, a rectangular grid is overlaid on the two-dimensional pattern, and straight line segments are used to connect the grid points falling closest to the pattern. Each line segment is assigned an octal digit according to its slope The pattern is thus represented by a chain (or string) or chains of octal digits. Figure 1.5 illustrates the primitives and the coded string describing a curve. This descriptive scheme has some useful properties. For example, patterns coded in this way can be rotated through multiples of 45° simply by adding an octal digit (modulo 8) to every digit in the string (however, only rotations by multiples of 90° can be accomplished without some distortion of the pattern). Other simple manipulations such as expansion, measurement of curve length, and determination of pattern self-intersections are easily carried out. Any desired degree of resolution can be obtained by adjusting the fineness of the grid imposed on the patterns. This method is, of course, not limited to simply connected closed boundaries; it can be used for describing arbitrary two-dimensional figures composed of straight or curved lines and line segments.

Notable work using Freeman's chain code include efforts by Knoke and Wiley [1.26] and by Feder [1.27]. Knoke and Wiley attempted to demonstrate that linguistic approaches can usually be applied to describe structural relationships within patterns (hand-printed characters, in this case). Feders's work considers only patterns which can be encoded as strings of primitives. Several bases for developing pattern languages are discussed, including equations in two variables (straight lines, circles and circular arcs, etc.), pattern properties (self-intersections, convexity, etc., and various measures of curve similarity. The computational power (automaton complexity) required to detect the elements

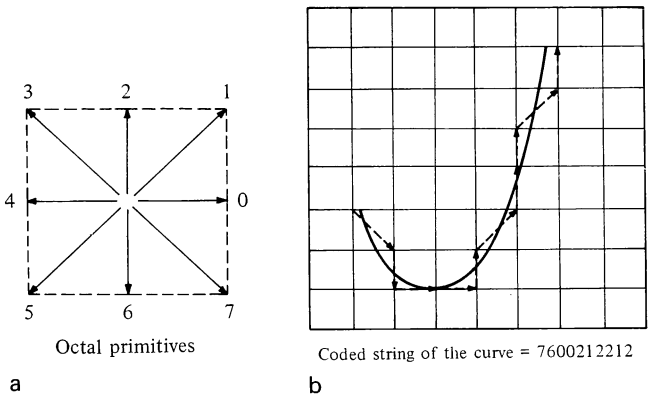


Fig. 1.5a and b. Freeman's chain code

of these pattern languages is studied. However, this problem is complicated considerably by the fact that i) these languages are mostly context-sensitive and not context-free, ii) the chain code yields only a piecewise linear approximation of the original pattern, and iii) the coding of a typical curve is not unique, depending to a degree on its location and orientation with respect to the coding grid.

Other applications of the chain code include description of contour maps [1.28], "shape matching" [1.29], and identification of high energy particle tracks in bubble chamber photographs [1.30]. Contour lines can be encoded as chains. Contour map problems may involve finding the terrain to be flooded by a dam placed at a particular location, the water shed area for a river basin, the terrain visible from a particular mountain-top location, or the determination of optimum highway routes through mountainous terrain. In shape matching, two or more two-dimensional objects having irregular contours are to be matched for all or part of their exterior boundary. For some such problems the relative orientation and scale of the objects to be matched may be known and only translation is required. The problem of matching aerial photographs to each other as well as to terrain maps falls into this category. For other problems either orientation, or scale, or both may be unknown and may have to be determined as part of the problem. An example of problems in which relative orientation has to be determined is that of the computer assembly of potsherds and jigsaw puzzles [1.31].

Other syntactic pattern recognition systems using primitives with the emphasis on boundary, skeleton or contour information include systems for hand-printed character recognition [1.32–34], bubble chamber and spark chamber photograph classification [1.35–38], chromosome analysis [1.39–41], fingerprint identification (Chapt. 8), face recognition [1.42, 43], and scene analysis [1.44–46].

1.3.2 Pattern Primitives in Terms of Regions

A set of primitives for encoding geometric patterns in terms of regions has been proposed by PAVLIDIS [1.47]. In this case, the basic primitives are halfplanes in the pattern space[6] (or the field of observation). It can be shown that any figure (or arbitrary polygon) may be expressed as the union of a finite number of convex polygons. Each convex polygon can, in turn, be represented as the intersection of a finite number of halfplanes. By defining a suitable ordering (a sequence) of the convex polygons composing the arbitrary polygon, it is possible to determine a unique minimal set of maximal (in an appropriate sense) polygons, called primary subsets, the union of which is the give polygon. In linguistic analogy, a figure can be thought of as a "sentence", the convex polygons composing it as "words" and the halfplanes as "letter". This process is summarized in this section.

Let A be a bounded polygon and let $s_1, s_2, \ldots, s_n$ be its sides. A point x in the plane will be said to be positive with respect to one side if it lies on the same side of the extension of a side as the polygon does with respect to the side itself. Otherwise, it will be said to be negative with respect to that side.

[6] This could be generalized to halfspaces of the pattern space.

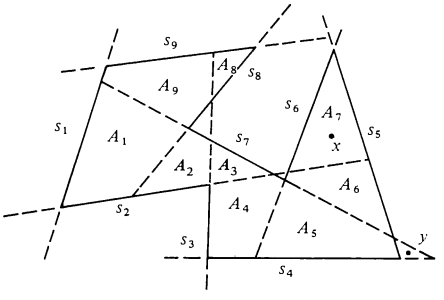


Fig. 1.6. Polygon A of Example 1.3

Example 1.3: For the polygon A given in Fig. 1.6, the point x is positive with respect to the sides s_5 and s_6, but negative with respect to s_7. Similarly, y is positive with respect to s_4 and s_7, but negative with respect to s_5. Extending all the sides of A in both directions, A is intersected by some of these extensions, and it is subdivided into $A_1, A_2, \ldots, A_9$ convex polygons.

Obviously, the points which are positive with respect to a side form a halfplane whose boundary is the extension of the side. Let h_i denote the halfplane corresponding to the side s_i, and let Q denote the intersection of all halfplanes $h_1, h_2, \ldots, h_n$ in A. If A is convex, then $A = Q$. If A is not convex, then Q may be empty or simply different from A. Let Q_I represent the intersection of all the halfplanes except $s_{i_1}, \ldots, s_{i_k}$ where $I = \{i_1, \ldots, i_k\}$, the index set. Then we can define a sequence of Q_I as follows:

$$
\begin{aligned}
Q &= \bigcap_{i=1}^{n} h_i \\
Q_j &= \bigcap_{\substack{i=1 \\ i \neq j}}^{n} h_i \\
Q_{jk} &= \bigcap_{\substack{i=1 \\ i \neq j,\, i \neq k}}^{n} h_i .
\end{aligned}
\tag{1.4}
$$

This is an increasing sequence since $Q \subset Q_j \subset Q_{jk} \ldots$. The last element of the sequence will be the whole plane, and it is obtained for $I = \{1, \ldots, n\}$. If a sequence of the above form has a maximal element, then that set is called a primary (convex) subset of A. A nonempty member of such a Q-sequence which is also a subset of A is called a nucleus of A if all the previous elements of the sequence are empty. Consequently, it can be shown that the union of the primary subsets of A precisely equals A.

For a given polygon the primary subsets can be found by forming all the sequences $Q, Q_j, Q_{jk}, \ldots$ and searching for their maximal elements. This is a well-defined procedure and, hence, the primary subsets of A are unique.

It is noted that this approach provides a formalism for describing the syntax of polygonal figures and more general figures which can be approximated

reasonably well by polygonal figures. The analysis or recognition procedure requires the definition of suitable measures of similarity between polygons. The similarity measures considered so far are quite sensitive to noise in the patterns and/or are difficult to implement practically on a digital computer. A somewhat more general selection procedure of pattern primitives based on regions has been recently proposed by ROSENFELD and STRONG [1.48].

Another form of representing polygonal figures is the use of primary graphs [1.49, 50]. The primary graph of a polygon A is one whose nodes correspond to the nuclei and the primary subsets of A, and its branches connect each nucleus to all the primary subsets containing it. Another approach to the analysis of geometric patterns using regions is discussed primarily in the problem of scene analysis [1.9, 45]. MINSKY and PAPERT [1.51] have considered the direct transformation of a gray scale picture to regions, bypassing the edge-finding, line-fitting procedures. Regions are constructed as the union of squares whose corners have the same or nearly the same gray scale. The method proposed by GUZMAN [1.52] assumes that a picture can be reduced by preprocessing to a list of vertices, lines and surfaces. Various heuristics, based on the analysis of types of intersections of lines and surfaces, are applied to this list to compose its elements into two- or three-dimensional regions. Some candidate pattern recognition schemes have been investigated, all of which involve methods for matching the reduced pattern descriptions against a prototype dictionary. The procedure studied by BRICE and FENNEMA [1.53] decomposes a picture into atomic regions of uniform gray scale. A pair of heuristics is used to join these regions in such a way as to obtain regions whose boundaries are determined more by the natural lines of the scene than by the artificial ones introduced by quantization and noise. Then a simple line-fitting technique is used to approximate the region boundaries by straight lines and finally, the scene analyzer interprets the picture using some simple tests on object groups generated by a Guzman-like procedure.

1.4 Pattern Grammar

Assume that a satisfactory solution of the "primitive selection" problem is available for a given application. The next step is the construction of a grammer (or grammars) which will generate a language (or languages) to describe the patterns under study. Ideally, it would be nice to have a grammatical inference machine which would infer a grammar from a given set of strings describing the patterns under study. Unfortunately, such a machine has not been available except for some very special cases [1.54]. In most cases so far, the designer constructs the grammar based on the a priori knowledge available and his experience. It is known that increased descriptive power of a language is paid for in terms of increased complexity of the analysis system (recognizer or acceptor). Finite-state automata are capable of recognizing or accepting finite-state languages although the descriptive power of finite-state languages is also known to be

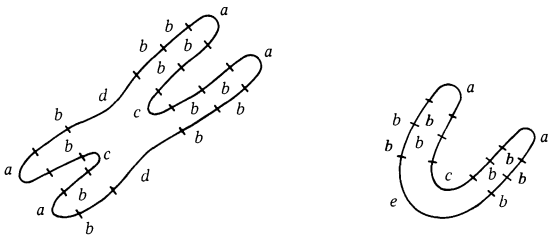
weaker than that of context-free and context-sensitive languages. On the other hand, nonfinite, nondeterministic devices are required, in general, to accept the languages generated by context-free and context-sensitive grammars. Except for the class of deterministic languages, nondeterministic parsing procedures are usually needed for the analysis of context-free languages. The tradeoff between the descriptive power and the analysis efficiency of a grammar for a given application is, at present, almost completely justified by the designer. (For example, a precedence language may be used for pattern description in order to obtain good analysis efficiency, or, on the other hand, a context-free programmed grammar generating a context-sensitive language may be selected in order to describe the patterns effectively.) The effect of the theoretical difficulty may not be serious in practice, as long as some care is exercised in developing the required grammars. This is especially true when the languages of interest are actually finite-state, even though the form of the grammars may be context-sensitive, or when the languages may be approximated by finite-state languages.

It should be remarked that a grammar is most appropriate for description when the pattern of interest is built up from a small set of primitives by recursive application of a small set of production rules. Also, the "primitive selection" and the "grammar construction" should probably be treated simultaneously rather than in two different stages. There is no doubt that a different selection of pattern primitives will result in a different grammar for the description of a given set of patterns. Sometimes, a compromise is necessary in order to develop a suitable grammar.

Although many classes of patterns appear to be intuitively context-sensitive, context-sensitive (but not context-free) grammars have rarely been used for pattern description simply because of their complexity. Context-free languages have been used to describe patterns such as English characters [1.55], chromosome images [1.39], spark chamber pictures [1.35], chemical structures [1.56], fingerprint patterns (Chapt. 8), plane projective geometry [1.57] and spoken digits (Chapt. 4).

Example 1.4: The following is a context-free grammar describing the chromosome images shown in Fig. 1.7 [1.39].

$$G=(V_N, V_T, P, \{\langle\text{submedian chromosome}\rangle, \langle\text{telocentric chromosome}\rangle\})$$



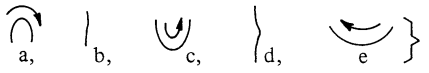
babcbabdbbabbcbbabbd *bbebbabbcbba*

a b

Fig. 1.7. (a) Submedian chromosome; (b) Telocentric chromosome

where

$V_N = \{$⟨submedian chromosome⟩, ⟨telocentric chromosome⟩, ⟨arm pair⟩, ⟨left part⟩, ⟨right part⟩, ⟨arm⟩, ⟨side⟩, ⟨bottom⟩$\}$

$V_T =$ 

and

P: ⟨submedian chromosome⟩→⟨arm pair⟩ ⟨arm pair⟩
⟨telocentric chromosome⟩→⟨bottom⟩ ⟨arm pair⟩
⟨arm pair⟩→⟨side⟩ ⟨arm pair⟩
⟨arm pair⟩→⟨arm pair⟩ ⟨side⟩
⟨arm pair⟩→⟨arm⟩ ⟨right part⟩
⟨arm pair⟩→⟨left part⟩ ⟨arm⟩
⟨left part⟩→⟨arm⟩ *c*
⟨right part⟩→*c*⟨arm⟩
⟨bottom⟩→*b* ⟨bottom⟩
⟨bottom⟩→⟨bottom⟩ *b*
⟨bottom⟩→*e*
⟨side⟩→*b* ⟨side⟩
⟨side⟩→⟨side⟩ *b*
⟨side⟩→*b*
⟨side⟩→*d*
⟨arm⟩→*b* ⟨arm⟩
⟨arm⟩→⟨arm⟩ *b*
⟨arm⟩→*a*

In addition to i) the tradeoff between the language descriptive power and the analysis efficiency, and ii) the compromise sometimes necessary between the primitives selected and the grammar constructed, the designer should also be aware of the need to control the excessive strings generated by the constructed grammar. The number of pattern strings available in practice is always limited. However, in most cases, the grammar constructed would generate a large or infinite number of strings[7]. It is hoped that the excessive strings generated

[7] It may be argued that, in practice, a pattern grammar can always be finite-state since it is constructed from a finite number of pattern strings. However, the finite-state grammar so constructed may require a large number of productions. In such a case, a context-free or a context-free programmed pattern grammar may be constructed for the purpose of significantly reducing the number of productions.

are similar to the available pattern strings. Unfortunately, this may not be true since the grammar, in many cases, is constructed heuristically. The problem may become very serious when the excessive strings include some pattern strings which should belong to other classes. In this case, adjustments should be made to exclude these strings from the language generated by the constructed grammar.

Recently, probably due to their relative effectiveness in describing natural language structures, transformational grammars have been proposed for pattern description [1.58–61]. Transformational grammars would allow the possibility of determining from the pattern generative mechanism a simple base grammar (deep structure) which generates a certain set of patterns and a problem-oriented set of transformations. Through the base grammar and the transformations, the original set of patterns can be described.

From the above discussion, it might be concluded that, before efficient grammatical inference procedures are available, a man-machine interactive system would be suitable for the problem of grammar construction. The basic grammar and the various tradeoffs and compromises have to be determined by the designer. The results of any adjustment on the grammar constructed can be easily checked and displayed through a computer system.

1.5 High-Dimensional Pattern Grammars

In describing patterns using a string grammar, the only relation between subpatterns and/or primitives is the concatenation; that is, each subpattern or primitive can be connected only at the left or right. This one-dimensional relation has not been very effective in describing two- or three-dimensional patterns. A natural generalization is to use a more general formalism including other useful relations [1.55, 56, 62–66]. Let R be a set of n-ary relations ($n \geqq 1$). A relation $r \in R$ satisfied by the subpatterns and/or primitives $X_1, \ldots, X_n$ is denoted $r(X_1, \ldots, X_n)$. For example, TRIANGLE (a, b, c) means that the ternary relation TRIANGLE is satisfied by the line segments a, b, and c, and ABOVE (X, Y) means that X is above Y. The following example illustrates pattern descriptions using this formalism of relations.

Example 1.5: The mathematical expression

$$\frac{a+b}{c}$$

can be described by

$$\text{ABOVE}(\text{ABOVE}(\text{LEFT}(a, \text{LEFT}(+, b)), —), c)$$

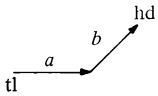
where LEFT(X, Y) means that X is to the left of Y.

A simple two-dimensional generalization of string grammars is to extend grammars for one-dimensional strings to two-dimensional arrays [1.67, 68]. The primitives are the array elements and the relation between primitives is the two-dimensional concatenation. Each production rewrites one subarray by

another, rather than one substring by another. Relationships between array grammars and array automata (automata with two-dimensional tapes) have been studied recently [1.69].

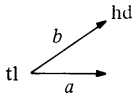
SHAW, by attaching a "head" (hd) and a "tail" (tl) to each primitive, has used the four binary operators $+$, $\times$, $-$ and $*$ for defining binary concatenation relations between primitives [1.70, 71].

$a+b$



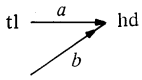
$$\mathrm{hd}(a+b)=\mathrm{hd}(b)$$
$$\mathrm{tl}(a+b)=\mathrm{tl}(a)$$

$a\times b$



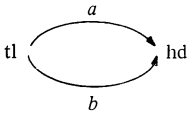
$$\mathrm{hd}(a\times b)=\mathrm{hd}(b)$$
$$\mathrm{tl}(a\times b)=\mathrm{tl}(b)$$

$a\text{-}b$



$$\mathrm{hd}(a-b)=\mathrm{hd}(a)$$
$$\mathrm{tl}(a-b)=\mathrm{tl}(a)$$

$a*b$



$$\mathrm{hd}(a*b)=\mathrm{hd}(a)$$
$$\mathrm{tl}(a*b)=\mathrm{tl}(a)$$

For string languages, only the operator $+$ is used. In addition, the unary operator $\sim$ acting as a tail/head reverser is also defined; i.e.,

$\sim a$

$$\mathrm{hd}(\sim a)=\mathrm{tl}(a)$$
$$\mathrm{tl}(\sim a)=\mathrm{hd}(a)\,.$$

In the case of describing patterns consisting of disconnected subpatterns, the "blank" or "don't care" primitive is introduced. Each pictorial pattern is represented by a "labelled branch-oriented graph" where branches represent primitives.

The grammar which generates sentences (PDL expressions) in PDL (Picture Description Language) is a context-free grammar

$$G=(V_{\mathrm{N}}, V_{\mathrm{T}}, P, S)$$

where

$$V_{\mathrm{N}}=\{S, SL\}$$

$V_{\mathrm{T}}=\{b\}\cup\{+, \times, -, /, (,)\}\cup\{l\}$, b may be any primitive (including the "null point primitive" λ which has identical tail and head)

and

$$S \to b,\ S \to (S\phi_b S),\ S \to (\sim S),\ S \to SL,\ S \to (/SL),$$
$$SL \to Sl,\ SL \to (SL\phi_b SL),\ SL \to (\sim SL),\ SL \to (/SL),$$
$$\phi_b \to +,\ \phi_b \to \times,\ \phi_b \to -,\ \phi_b \to *.$$

l is a label designator which is used to allow cross reference to the expressions S within a description. The / operator is used to enable the tail and head of an expression to be arbitrarily located.

A top-down parsing procedure (see Sect. 1.6.) was used for the recognition of PDL expressions describing pictorial patterns [1.71].

Based on an idea in [1.32], Feder has formalized a "plex" grammar which generates languages with terminals having an arbitrary number of attaching points for connecting to other primitives or subpatterns [1.56]. The primitives of the plex grammar are called N-Attaching Point Entity (NAPE). Each production of the plex grammar is in context-free form in which the connectivity of primitives or subpatterns is described by using explicit lists of labelled concatenation points (called joint lists). While the sentences generated by a plex grammar are not directed graphs, they can be transformed by either assigning labelled nodes to both primitives and concatenation points as suggested by Pfaltz and Rosenfeld [1.72] or by transforming primitives to nodes and concatenations to labelled branches [1.73].

Pfaltz and Rosenfeld have extended the concept of string grammars to grammars for labelled graphs called webs. Labelled node-oriented graphs are explicitly used in the productions. Each production describes the rewriting of a graph α into another graph β and also contains an "embedding" rule E which specifies the connection of β to its surrounding graph (host web) when α is rewritten. A web grammar G is a 4-tuple

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

where V_N is a set of nonterminals, V_T is a set of terminals, S is a set of "initial" webs, and P is a set of web productions. A web production is defined as[8].

$$\alpha \to \beta, E$$

where α and β are webs, and E is an embedding of β. If we want to replace the subweb α of the web ω by another subweb β, it is necessary to specify how to "embed" β in ω in place of α. The definition of an embedding must not depend on the host web ω since we want to be able to replace α by β in any web containing α as a subweb. Usually E consists of a set of logical functions which specify whether or not each vertex of $\omega - \alpha$ is connected to each vertex of β.

[8] In a most general formulation, the contextual condition of the production is added.

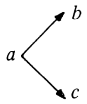
Example 1.6: Consider the web grammar

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

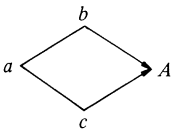
where

$$V_N=\{A\}, \quad V_T=\{a, b, c\}, \quad S=\{\dot{A}\}$$

and

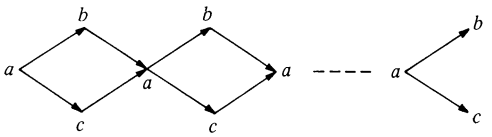
P: 1) $\dot{A}\rightarrow$ 

$E=\{(p, a)|(p, A)$ an edge in the host web$\}$

2) $\dot{A}\rightarrow$ 

E is the same as in(1).

The language of this grammar is the set of all webs of the form



It is noted that web grammars are vertex- or node-oriented compared with the branch- or edge-oriented grammars (e.g., PDL, plex grammars, etc.). That is, terminals or primitives are represented as vertices in the graph rather than as branches. Some applications of web grammars for picture description can be found in [1.74, 75].

An important special case of a web grammar is that in which the terminal set V_T consists of only a single symbol. In this case, every point of every web in the language has the same label, so that we can ignore the labels and identify the webs with their underlying graphs. This type of web grammar is called a "graph grammar", and its language is called graph language [1.76]. The formalism of web grammars has been rather extensively analyzed [1.72, 74]. The relations among PDL grammars, plex grammars and web grammars have been discussed by SHAW [1.73] and ROSENFELD [1.77].

PAVLIDIS [1.76, 78] has generalized string grammars to graph grammars by including nonterminal symbols which are not simple branches or nodes. An mth order nonterminal structure is defined as an entity which is connected to the rest of the graph by m nodes. In particular, a second-order structure is called a branch structure and a first-order structure a node structure. Then an mth-order context-free graph grammar G_g is a quadruple

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

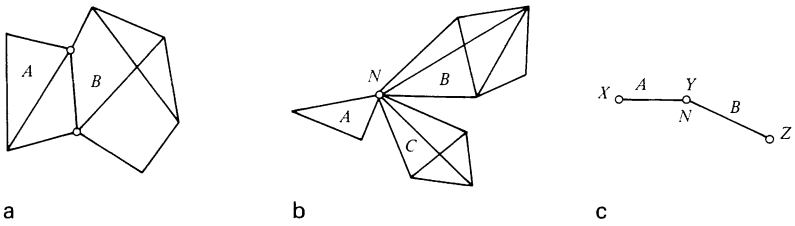


Fig. 1.8a–c. Illustration of (a) $A * B$; (b) $N(A+B+C)$; and (c) ANB

where

V_N is a set-of mth-order nonterminal structures: nodes, branches, triangles, ..., polygons with m vertices;

V_T is a set of terminals: nodes and branches;

P is a finite set of productions of the form $A \rightarrow \alpha$, where A is a nonterminal structure and α a graph containing possibly both terminals and nonterminals. α is connected to the rest of the graph through exactly the same nodes as A;

S is a set of initial graphs.

The expression $A * B$ denotes that the two graphs A and B are connected by a pair of nodes (Fig. 1.8a), and $N(A+B+C)$ denotes that the graphs A, B and C are connected through a common node N (Fig. 1.8b). Thus the production $A \rightarrow B * C$ where A, B and C are branch structures should be interpreted as: replace branch structure A by branch structures B and C connected to the graph-through the same nodes as A. No other connection exists between B and C. Similarly, the production $N \rightarrow M(A+B)$ should be interpreted as: replace node structure N by a node structure M and two other structures A and B connected to the rest of the graph by the same node as N. When no ambiguity occurs, we can use simple concatenations, e.g., ANB to denote a nonterminal subgraph consisting of a branch structure A with nodes X and Y connected to the node structure N through Y and a branch structure B with nodes Y and Z connected to N through Y (Fig. 1.8c). The subgraph is connected to the rest of the graph through the nodes X and Z.

The following examples illustrate the use of graph grammars for pattern description:

Example 1.7: The following grammar describes graphs representing two terminal series-parallel networks (TTSPN)

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

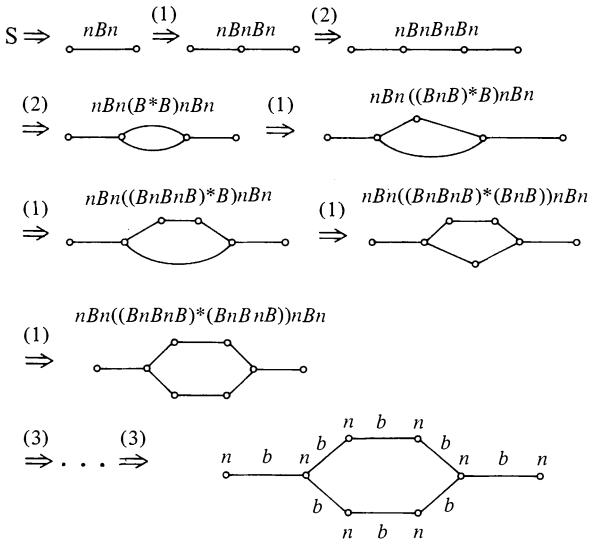
where

$$V_N = \{S, B\}, \qquad V_T = \{\overset{b}{-}, \overset{n}{|}\}, \qquad S = \{nBn\}$$

and

$$P\colon (1)\ B \to B\dot{n}B$$
$$(2)\ B \to B * B$$
$$(3)\ B \to b$$

A typical generation would be



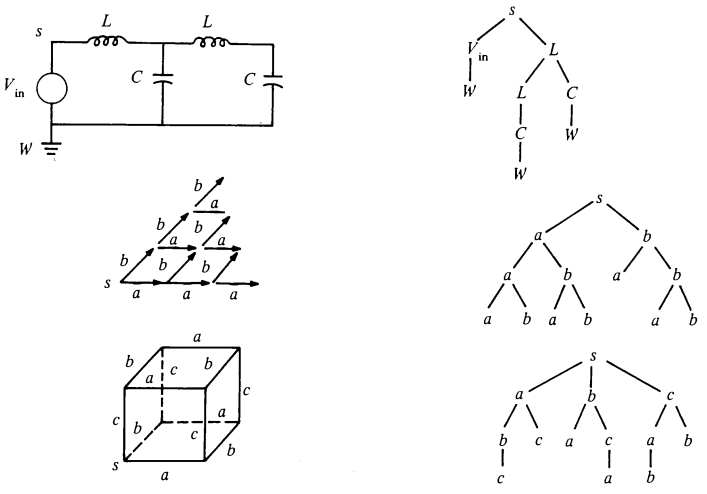
By extending one-dimensional concatenation to multi-dimensional concatenation, strings are generalized to trees. Tree grammars and the corresponding recognizers, tree automata, have been studied recently by a number of authors [1.79, 80]. Naturally, if a pattern can be conveniently described by a tree, it will easily be generated by a tree grammar. For example, in Fig. 1.9, patterns and their corresponding tree representations are listed in (a) and (b), respectively [1.81, 82]. GIPS has recently used a formal syntactic technique in the analysis of line drawings of three-dimensional objects [1.83].

Definition 1.4: A (regular) tree grammar G_t is a four-triple

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

where $\langle V, r' \rangle$ is a finite ranked alphabet such that $V_T \subseteq V$ and $r'|V_T = r$. The elements of V_T and $V - V_T = V_N$ are terminals and nonterminals, respectively. P is a finite set of productions of the form $\phi \to \psi$ where $\phi, \psi \in T_V$. $S \subseteq T_V$ is a finite set of axioms and T_V is a set of trees over $\langle V, r' \rangle$.

A generation or derivation $\alpha \Rightarrow \beta$ is in G_t if and only if there is a production $\phi \to \psi$ in P such that $\alpha|a = \phi$ and $\beta = (a \to \psi)\alpha$. That is, ϕ is a subtree of α at "a" and β is generated by replacing ϕ by ψ at "a". $\alpha \overset{*}{\Rightarrow} \beta$ is in G_t if and only if there



a b

Fig. 1.9. (a) Patterns; (b) Corresponding tree representations

exist trees $t_0, t_1, \ldots, t_m \in T_V$, $m \geqq 0$ such that

$$\alpha = t_0 \Rightarrow t_1 \Rightarrow \cdots \Rightarrow t_m = \beta$$

in G_t. The sequence $t_0, t_1, \ldots, t_m$ is called a derivation of β from α.

Definition 1.5: The language generated by tree grammar G_t is

$$L(G_t) = \{t \in T_{V_T} \mid \text{there exists } Y \in S \text{ such that } Y \overset{*}{\Rightarrow} t \text{ is in } G_t\}.$$

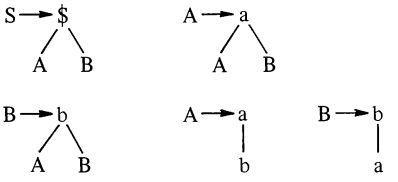
Example 1.8: The tree grammar

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

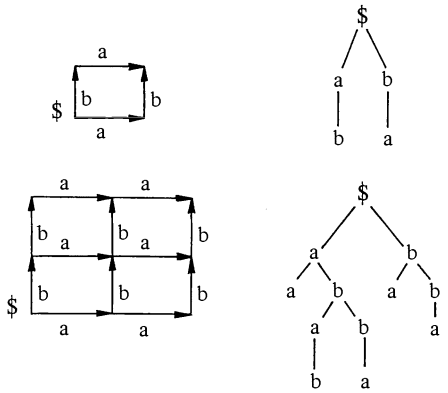
where

$$V = \{S, a, b, \$, A, B\}$$
$$V_T = \{\overset{a}{\rightarrow}, \uparrow b, .\$\}$$
$$r(a) = \{2,1,0\}, r(b) = \{2,1,0\}, r(\$) = 2$$

and P:



generates such patterns as



1.6 Syntax Analysis as Recognition Procedure

As is pointed out in Section 1.2, a parsing or syntax analysis is necessary if a complete description of the input is required for recognition. This requirement may be necessary due to the fact that the number of pattern classes is very large such as in a fingerprint recognition problem. It may also be necessary in the case that the complete description of each pattern will be stored for data retrieval purpose. In this section, syntax analysis for finite-state and context-free (string) languages will be briefly reviewed [1.84, 85]. Parsing of context-sensitive languages and web (and graph) languages is still an important topic for investigation. Regular tree languages are accepted by tree automata. The procedure of constructing a tree automaton to accept the language generated by a tree grammar is available [1.79–81].

1.6.1 Recognition of Finite-State Languages

Finite-state automata are known to recognize or accept finite-state languages [1.85]. If a class of patterns can be described by a finite-state language, a finite-state automaton can then be constructed to recognize the strings or sentences describing this class of patterns.

Definition 1.6: A nondeterministic finite-state automaton is a quintuple $(\Sigma, Q, \delta, q_0, F)$, where Σ is a finite set of input symbols (alphabet), Q is a finite set of states, δ is a mapping of $Q \times \Sigma$ into subsets of Q, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.

The interpretation of $\delta(q, a) = \{q_1, q_2, \ldots, q_l\}$ is that the automaton A, in state q, scanning a on its input tape, chooses any one of $q_1, \ldots, q_l$ as the next state and moves its input head one square to the right. The mapping δ can be extended from an input symbol to a string of input symbols by defining

$$\delta(q, \lambda) = \{q\}$$

$$\delta(q, xa) = \bigcup_{q_i \in \delta(q, x)} \delta(q_i, a), \qquad x \in \Sigma^* \quad \text{and} \quad a \in \Sigma .$$

Furthermore, we can define

$$\delta(\{q_1, q_2, \ldots, q_l\}, x) = \bigcup_{i=1}^{l} \delta(q_i, x).$$

When $\delta(q, a)$ consists of only a single state, the automaton is called a deterministic finite-state automaton.

A string x is accepted by A if there is a state p such that

$$p \in \delta(q_0, x) \quad \text{and} \quad p \in F.$$

The set of all strings accepted A is defined as

$$T(A) = \{x \mid p \in \delta(q_0, x) \text{ and } \quad p \in F\}.$$

Example 1.9: Given a nondeterministic finite-state automaton

$$A = (\Sigma, Q, \delta, q_0, F)$$

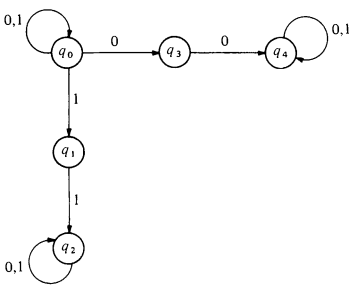
where

$$\Sigma = \{0,1\}$$
$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$
$$F = \{q_2, q_4\}$$

The state transition diagram of A is shown in Fig. 1.10. A typical sentence accepted A is 01011 since $\delta(q_0, 01011) = q_2 \in F$. In general, $T(A)$ is the set of all strings containing either two consecutive 0's or two consecutive 1's.

The relationship between deterministic finite-state automata and nondeterministic finite-state automata and the relationship between the finite-state languages and the sets of strings accepted by finite-state automata can be expressed by the following theorems.

Theorem 1.1: Let L be a set of strings accepted by a nondeterministic finite-state automaton $A = (\Sigma, Q, \delta, q_0, F)$. Then there exists a deterministic finite-state automaton $A' = (\Sigma', Q', \delta', q_0', F')$ that accepts L. The states of A' are all the subsets



$$\delta(q_0, 0) = \{q_0, q_3\}, \quad \delta(q_0, 1) = \{q_0, q_1\}$$
$$\delta(q_1, 0) = \phi, \quad \delta(q_1, 1) = q_2$$
$$\delta(q_2, 0) = q_2, \quad \delta(q_2, 1) = q_2$$
$$\delta(q_3, 0) = q_4, \quad \delta(q_3, 1) = \phi$$
$$\delta(q_4, 0) = q_4, \quad \delta(q_4, 1) = q_4$$

Fig. 1.10. Transition diagram for automaton A

of Q, i.e., $Q' = 2^Q$, and $\Sigma' = \Sigma$. F' is the set of all states in Q' containing a state of F. A state of A' will be denoted by $[q_1, q_2, \ldots, q_i] \in Q'$ where $q_1, q_2, \ldots, q_i \in Q$. $q_0 = [q_0]$. $\delta'([q_1, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$ if and only if $\delta(\{q_1, \ldots, q_i\}, a) = \bigcup_{k=1}^{i} \delta(q_k, a) = \{p_1, p_2, \ldots, p_j\}$.

Since the deterministic and nondeterministic finite-state automata accept the same sets of strings, we shall not distinguish between them unless it becomes necessary, but shall simply refer to both as finite-state automata.

Theorem 1.2: Let $G = (V_N, V_T, P, S)$ be a finite-state grammar. Then there exists a finite state automaton $A = (\Sigma, Q, \delta, q_0, F)$ with $T(A) = L(G)$, where

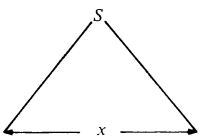
i) $\Sigma = V_T$,
ii) $Q = V_N \cup \{T\}$,
iii) $q_0 = S$,
iv) if P contains the production $S \to \lambda$, then $F = \{S, T\}$, otherwise, $F = \{T\}$,
v) the state T is in $\delta(B, a)$ if $B \to a$, $B \in V_N$, $a \in V_T$ is in P; and
vi) $\delta(B, a)$ contains all $C \in V_N$ such that $B \to aC$ is in P and $\delta(T, a) = \phi$ for each $a \in V_T$.

Theorem 1.3: Given a finite-state automaten $A = (\Sigma, Q, \delta, q_0, F)$. Then there exists a finite-state grammar $G = (V_N, V_T, P, S)$ with $L(G) = T(A)$, where

i) $V_N = Q$,
ii) $V_T = \Sigma$,
iii) $S = q_0$,
iv) $B \to aC$ is in P if $\delta(B, a) = C$, $B, C \in q$, $a \in \Sigma$; and
v) $B \to a$ is in P if $\delta(B, a) = C$ and $C \in F$.

1.6.2 Syntax Analysis of Context-Free Languages

When a context-free language is used to describe a class of patterns, the corresponding recognition device (or acceptor) is, in general, a nondeterministic, pushdown automaton. Not every nondeterministic pushdown automaton can have an equivalent deterministic pushdown automaton. Therefore, the process or the algorithm of performing the recognition, called "syntax analysis" or "parsing" is in general a nondeterministic procedure. The output of the syntax analyzer usually includes not only the decision of accepting the string generated by the given grammar, but also the derivation tree of the string which, in turn, gives the complete structural description of the pattern. Alternatively speaking, given a sentence x and a context-free (or context-free programmed) grammar G, construct a self-consistent derivation tree to fill the interior of the following triangle [1.84].



If the attempt is successful, $x \in L(G)$. Otherwise, $x \notin L(G)$.

It is in principle unimportant how we attempt to fill the interior of the triangle. We may do it from the top (the root of the tree) towards the bottom (called top-down parsing), or from the bottom toward the top (called bottom-up parsing). A number of top-down and bottom-up parsing algorithms have been developed [1.84, 85]. Unless for special classes of grammars, backtrackings are often required because of the nondeterministic nature of the parsing processes. In the following, a parsing algorithm for general context-free languages proposed by Earley is briefly presented [1.85].

Let $G=(V_N, V_T, P, S)$ be a context-free grammar and let $x=a_1a_2\ldots a_n$ be a string in V_T^*. An object of the form $[A\to X_1X_2\ldots X_k\cdot X_{k+1}\ldots X_m, i]$ is called an item for if $A\to X_1X_2\ldots X_m$ is a production in P. For each integer j, $0\leqq j\leqq n$, we can construct a list of items I_j such that $[A\to\alpha\cdot\beta, i]$ is in I_j for $0\leqq i\leqq j$ if and only if for some γ and δ we have

$$S\overset{*}{\Rightarrow}\gamma A\delta$$

where $\gamma\overset{*}{\Rightarrow}a_1\ldots a_i$, and $\alpha\overset{*}{\Rightarrow}a_{i+1}\ldots a_j$.

The sequence of lists $I_0, I_1, \ldots, I_n$ is called the parse lists for the string x. It is noted that $x\in L(G)$ if and only if there is some item of the form $[S\to\alpha\cdot, 0]$ in I_n.

The procedure of constructing the parse lists $I_0, \ldots, I_n$ for x consists of the following:

1. Construction of I_0
   1) If $S\to\alpha$ is a production in P, add $[S\to\cdot\,\alpha, 0]$ to I_0.
   2) Perform the following steps with $j=0$ until no new items can be added to I_0:
      a) If $[A\to\alpha\cdot B\beta, i]$ is in I_j and $B\to\gamma$ is a production in P, add $[B\to\cdot\,\gamma, j]$ to I_j.
      b) If $[A\to\alpha\cdot, i]$ is in I_j, then for all items in I_i of the form $[B\to\beta\cdot A\gamma, k]$ add $[B\to\beta A\cdot\gamma, k]$ to I_j.
2. Construction of I_j from I_{j-1}
   3) For all $[A\to\alpha\cdot a_j\beta, i]$ in I_{j-1}, add $[A\to\alpha a_j\cdot\beta, i]$ to I_j.
   4) Perform 2) to I_j.

1.7 Concluding Remarks

In this chapter, we have demonstrated that languages can be used to describe complex patterns. Consequently, syntax analysis procedures can be used to implement the pattern recognition process. It should be noted that, for many practical applications, often both syntactic and decision-theoretic approaches are used [1.12, 86]. For example, decision-theoretic approaches are usually effective in the recognition of pattern primitives. This is primarily due to the fact that the structural information of primitives is considered not important and the (local) measurements taken from the primitives are sensitive to noise and distortion. On the other hand, in the recognition of subpatterns and the pattern itself which are rich in structural information, syntactic approaches are therefore required.

In some practical applications, a certain amount of uncertainty exists in the process under study. For example, due to the presence of noise and variations

in the pattern measurements, ambiguities often occur in the languages describing real-data patterns. In order to describe and recognize noisy patterns under possible ambiguous situations, the use of stochastic languages has been recently suggested [1.87–93].

A stochastic grammar is a four-tuple $G_s=(V_N, V_T, P_s, S)$ where P_s is a finite set of stochastic productions and all the other symbols are the same as defined in Section 1.1. For a stochastic context-free grammar, a production in P_s is of the form

$$A_i \xrightarrow{p_{ij}} \alpha_j, \qquad A_i \in V_N, \qquad \alpha_j \in (V_N \cup V_T)^*$$

where p_{ij} is called the production probability. The probability of generating a string x, called the string probability $p(x)$, is the product of all production probabilities associated with the productions used in the generation of x. The language generated by a stochastic grammar consists of the strings generated by the grammar and their associated string probabilities.

By associating probabilities with the strings, we can impose a probabilistic structure on the language to describe noisy patterns. The probability distribution characterizing the patterns in a class can be interpreted as the probability distribution with the strings in a language. Thus, statistical decision rules can be applied to the classification of a pattern under ambiguous situations (for example, use the maximum-likelihood or Bayes decision rule). Furthermore, because of the availability of the information about production probabilities, the speed of syntactic analysis can be improved through the use of this information [1.93, 94]. Of course, in practice, the production probabilities will have to be inferred from the observation of relatively large numbers of pattern samples [1.54, 88].

Other approaches for the recognition of distorted or noisy patterns using syntactic methods include the use of transformational grammar [1.58] and the application of error-correcting parsing techniques [1.95]. In the use of error-correcting parsing as a recognition procedure, different types of primitive extraction error (substitution, deletion and insertion) are introduced. The original pattern grammar is modified by taking these errors into consideration. The recognition process is then based on the parser designed according to the modified grammar. The error-correcting capability of this class of parsers can be achieved by using either a minimum-distance or a maximum-likelihood decision criterion [1.96–99].

In specifying a selected primitive, a set of attributes is often required. A primitive (terminal) with different properties can be expressed in terms of its attribute values. This set of attributes may be considered as a semantic information of the primitive. Each attribute may be expressed by numerical functions or logical predicates. The semantic information of a subpattern (nonterminal) is, in general, evaluated either from the semantic information of the composed primitives according to the syntactic relations or operators and the semantic rules associated with each production of the grammar, or on the basis of a separate set of functions or rules which are not necessarily defined in conjunction with the productions of the grammar [1.12]. This class of grammars is sometimes called attribute grammars or "grammars with coordinates" [1.100–102].

References

1.1 K. S. Fu: *Sequential Methods in Pattern Recognition and Machine Learning* (Academic Press, 1968)

1.2 G. S. Sebestyen: *Decision Process in Pattern Recognition* (Macmillan, New York 1962)

1.3 N. J. Nilsson: *Learning Machines-Foundations of Trainable Pattern-Classifying System* (McGraw-Hill, 1965)

1.4 J. M. Mendel, K. S. Fu: *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications* (Academic Press, 1970)

1.5 W. Meisel: *Computer-Oriented Approaches to Pattern Recognition* (Academic Press, 1972)

1.6 K. Fukunaga: *Introduction to Statistical Pattern Recognition* (Academic Press, 1972)

1.7 E. A. Patrick: *Fundamentals of Pattern Recognition* (Prentice-Hall, 1972)

1.8 H. C. Andrews: *Introduction to Mathematical Techniques in Pattern Recognition* (Wiley, 1972)

1.9 R. O. Duda, P. E. Hart: *Pattern Classification and Scene Analysis* (Wiley, 1973)

1.10 C. H. Chen: *Statistical Pattern Recognition* (Hayden Book Company, Washington, D. C. 1973)

1.11 T. Y. Young, T. W. Calvert: *Classification, Estimation, and Pattern Recognition* (American Elsevier, 1973)

1.12 K. S. Fu: *Syntactic Methods in Pattern Recognition* (Academic Press, 1974)

1.13 W. F. Miller, A. C. Shaw: Proc. AFIPS Fall Joint Computer Conference (1968)

1.14 R. Narasimhan: Rpt. 121, (Digital Computer Laboratory, University of Illinois, Urbana, Illinois 1962)

1.15 Special Issues of PATTERN RECOGNITION on Syntactic Pattern Recognition, Vol. 3, No. 4, 1971 and Vol. 4, No. 1, 1972

1.16 N. V. Zavalishin, I. B. Muchnik: Automatika i Telemekhanika **86** (1969)

1.17 Ya. Z. Tsypkin: *Foundations of the Theory of Learning System* (Nauka, Moscow 1970)

1.18 M. A. Aiserman, E. M. Braverman, L. I. Rozonoer: *Potential Function Method in Theory of Learning Machines* (Nauka, Moscow 1970)

1.19 K. S. Fu: *Pattern Recognition and Machine Learning* (Plenum Press, 1971)

1.20 A. G. Arkadev, E. M. Braverman: *Learning in Pattern Classification Machines* (Nauka, Moscow 1971)

1.20a R. L. Grimsdale, F. H. Summer, C. J. Tunis, T. Kilburn: Proc. IEEE, Vol. 106, Part B, No. 26, March, 1959, pp. 210–221; reprinted in *Pattern Recognition* ed. by L. Uhr, (Wiley, 1966) pp. 317–338

1.21 M. Eden, M. Halle: *Proc. 4th London Symp. on Information Theory* (Butterworth, London 1961) pp. 287–299

1.22 L. D. Earnest: *Information Processing*, ed. by C. M. Poplewell (North Holland Publishing Co., Amsterdam 1963) pp. 462–466

1.23 P. Mermelstein, M. Eden: Information and Control **7**, 255 (1964)

1.24 M. Eden, P. Mermelstein: *Proc. 16th Ann. Conf. on Engineering in Medicine and Biology* (1963), pp. 12–13

1.25 H. Freeman: IEE Trans. *EC-10*, 260 (1961)

1.25a H. Freeman: Proc. National Electronics Conf. **18**, 312 (1962)

1.26 P. J. Knoke, R. G. Wiley: Proc. IEEE Comp. Conf., 142 (1967)

1.27 J. Feder: Information and Control **13**, 230 (1968)

1.28 H. Freeman, S. P. Morse: J. of Franklin Inst. **284**, 1 (1967)

1.29 J. Feder and H. Freeman: IEEE Intern. Convention Record, Part 3, 1966, pp. 69–85

1.30 C. T. Zahn: SLAC Rpt. 72 (Stanford Linear Accelerator Center, Stanford, Calif. 1966)

1.31 H. Freeman, J. Garder: IEEE Trans. **EC-13**, 118 (1964)

1.32 R. Narasimhan: Comm. ACM **9**, 166 (1966)

1.33 R. J. Spinrad: Information and Control **8**, 124 (1965)

1.34 J. F. O'Callaghan: "Problems in On-Line Character Recognition", *Picture Language Machines*, ed. by S. Kaneff (Academic Press, 1970)

1.35 A. C. Shaw: SLAC-Rpt. 84 (Stanford Linear Accelerator Center, Stanford, Calif. March, 1968)

1.36 B. K. Bhargava, K. S. Fu: Tech. Rpt. TR-EE 72–30, School of Electrical Engineering, Purdue University, West Lafayette, Ind. (1972)

1.37 M. Nir: Ph. D. thesis, Moore School of Electrical Engineering, University of Pennsylvania, (1967)

1.38 R. NARSIMHAN: Information and Control **7**, 151 (1964)
1.39 R. S. LEDLEY, L. S. ROTOLO, T. J. GOLAB, J. D. JACOBSEN, M. D. GINSBURG, J. B. WILSON: *Optical and Electro-Optical Information Processing*, ed. by J. T. TIPPETT, D. BECKOWITZ, L. CLAPP, C. KOESTER, A. VANDERBURGH, JR. (MIT Press, Cambridge, Mass. 1965), p. 591
1.40 J. W. BUTLER, M. K. BUTLER, A. STROUD: "Automatic Classification of Chromosomes", Proc. Conf. on Data Acquisition and Processing in Biology and Medicine, New York (1963)
1.41 H. C. LEE, K. S. FU: "A Syntactic Pattern Recognition System with Learning Capability", *Proc. COINS-72* (1972)
1.42 M. NAGAO: "Picture Recognition and Data Structure", in *Graphic Languages*, ed. by F. NAKE, A. ROSENFELD (North-Holland Publishing Co., Amsterdam, London 1972)
1.43 M. D. KELLEY: Ph. D. thesis, Dept. of Computer Science, Stanford University, Stanford, Calif. (1970)
1.44 L. G. ROBERTS: *Optical and Electro-Optical Information Processing*, ed. by J. T. TIPPETT, D. BECKOWITZ, L. CLAPP, C. KOESTER, A. VANDERBURGH, JR. (MIT Press, Cambridge, Mass. 1965), p. 159
1.45 R. O. DUDA, P. E. HART: "Experiments in Scene Analysis", Proc. 1st Nat. Symp. on Industrial Robots, Chicago (1970)
1.46 J. A. FELDMAN, G. M. FELDMAN, G. FALK, G. GRAPE, J. PEARLMAN, I. SOBEL, J. M. TENENBAUM: "The Stanford Hand-Eye Project", Proc., 1st Intern. Joint Conf. on Artificial Intelligence, Washington D. C., (1969)
1.47 T. PAVLIDIS: Pattern Recognition **1**, 165 (1968)
1.48 A. ROSENFELD, J. P. STRONG: "A Grammar for Maps", *Software Engineering* (**2** ed. by J. T. TOU, Academic Press 1971)
1.49 T. PAVLIDIS: Pattern Recognition **4**, 5 (1972)
1.50 T. PAVLIDIS: "Structural Pattern Recognition: Primitives and Juxtaposition", *Frontiers of Pattern Recognition*, ed. by S. WATANABE (Academic Press, 1972)
1.51 M. L. MINSKY, S. PAPERT: Project MAC Progress Rpt. IV (MIT Press, Cambridge, Mass. 1967)
1.52 A. GUZMAN: Proc. AFIPS FJCC **33**, Pt. 1, 291 (1968)
1.53 C. R. BRICE, C. L. FENNEMA: Artificial Intelligence **1**, 205 (1970)
1.54 K. S. FU, T. L. BOOTH: "Grammatical Inference—Introduction and Survey". IEEE Trans. *SMC-5*, 95 and 409 (1975)
1.55 R. NARASIMHAN: "On the Description, Generation, and Recognition of Classes of Pictures", in *Automatic Interpretation and Classification of Images*, ed. by A. GRASSELLI (Academic Press, 1969)
1.56 J. FEDER: Information Sciences **3**, 225 (1971)
1.57 A. W. LAFFAN, R. C. SCOTT: "A New Tool for Automatic Pattern Recognition: A Context-Free Grammar for Plane Projective Geometry", Proc. Ind. Intern. Joint Conf. on Pattern Recognition, Lyngby-Copenhagen, Denmark, 1974
1.58 M. C. CLOWES: "Transformational Grammars and the Organization of Pictures", in *Automatic Interpretation and Classification of Images*, ed. by A. GRASSELLI (Academic Press, 1969)
1.59 LAVEEN KANAL, B. CHANDRASEKARAN: "On the Linguistic, Statistical and Mixed Models for Pattern Recognition", in *Frontiers of Pattern Recognition*, ed. by S. WATANABE (Academic Press, 1972)
1.60 R. NARSIMHAN: "Picture Languages", in *Picture Language Machines*, ed. by S. KANEFF (Academic Press, 1970)
1.61 W. E. UNDERWOOD, L. N. KANAL: "Structural Description, Transformational Rules and Pattern Analysis", Proc. 1st Intern. Joint on Pattern Recognition, Washington, D.C. (1973)
1.62 M. L. MINSKY: Proc. IRE **49**, 8 (1961)
1.63 T. G. EVANS: "A Formalism for the Description of Complex Objects and Its Implementation", Proc. 5th Intern. Congress on Cybernetics, Namur, Belgium (1967)
1.64 M. B. CLOWES: "Pictorial Relationships—A Syntactic Approach", in *Machine Intelligence IV*, ed. by B. MELTZER, D. MICHIE (American Elsevier, New York 1969)
1.65 T. G. EVANS: "Descriptive Pattern Analysis Techniques", in *Automatic Interpretation and Classification of Images*, ed. by A. Grasselli (Academic Press, 1969)
1.66 H. G. BARROW, J. R. POPPLESTONE: *Machine Intelligence 6*, ed. by B. MELTZER, D. MICHIE (Edinburgh University Press, 1971) pp. 377–396
1.67 R. A. KIRSCH: IEEE Trans. **EC-13**, 363 (1964)
1.68 M. F. DACEY: Pattern Recognition **2**, 11 (1970)

1.69 D.M. MILGRAM, A. ROSENFELD: **IFIP Congress 71**, (North-Holland, Amsterdam 1971) pp. 166–173
1.70 A.C. SHAW: Information and Control **14**, 9 (1969)
1.71 A.C. SHAW: J.ACM **17**, 453 (1970)
1.72 J.L. PFALTZ, A. ROSENFELD: Proc. 1st Intern. Joint Conf. on Artificial Intelligence, Washington D.C. (1969), pp. 609–619
1.73 A.C. SHAW: "Picture Graphs, Grammars, and Parsing", in *Frontiers of Pattern Recognition*, ed. by S. WATANABE (Academic Press, 1972)
1.74 J.L. PFALTZ: Tech. Rpt. 70–138, Computer Science Center, University of Maryland, College Park, Md. (1970)
1.75 J.M. BRAYER, K.S. FU: Rpt. 75–1, School of Electrical Engineering, Purdue University, W. Lafayette, Ind. (1975)
1.76 T. PAVLIDIS: Jour. of ACM **19**, 11 (1972)
1.77 A. ROSENFELD: "Picture Automata and Grammars: An Annotated Bibliography", Proc. Symp. on Computer Image Processing and Recognition, Columbia, Mo. (1972), Vol. 2
1.78 T. PAVLIDIS: "Graph Theoretic Analysis of Pictures", in *Graphic Languages*, ed. by F. NAKE and A. ROSENFELD (North-Holland, Amsterdam 1972)
1.79 W.S. BRAINERD: Information and Control **14**, 217 (1969)
1.80 J.E. DONAR: "Tree Acceptors and Some of Their Applications", J. of Computer and System Sciences **4**, (1970)
1.81 K.S. FU, B.K. BHARGAVA: IEEE Trans. **C22**, 1087 (1973)
1.82 B.K. BHARGAVA, K.S. FU: "Transformation and Inference of Tree Grammars for Syntactic Pattern Recognition", Proc. IEEE Intern. Conf. on Systems, Man and Cybernetics, Dallas, Tex. (1974)
1.83 J. GIPS: Pattern Recognition **6**, 189 (1974)
1.84 J.M. FOSTER: *Automatic Syntactic Analysis*, (American Elsevier 1970)
1.85 A.V. AHO, J.D. ULLMAN: *The Theory of Parsing, Translation, and Compiling*, Vol. 1, Parsing (Prentice-Hall 1972)
1.86 F.W. BLACKWELL: "Combining Mathematical and Structural Pattern Recognition", Proc. 2nd Intern. Joint Conf. on Pattern Recognition, Copenhagen, Denmark (1974)
1.87 V. GRENANDER: Tech. Rpt., Division of Applied Math., Brown University, Providence, R.I.
1.88 K.S. FU: "Syntactic Pattern Recognition and Stochastic Languages", in *Frontiers of Pattern Recognition*, ed. by S. WATANABE (Academic Press 1972)
1.89 V.A. KOVALEVSKY: "Sequential Optimization in Pattern Recognition and Pattern Description", *Proc.* IFIP *Congress*, Amsterdam (1968)
1.90 K.S. FU: Computer Graphics and Image Processing **2**, 433 (1973)
1.91 L.W. FUNG, K.S. FU: "Stochastic Syntactic Classification of Noisy Patterns", Proc. 2nd Intern. Joint Conf. on Pattern Recognition, Copenhagen, Denmark (1974)
1.92 V. DIMITROV: "Multilayered Stochastic Languages for Pattern Recognition", Proc. 1st Intern. Joint Conf. on Pattern Recognition, Washington, D.C. (1973)
1.93 H.C. LEE, K.S. FU: "A Stochastic Syntax Analysis Procedure and Its Application to Pattern Classification", IEEE Trans. **C-21**, 660 (1972)
1.94 T. HUANG, K.S. FU: Computer Graphics and Image Processing, **1**, 257 (1972)
1.95 A.V. AHO, T.G. PETERSON: "A Minimum Distance Error-Correcting Parser for Context-Free Languages", SIAM J. Compt., **1**, No. 4, 305 (December, 1972)
1.96 L.W. FUNG, K.S. FU: IEEE Trans. **C-24**, 662 (1975)
1.97 M.G. THOMASON, R.C. GONZALEZ: "Classification of Imperfect Syntactic Pattern Structures", Proc. 2nd Intern. Joint Conf. on Pattern Recognition, Copenhagen, Denmark (1974)
1.98 K.S. FU: "Error-Correcting Parsing for Syntactic Pattern Recognition", in *Data Structures in Pattern Recognition and Computer Graphics*, ed. by A. KLINGER, K.S. FU, T. KUNII (Academic Press, 1977)
1.99 M.G. Thomason: IEEE Trans. **C-24**, 1211 (1975)
1.100 D.E. KNUTH: *Math. Syst. Theory* **2**, 127 (1968)
1.101 W.T. WILSON: "Formal Semantic Definition Using Synthesized and Inherited Attributes", in *Formal Semantics of Programming Languages*, ed. by R. RUSTIN (Prentice-Hall, 1972)
1.102 D.L. MILGRAM, A. ROSENFELD: "A Note on Grammars with Coordinates", in *Graphic Languages*, ed. by F. NAKE, A. ROSENFELD (North-Holland Publ. 1972)

2. Peak Recognition in Waveforms

S. L. HOROWITZ

With 4 Figures

A new system of peak component recognition and measurement in digitized waveforms is detailed. Two input parameters identify waveform context (scale and noise content), and a third specifies baseline tolerance (if applicable). The input waveform is preprocessed by a discrete linear piecewise approximation algorithm yielding a segmentation in endpoint/slope/constant format. Slope values are encoded as symbols of a string which is parsed by a syntax-directed finite-state automaton. One of three different machines and sets of semantic routines is chosen depending upon whether the waveform is baseline-free, unipolar, or bipolar. In the last case, the endpoint values relative to the baseline are encoded as symbols of a second string which modifies the action of the machine. An electrocardiogram is employed as an example waveform to demonstrate the bipolar algorithm, which runs with sufficient speed to allow real-time processing. A proposed on-line implementation of the system is outlined.

The extraction of selected waveform features and subsequent comparison of these features with predetermined standards are crucial factors in the monitoring and classification of a wide range of phenomena in the natural sciences. Although these phenomena are manifested by diverse activities such as physical motion, chemical interaction or electromagnetic generation, they usually are reduced to a set of one-dimensional functions to permit more facile and precise observation. Units of the abscissa (independent variable) represent time, and units of the ordinate (dependent variable) represent voltage which may correspond to a different quantity of the phenomenon (e.g., centimeters of displacement, decibels of sound, lumens of light).

Much of the useful information contained in such a function is obtained by locating peaks, measuring their amplitudes and durations, and determining their directions, shapes, and juxtapositions. Research in recent years attests to the important role of peak detection in various pattern recognition media such as digitized terrain profiles [2.1], picture rasters [2.2], chromatograms [2.3,4], speech [2.5,6], electroencephalograms [2.3,7–11], blood pressure (pulse) waves [2.7, 12, 13], and electrocardiograms [2.7, 12, 14–22].

In many of the references, analog, digital, or hybrid systems incorporate differentiation, threshold comparison, and zero-crossing detection in a naive fashion to mark possible peak components [2.4, 5, 7, 9, 11, 12, 15, 16, 18–20]. Ad hoc schemes employed by others are restricted to specific waveform classes where a priori knowledge about the locations and characteristics of peak complexes is available [2.3, 6–8, 10, 12–14, 20, 22]. Often, researchers prefer to transform the waveform space from the time domain to the frequency and quefrency domains as a

means of data compression and feature enhancement. For the same reasons, waveforms are decomposed as weighted sums of (possibly orthogonal) functions through series expansion and eigenvector operations to yield Fourier, Walsh, or Karhunen-Loève coefficients, principle factors, and intrinsic or latent components. In many cases, individual peaks in the new transform spaces relate to certain feature groups in the original [2.7, 17, 21, 23, 24].

This chapter elaborates upon a new approach to general waveform peak recognition initially outlined in [2.25]. A revised two-part technique is presented here in more detail. The first part performs a piecewise linear approximation of the input waveform which is encoded as a string of symbols, and the second part parses the string to recognize the peaks. Several parameters indirectly specify the waveform environment (noise content, time scale, amplitude range, and possibly baseline location) eliminating the need for preliminary normalization. No other information about the source or structure of the waveform is necessary—any of the types cited above may serve as input (with the appropriate parameter choices). The final section in the chapter demonstrates the applicability of the algorithm as a viable analytic tool by correctly selecting the significant peaks in a standard electrocardiogram lead.

2.1 What is a Peak?

2.1.1 Mathematical Definition

Let a waveform be represented by a discrete set of points (x_1, y_1), (x_2, y_2), ..., (x_i, y_i), ..., (x_n, y_n) representing the analog function $y = f(x)$, with $x_i < x_{i+1}$ for all $i = 1, \ldots, n$. A peak, in the mathematical sense, is a feature of this waveform denoted by the triple of point pairs $[(x^l, y^l), (x^m, y^m), (x^r, y^r)]$. The pair (x^m, y^m) is the peak extremum, (x^l, y^l) is the left peak boundary, (x^r, y^r) is the right peak boundary and $x^l < x^m < x^r$. Define $[(x^l, y^l), (x^m, y^m), (x^r, y^r)]$ to be a positive peak where (x^m, y^m) is a local maximum if and only if the following expressions are satisfied simultaneously:

$$\forall i | x^l \leqq x_i \leqq x^r \ (y^m \geqq y_i), \tag{2.1}$$

$$\forall i | x^l \leqq x_i \leqq x^m \ (y_i \leqq y_{i+1}), \tag{2.2}$$

$$\forall i | x^m \leqq x_i \leqq x^r \ (y_i \geqq y_{i+1}), \tag{2.3}$$

$$\exists i | x_i = x^l \ (y_{i-1} \geqq y^l), \tag{2.4}$$

$$\exists i | x_i = x^r \ (y_r \leqq y_{i+1}). \tag{2.5}$$

Expression (2.1) describes the local maximum property, (2.2,3) establish the left non-negative and right non-positive slopes required for the two sides of the local maximum (also preventing the overlapping of two positive peaks) and (2.4, 5) yield the left and right positive peak boundary properties. A negative peak where (x^m, y^m) is a local minimum can be defined similarly by changing the direction of the inequalities inside the parentheses of (2.1–5).

Note that as the definition stands, x^m may not be unique for fixed x^l and x^r, and vice versa, as the enclosed inequalities allow the extrema and boundaries

to be anywhere on horizontal lines. However, if these inequalities were made strict, then (2.1), or (2.2) and (2.4), or (2.3) and (2.5) would be unsatisfiable for any peak containing horizontal lines. Thus three more expressions are required to fix x^{m} as the midpoint of a flat positive peak, x^{l} as the left endpoint and x^{r} as the right endpoint of a flat positive peak boundary

$$\exists i,j|x_i \neq x_j,\ y_i=y_j=y^{\mathrm{m}}\ \{(y_{i-1}<y_i)\wedge(y_i>y_{j+1})\wedge x^{\mathrm{m}}=(x_i+x_j)/2]\}\,, \tag{2.6}$$

$$\exists i|x_i=x^{\mathrm{l}}\ (y^{\mathrm{l}}<y_{i+1})\,, \tag{2.7}$$

$$\exists i|x_i=x^{\mathrm{r}}\ (y_{i-1}>y^{\mathrm{r}})\,. \tag{2.8}$$

Two final expressions are needed to eliminate multiple occurrences of x^{l} and x^{r} (thereby preventing the nesting of two positive peaks) by forcing a peak to be as wide as possible.

$$\exists i,j|x_i=x^{\mathrm{l}},\ x_j\leqq x^{\mathrm{l}},\ y_j=y^{\mathrm{l}}\ [(y_{j-1}>y_j)\rightarrow\forall k|x_j\leqq x_k\leqq x_i\ (y_k=y^{\mathrm{l}})]\,, \tag{2.9}$$

$$\exists i,j|x_i=x^{\mathrm{r}},\ x_j\geqq x^{\mathrm{r}},\ y_j=y^{\mathrm{r}}\ [(y_j<y_{j+1})\rightarrow\forall k|x_i\leqq x_k\leqq x_j\ (y_k=y^{\mathrm{r}})]\,. \tag{2.10}$$

Again, the same arguments pertain to negative peaks if the resulting inequalities enclosed in parentheses are reversed in (2.6–10).

An often-used criterion for detecting peak extrema is derivable from this model. Denote the first difference of a waveform by $d_1, d_2, \ldots, d_i, \ldots, d_{n-1}$ where

$$d_i=(y_{i+1}-y_i)/(x_{i+1}-x_i) \tag{2.11}$$

corresponds to f', the first derivative (slope) of the analog function. By (2.1–3, 6), the first difference reverses sign (possibly through zero) if and only if a local extremum has been encountered.

$$\exists i,j\,\mathrm{sign}(d_i)\neq\mathrm{sign}(d_j)\ \{[(i+1=j)\vee\forall k|i<k<j(d_k=0)]$$
$$\leftrightarrow\exists k|x_k=x^{\mathrm{m}}\ [x_k=(x_{i+1}+x_j)/2]\}\,. \tag{2.12}$$

2.1.2 Syntactical Definition

Assign to the $i^{-\mathrm{th}}$ pair of waveform points $[(x_i, y_i), (x_{i+1}, y_{i+1})]$ the symbol ω_i connoting the slope characteristic of the line segment joining the two points. Let $w=\omega_1\omega_2\ldots\omega_i\ldots\omega_{n-1}$ be the resulting string encoding of the waveform (with respect to first difference) such that

$$(\omega_i=/)\leftrightarrow(d_i>0)\,, \tag{2.13}$$

$$(\omega_i=\backslash)\leftrightarrow(d_i<0)\,, \tag{2.14}$$

$$(\omega_i=0)\leftrightarrow(d_i=0)\,, \tag{2.15}$$

where "/" denotes "positive slope", "\" denotes "negative slope", and "0" denotes "zero slope".

Regular expressions [2.26] over the alphabet $\{/, \backslash, 0\}$ may be constructed to denote infinite sets of substrings of $w \in W = \{/, \backslash, 0\}^*$ representing positive and negative peaks. The following notation is adopted: If P and Q are regular expressions, then

1) "$P|Q$" is the union of the sets denoted by P and Q;
2) "PQ" is the cartesian cross-product of the sets denoted by P and Q; and
3) "P^*" is the reflexive-transitive closure of the set denoted by P.

Redundant parentheses are removed from the expressions by giving "*" the highest precedence, then concatenation, and then "|".

By (2.7) the left side of a positive peak must start with the symbol "/"; by (2.2), it may contain any number (possibly zero) of the symbols "/" and "0" only; and by (2.1), it must end with the symbol "/". A regular expression for the left side of a positive peak is given by

$$P = /|/(/|0)^*/ . \tag{2.16}$$

Likewise, the right side of a positive peak must begin and end with the symbol "\" and contain any number of the symbols "\" and "0" only, by (2.1, 3, 8). A regular expression for the right side of a positive peak is given by

$$Q = \backslash|\backslash(\backslash|0)^*\backslash . \tag{2.17}$$

Combining (2.16) and (2.17) with the positive extremum (represented by 0^*) yields a complete positive peak given by

$$R = P0^*Q . \tag{2.18}$$

Similarly, a complete negative peak is given by

$$S = Q0^*P . \tag{2.19}$$

By examining (2.4, 5, 9, 10) together with their duals, the left (right) side of a positive peak is the right (left) side of a negative peak, and vice versa. Thus, either a waveform contains either one peak or no peaks, or the peaks must form an uninterrupted alternating sequence, i.e.,

$$W = 0^* \ldots R0^*S0^*R0^*S \ldots 0^* . \tag{2.20}$$

With the aid of (2.16–20) it is possible to construct a deterministic context-free grammar [2.26, 27] G that recognizes positive and negative peaks (if any) in a waveform represented by a string w:

$$G = (V_N, V_T, P, \{W\}) \tag{2.21}$$

where

$$V_N = \{W, \langle\text{wave}\rangle, \langle\text{peak}^+\rangle, \langle\text{peak}^-\rangle, \langle\text{pos}_1\rangle, \langle\text{neg}_1\rangle, \langle\text{pos}_2\rangle, \langle\text{neg}_2\rangle, \langle\text{zero}\rangle\} \tag{2.22}$$

is the set of nonterminals;

$$V_T = \{/, \backslash, 0\} \tag{2.23}$$

is the set of terminal symbols;

$$\begin{aligned}
P = \{ & W \to \langle \text{zero} \rangle \langle \text{wave} \rangle \langle \text{zero} \rangle , \\
& W \to \langle \text{zero} \rangle \langle \text{wave} \rangle , \\
& W \to \langle \text{wave} \rangle \langle \text{zero} \rangle , \\
& W \to \langle \text{wave} \rangle , \\
& W \to \langle \text{zero} \rangle , \\
& \langle \text{wave} \rangle \to \langle \text{peak}^+ \rangle , \\
& \langle \text{wave} \rangle \to \langle \text{peak}^- \rangle , \\
& \langle \text{wave} \rangle \to \langle \text{pos}_1 \rangle , \\
& \langle \text{wave} \rangle \to \langle \text{neg}_1 \rangle , \\
& \langle \text{peak}^+ \rangle \to \langle \text{peak}^- \rangle \langle \text{zero} \rangle \langle \text{neg}_1 \rangle , \\
& \langle \text{peak}^+ \rangle \to \langle \text{peak}^- \rangle \langle \text{neg}_1 \rangle , \\
& \langle \text{peak}^+ \rangle \to \langle \text{pos}_1 \rangle \langle \text{zero} \rangle \langle \text{neg}_1 \rangle , \\
& \langle \text{peak}^+ \rangle \to \langle \text{pos}_1 \rangle \langle \text{neg}_1 \rangle , \\
& \langle \text{peak}^- \rangle \to \langle \text{peak}^+ \rangle \langle \text{zero} \rangle \langle \text{pos}_1 \rangle , \\
& \langle \text{peak}^- \rangle \to \langle \text{peak}^+ \rangle \langle \text{pos}_2 \rangle , \\
& \langle \text{peak}^- \rangle \to \langle \text{neg}_1 \rangle \langle \text{zero} \rangle \langle \text{pos}_1 \rangle , \\
& \langle \text{peak}^- \rangle \to \langle \text{neg}_1 \rangle \langle \text{pos}_1 \rangle , \\
& \langle \text{pos}_1 \rangle \to \langle \text{pos}_1 \rangle / , \\
& \langle \text{pos}_1 \rangle \to \langle \text{pos}_2 \rangle / , \\
& \langle \text{pos}_1 \rangle \to / , \\
& \langle \text{neg}_1 \rangle \to \langle \text{neg}_1 \rangle \backslash , \\
& \langle \text{neg}_1 \rangle \to \langle \text{neg}_2 \rangle \backslash , \\
& \langle \text{neg}_1 \rangle \to \backslash , \\
& \langle \text{pos}_2 \rangle \to \langle \text{pos}_1 \rangle 0 , \\
& \langle \text{pos}_2 \rangle \to \langle \text{pos}_2 \rangle 0 , \\
& \langle \text{neg}_2 \rangle \to \langle \text{neg}_1 \rangle 0 , \\
& \langle \text{neg}_2 \rangle \to \langle \text{neg}_2 \rangle 0 , \\
& \langle \text{zero} \rangle \to \langle \text{zero} \rangle 0 , \\
& \langle \text{zero} \rangle \to 0 \}
\end{aligned} \tag{2.24}$$

is the set of productions; and W is the start symbol. When parsing a specific w, a positive peak is recognized if and only if a section of w is completely reduced by some production to the nonterminal $\langle \text{peak}^+ \rangle$, by (2.12). The same relation holds between a negative peak and the nonterminal $\langle \text{peak}^- \rangle$.

2.2 Practical Peak Recognition

2.2.1 Signal and Noise

In most practical situations, the input waveform is converted from a sampled analog signal containing noise. Quantization, extraneous electromagnetic radiation, and high gain amplification are well known sources of noise, in addition to other interference engendered by a particular environment. Indeed, one of the persistent problems in the field of waveform analysis is the separation of signal from noise. Of course, if the characteristics of the signal or the noise components are not understood beforehand, then they are so designated to obtain a high signal-to-noise ratio (i.e., the minimum peak height of the signal is greater than the maximum peak height of the noise).

Low frequency noise shifts the baseline, causing the signal to rise and fall (discussed later in this section). High frequency noise appears as small spikes, often randomly distributed throughout the original signal. Unfortunately, these spikes satisfy expressions (2.1–12).

One simple answer is to furnish two more restrictive criteria:

$$\min(|y^{\mathrm{m}} - y^{\mathrm{l}}|, |y^{\mathrm{m}} - y^{\mathrm{r}}|) \geqq c_{\mathrm{a}} \tag{2.25}$$

$$(x^{\mathrm{r}} - x^{\mathrm{l}}) \geqq c_{\mathrm{d}} \tag{2.26}$$

where c_{a} and c_{d} are constants of amplitude and duration, respectively. For appropriate choices of c_{a} and c_{d} (depending upon the waveform) peaks of insufficient height and width fail to satisfy (2.25, 26); thus high frequency noise is rejected. Actually, this idea is adapted directly to implement "logical filtering" [2.1, 2, 28], however signal peak boundaries may not satisfy (2.2–5, 7–10) unless nested candidate peaks are catalogued in advance.

An alternative is to smooth and reduce the waveform before searching for peaks, thereby decreasing the number of individual points and spurious peaks to be considered. Among the most common methods are bandpass filtering [2.7, 29–32], series expansion/reconstruction [2.7, 21, 32–34], and piecewise polynomial approximation [2.35–39]. Analog and digital filters suppress noise, but at the expense of attenuating portions of the signal (e.g., local extrema) and without providing any compaction. Series expansion/reconstruction is suitable for waveforms with relevant features of relatively short extent and where different sections of the domain are unrelated.

Piecewise polynomial approximation has many advantages, the foremost being that any waveform can be approximated without respect to or change in the size and distribution of its features. The method blends local and global traits to adjust automatically to specific contours. Piecewise linear (first order

spline) approximation [2.14, 24, 28, 37, 38] was chosen for peak detection purposes since peaks, like the resulting approximation, are first order entities [see Eq. (2.12)]. In particular, small peaks are eliminated indirectly (c_a and c_d are not specified), retaining only those with sufficient amplitude and duration. Also if no continuity constraints are imposed at the breakpoints (knots, joints) processing time is linear in n the size of the input point set (the best series expansion algorithms run in $n \cdot \log(n)$ time). Furthermore, discontinuities in the original waveform are maintained at the breakpoints in the approximation. In general, the method preserves peak integrity to a better degree than conventional filtering and achieves a higher compaction ratio than series reconstruction.

2.2.2 Piecewise Linear Approximation Preprocessor

In this implementation, the piecewise linear approximation routine serving as the preprocessor is a variation of the "split-and-merge" algorithm [2.38] briefly outlined below:

Step 1. Divide the waveform into an arbitrary number of equal length segments with coincident endpoints, and calculate the error of a linear mean square approximation for each segment.

Step 2. Split segments which yield the greatest decrease in the sum of the squares of the segment error values (E) until E is less than a specified error tolerance (ETOL).

Step 3. Merge adjacent segments which yield the smallest increase in E while E is less than ETOL.

Step 4. Adjust the segment endpoints (breakpoints) to minimize E, then go to step 3 if any adjustments are made.

The choice of a mean square error norm was made for several reasons. Define a discrete, one-dimensional, linear error norm [2.39] measured along the y-coordinate to be

$$L_k = \left[\sum_i |y_i - (ax_i + b)|^k\right]^{1/k} \tag{2.27}$$

where a and b are the coefficients (slope and constant) of the approximating line segment to the point set $\{(x_i, y_i)\}$. Values of $k = 1, 2, \infty$ are widely discussed in the approximation theory literature. Minimizing L_1, the median error norm [2.36], yields

$$\min_{a,b} \left[\sum_i |y_i - (ax_i + b)|\right] \tag{2.28}$$

which is the most robust of the three norms, since outlying points (those farthest from the line) have a bounded effect on the result. Unfortunately, the a and b satisfying (2.28) are not unique. Minimizing L_∞, the minimax error norm [2.36, 37] yields

$$\min_{a,b} \{\max_i [|y_i - (ax_i + b)|]\} \tag{2.29}$$

which is the most sensitive to outlying points and although unique (if no two x_i are equal), requires an iterative linear program to compute the optimal a and b satisfying (2.29).

Minimizing L_2, the mean (least, integral) square error norm [2.37, 39, 40] yields

$$\min_{a,b}\left\{\sum_i [y_i-(ax_i+b)]^2\right\} \tag{2.30}$$

which compromises between the robustness of (2.28) and the sensitivity of (2.29). Furthermore, there exist unique closed forms for a and b satisfying (2.30):

$$a=\left(n\sum_i x_iy_i-\sum_i x_i\sum_i y_i\right)\Big/\left[n\sum_i x_i^2-\left(\sum_i x_i\right)^2\right], \tag{2.31}$$

$$b=\frac{1}{n}\left(\sum_i y_i-a\sum_i x_i\right) \tag{2.32}$$

with a undefined if and only if all of the x_i are equal. Since y is a function of x, this cannot occur. Thus an orthogonal mean square (eigenvector) fit [2.24, 38] is not adopted here. Also, the L_2 norm measured along the y-coordinate is invariant for any $x'=cx$ (change in x-scale) and linearly proportional to c for $y'=cy$ (change in y-scale). The sum of the squares of the segment error values is minimized to be compatible with the L_2 norm.

Finally, the method does not guarantee that a global minimum solution (with respect to the number of segments and placement of breakpoints for a given ETOL) is attained. However, the probability is very high that peak extrema and boundaries (a subset of the points of maximum curvature) are breakpoints in the set of local optima. Although the size of the initial segmentation determines the number of splits, merges and adjustments, the size of the ultimate segmentation is affected only slightly. The result mainly depends upon ETOL, which controls the resolution of the fit. A lower (higher) value of ETOL corresponds to lower (higher) values for c_a and c_d in (2.25, 26). Thus, ETOL can be chosen so that c_a and c_d are larger than the dimensions of the noise peaks, but smaller than the dimensions of the feature peaks.

This fact may be used to estimate a (possibly shifting) baseline for waveforms in the absence of a reliable measure, especially if low frequency noise intrudes. A trick is to treat the baseline as signal and the feature peaks as high frequency noise by first approximating the waveform with a very large value of ETOL, thereby producing a set of baseline coefficients. In the event that the waveform is unipolar, the constants of the baseline segments should be adjusted equally so that the resulting baseline is appropriately above or below the entire waveform. If the waveform is bipolar but the signal is known to be distributed asymmetrically about the mean amplitude, better statistics for the constants may be computed using percentile or histogram approaches after compensating for the slopes of the baseline segments as computed above [2.41]. In any case, baseline segments, whether calculated by the above heuristics or precomputed elsewhere, should be subtracted pointwise from the raw data to yield a new waveform with a baseline having zero slope and constant.

2.2.3 Peak Recognition Algorithms

The output of the preprocessor consists of a compacted waveform (x_1, y_1), $(x_2, y_2), \ldots, (x_j, y_j), \ldots, (x_{ns+1}, y_{ns+1})$ and the set of segment linear approximation coefficients $\{(a_j, b_j)\}$ for $j=1, \ldots, ns$. Note that (x_j, y_j) is the left (right) endpoint of segment $j\,(j-1)$. In practice, $a_j=0$ for some segment j is a very low probability event. Thus, a slope tolerance (STOL) is needed to provide a relatively horizontal segment with a "zero-like" slope attribute. For the approximated waveform string representation $w=\omega_1\omega_2\ldots\omega_j\ldots\omega_{ns}$, the ω_j are assigned as follows:

$$(\omega_j=/)\leftrightarrow(a_j>\mathrm{STOL})\,, \tag{2.33}$$

$$(\omega_j=\backslash)\leftrightarrow(a_j<-\mathrm{STOL})\,, \tag{2.34}$$

$$(\omega_j=0)\leftrightarrow(|a_j|\leqq\mathrm{STOL})\,. \tag{2.35}$$

The symbols "\", "/", and "0" have the same connotations as in (2.13–15).

At this point, the nature of the waveform with respect to baseline designation must be determined. In general, waveforms may be divided into three categories: 1) bipolar (e.g., electrocardiograms), where the signal falls on both sides of the baseline; 2) unipolar (e.g., chromatograms) or rectified bipolar waveforms, where the entire signal lies on one side of the baseline; and 3) neither unipolar nor bipolar (e.g., terrain profiles) where the notion of a baseline is irrelevant.

For waveforms in the last category, all positive and negative peaks are meaningful, and G from (2.21) recognizes these peaks in the compacted waveform as well. This grammar may be parsed in linear time by a deterministic finite state automaton [2.26, 27]

$$A=\{\Sigma, Q, \delta, z, F\} \tag{2.36}$$

where

$$\Sigma=\{/, \backslash, 0\} \tag{2.37}$$

is the set of input symbols (alphabet),

$$Q=\{z, p_1, q_1, p_2, q_2\} \tag{2.38}$$

is the set of states;

$$\begin{aligned}
&\{\delta(z,/)=p_1\,, && \delta(z,\backslash)=q_1\,, && \delta(z,0)=z\,,\\
&\delta(p_1,/)=p_1\,, && \delta(p_1,\backslash)=q_1\,, && \delta(p_1,0)=p_2\,,\\
&\delta(q_1,/)=p_1\,, && \delta(q_1,\backslash)=q_1\,, && \delta(q_1,0)=q_2\,,\\
&\delta(p_2,/)=p_1\,, && \delta(p_2,\backslash)=q_1\,, && \delta(p_2,0)=p_2\,,\\
&\delta(q_2,/)=p_1\,, && \delta(q_2,\backslash)=q_1\,, && \delta(q_2,0)=q_2\}
\end{aligned} \tag{2.39}$$

is the set of state transitions; z is the initial state; and $F=Q$ is the set of final states.

The "machine" starts with $[j \leftarrow 1; k \leftarrow 0; t \leftarrow z]$ and proceeds according to the following algorithm:

Step 1. Perform the following semantic routine for the pair (t, ω_j):

1) $\{(z,0), (p_1, /), (q_1, \backslash), (p_2, /), (q_2, \backslash), (p_2, 0), (q_2, 0)\}$:null.
2) $\{(z, /), (z, \backslash)\}$: $x^{\mathrm{l}}_{k+1} \leftarrow x_j;\ y^{\mathrm{l}}_{k+1} \leftarrow y_j$.
3) $\{(p_1, 0), (q_1, 0)\}$: $x^{\mathrm{r}}_k \leftarrow x_j;\ y^{\mathrm{r}}_k \leftarrow y_j;\ x^{\mathrm{m}}_{k+1} \leftarrow x_j;\ y^{\mathrm{m}}_{k+1} \leftarrow y_j$.
4) $\{(p_1, \backslash), (q_1, /)\}$: $x^{\mathrm{r}}_k \leftarrow x_j;\ y^{\mathrm{r}}_k \leftarrow y_j;\ k \leftarrow k+1;\ x^{\mathrm{m}}_k \leftarrow x_j;$
$y^{\mathrm{m}}_k \leftarrow y_j;\ x^{\mathrm{l}}_{k+1} \leftarrow x_j;\ y^{\mathrm{l}}_{k+1} \leftarrow y_j$.
5) $\{(p_2, \backslash), (q_2, /)\}$: $k \leftarrow k+1;\ x^{\mathrm{m}}_k \leftarrow (x^{\mathrm{m}}_k + x_j)/2;\ y^{\mathrm{m}}_k \leftarrow (y^{\mathrm{m}}_k + y_j)/2;$
$x^{\mathrm{l}}_{k+1} \leftarrow x_j;\ y^{\mathrm{l}}_{k+1} \leftarrow y_j$.

Step 2. if $(j = ns)$ then $(np \leftarrow k$; halt)

else $(j \leftarrow j+1;\ t \leftarrow \delta(t, \omega_j)$; go to Step 1).

□

When the algorithm halts, all of the positive and negative peaks in the compacted waveform have been recognized and are contained in the set $\{[(x^{\mathrm{l}}_k, y^{\mathrm{l}}_k), (x^{\mathrm{m}}_k, y^{\mathrm{m}}_k), (x^{\mathrm{r}}_k, y^{\mathrm{r}}_k)]\}$ for $k = 1, \ldots, np$. Note that if ETOL = STOL = 0 then these are all of the peaks in the original waveform satisfying (2.1–12).

For unipolar waveforms above the baseline, only positive peaks are relevant features, since negative peaks represent valleys that separate positive peaks. A reduced finite state automaton derived from (2.36–39) will meet these requirements, where

$$Q = \{z, p_1, p_2\} \tag{2.40}$$

and

$$\begin{aligned} &\{\delta(z, /) = p_1, \quad \delta(z, \backslash) = z, \quad \delta(z, 0) = z, \\ &\delta(p_1, /) = p_1, \quad \delta(p_1, \backslash) = z, \quad \delta(p_1, 0) = p_2, \\ &\delta(p_2, /) = p_1, \quad \delta(p_2, \backslash) = z, \quad \delta(p_2, 0) = p_2\}. \end{aligned} \tag{2.41}$$

The corresponding changes in Step 1 of the preceding algorithm are:

1) $\{(z,0), (p_1, /), (p_2, /), (p_2, 0)\}$: null.
2) $\{(z, \backslash)\}$: $x^{\mathrm{r}}_k \leftarrow x_{j+1};\ y^{\mathrm{r}}_k \leftarrow y_{j+1}$.
3) $\{(z, /)\}$: $x^{\mathrm{l}}_{k+1} \leftarrow x_j;\ y^{\mathrm{l}}_{k+1} \leftarrow y_j$.
4) $\{(p_1, 0)\}$: $x^{\mathrm{m}}_{k+1} \leftarrow x_j;\ y^{\mathrm{m}}_{k+1} \leftarrow y_j$.
5) $\{(p_1, \backslash)\}$: $k \leftarrow k+1;\ x^{\mathrm{m}}_k \leftarrow x_j;\ y^{\mathrm{m}}_k \leftarrow y_j;\ x^{\mathrm{r}}_k \leftarrow x_{j+1};\ y^{\mathrm{r}}_k \leftarrow y_{j+1}$.
6) $\{(p_2, \backslash)\}$: $k \leftarrow k+1;\ x^{\mathrm{m}}_k \leftarrow (x^{\mathrm{m}}_k + x_j)/2;\ y^{\mathrm{m}}_k \leftarrow (y^{\mathrm{m}}_k + y_j)/2;$
$x^{\mathrm{r}}_k \leftarrow x_{j+1};\ y^{\mathrm{r}}_k \leftarrow y_{j+1}$.

A unipolar waveform below the baseline may be inverted by changing the signs of the a_j to produce the correct results (only the negative peaks will be recognized).

Bipolar waveforms may be considered as a series of alternating unipolar segments, and each of these segments may be parsed individually in a similar fashion. A problem arises when a boundary between two unipolar segments falls in the middle of a segment produced by the preprocessor. Thus the actual segment must be split where it crosses the baseline, each part belonging to a different unipolar segment and parsed accordingly.

In practice, $y_j=0$ for some segment j is also unlikely. Thus, a baseline tolerance (BTOL) is needed to provide low amplitude endpoints with a "baseline-like" attribute. Another string $v=v_1v_2\ldots v_j\ldots v_{ns+1}$ representing the approximated waveform with respect to the baseline is formed with v_j assigned as follows:

$$(v_j=+)\leftrightarrow(y_j>\mathrm{BTOL}), \tag{2.42}$$

$$(v_j=-)\leftrightarrow(y_j<-\mathrm{BTOL}), \tag{2.43}$$

$$(v_j=*)\leftrightarrow(|y_j|\leqq\mathrm{BTOL}), \tag{2.44}$$

where "+" denotes "above the baseline", "−" denotes "below the baseline", and "*" denotes "coincident with the baseline".

Rather than construct a new grammar, a few modifications and additions made directly to the preceding algorithm will suffice. Only the string w will be parsed, while v will be inspected to perform the unipolar segmentation. Scalar variables x and y will be assigned either the point where segment j intersects the baseline (flag $f=0$) or the right endpoint of segment j (flag $f=1$). Unipolar segments below the baseline will be inverted while being parsed. The new "machine" starts with $[j\leftarrow1; k\leftarrow0; t\leftarrow z; \sigma\leftarrow\omega_j]$ and proceeds according to the following algorithm:

Step 1. Perform the following semantic routines for the pair (v_j, v_{j+1}):

1) $\{(+,+),(+,*),(*,+)\}$: $f\leftarrow1; x\leftarrow x_{j+1}; y\leftarrow y_{j+1}$.
2) $\{(-,-),(-,*),(*,-)\}$: $f\leftarrow1; x\leftarrow x_{j+1}; y\leftarrow y_{j+1}$;
   if $(\sigma=/)$ then $(\sigma\leftarrow\backslash)$
   else [if $(\sigma=\backslash)$ then $(\sigma\leftarrow/)$].
3) $\{(+,-)\}$: $f\leftarrow0; x\leftarrow-b_j/a_j; y\leftarrow0; v_j\leftarrow*$.
4) $\{(-,+)\}$: $f\leftarrow0; x\leftarrow-b_j/a_j; y\leftarrow0; v_j\leftarrow*$;
   if $(\sigma=/)$ then $(\sigma\leftarrow\backslash)$ else [if $(\sigma=\backslash)$ then $(\sigma\leftarrow/)$].
5) $\{(*,*)\}$: $f\leftarrow1; \sigma\leftarrow0; t\leftarrow z$; go to Step 3.

Step 2. Perform the following semantic routines for the current pair (t,σ):

1) $\{(z,0),(p_1,/),(p_2,/),(p_2,0)\}$: null.
2) $\{z,\backslash)\}$: $x_k^r\leftarrow x; y_k^r\leftarrow y$.

3) $\{(z, /)\}$: $x^{\mathrm{l}}_{k+1} \leftarrow x_j$; $y^{\mathrm{l}}_{k+1} \leftarrow y_j$.

4) $\{(p_1, 0)\}$: $x^{\mathrm{m}}_{k+1} \leftarrow x_j$; $y^{\mathrm{m}}_{k+1} \leftarrow y_j$.

5) $\{(p_1, \backslash)\}$: $k \leftarrow k+1$; $x^{\mathrm{m}}_k \leftarrow x_j$; $y^{\mathrm{m}}_k \leftarrow y_j$; $x^{\mathrm{r}}_k \leftarrow x$; $y^{\mathrm{r}}_k \leftarrow y$.

6) $\{(p_2, \backslash)\}$: $k \leftarrow k+1$; $x^{\mathrm{m}}_k \leftarrow (x^{\mathrm{m}}_k + x_j)/2$; $y^{\mathrm{m}}_k \leftarrow (y^{\mathrm{m}}_k + y_j)/2$;
$x^{\mathrm{r}}_k \leftarrow x$; $y^{\mathrm{r}}_k \leftarrow y$.

Step 3. if $(f=0)$ then $(x_j \leftarrow x;\ y_j \leftarrow y;\ t \leftarrow z;\ \sigma \leftarrow \omega_j$; go to Step 1)
else [if $(j = ns)$ then $(np \leftarrow j$; halt)
else $(j \leftarrow j+1;\ t \leftarrow \delta(t, \sigma);\ \sigma \leftarrow \omega_j$;
go to Step 1)].

□

2.3 Application to Electrocardiography

Electrocardiograms (recordings of the electrical activity within the heart) are model waveforms with which to test the efficacy of feature extraction techniques. In a single cardiac cycle (heartbeat), the relevant diagnostic information is contained in the components of up to five peaks forming a contiguous area (about two-thirds of the entire cycle) [2.7, 15, 42]. Although the dimensions of the peaks and cycles may vary over time and between individuals, the gross structure and relative position of the peaks (denoted by the sequence PQRST) are the same within the class of individuals with normal heartbeats measured by a standard system of electrode leads (attached to the chest and arms and grounded to the right leg, in this case).

Unfortunately, the environment of the patient and measuring apparatus introduces noise from a number of origins into the signal [2.7, 15, 29, 30]. Involuntary muscle activity and sixty Hertz alternating current interference are exaggerated by extreme amplification generating high frequency peaks interspersed throughout the waveform. Low frequency noise, manifested as vertical displacement of portions of the waveform, is due to body movement and shifting of the lead electrodes across equipotential lines of the skin. Respiration and changes in cutaneous resistance contribute to both kinds of distortion and modulate the entire signal as well.

Successful electrocardiogram recognition schemes must distinguish the significant peaks from the spurious ones. Since the waveform is periodic, a simple averaging of several cycles appears to be an effective signal enhancer, however successive cycles may have different lengths. Furthermore, a feducial point must be established for each cycle, requiring preliminary pattern discrimination which may be complicated by the noise.

An unenhanced electrocardiogram was chosen as an appropriate example to demonstrate the peak recognition methods presented in this chapter. Electrocardiograms are bipolar waveforms asymmetrically distributed about an isoelectric baseline (usually taken to be that part of the signal connecting adjacent PQRST complexes, during which the heart is quiescent).

Figure 2.1 displays about three-quarters (275 points) of a single cardiac cycle in lead number one (written "I") of a digitized (at 400 7-bit samples per second),

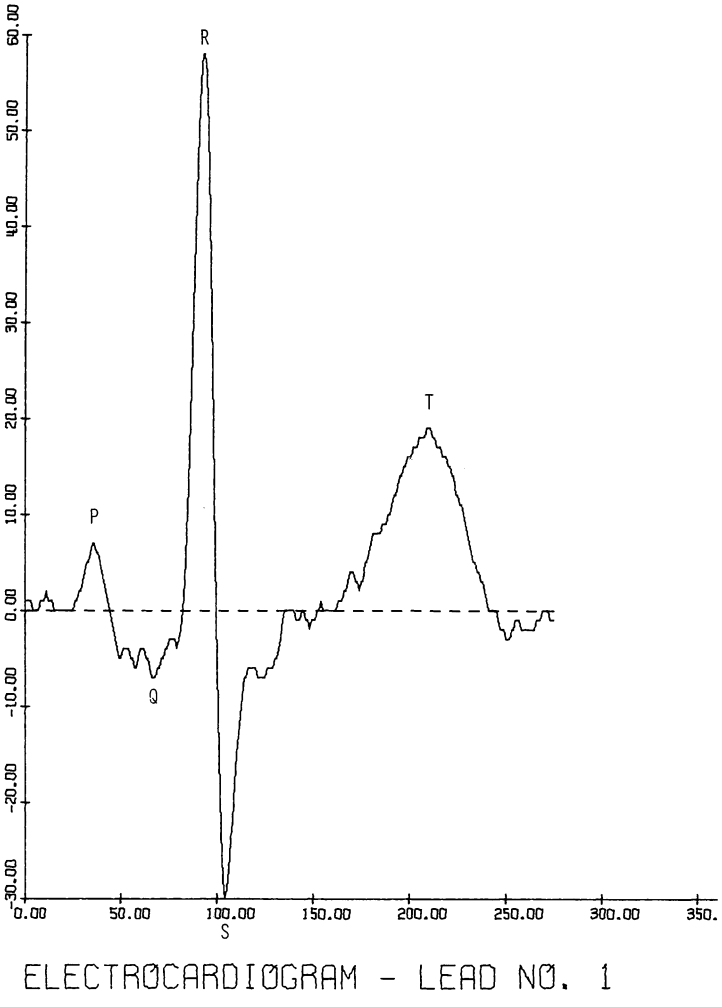


Fig. 2.1. The active portion of a single cardiac cycle in lead number one (I) of a digitized, normal, twelve-lead electrocardiogram (enlarged). Dotted line marks the baseline. Significant peaks are labeled (by hand)

standard twelve lead electrocardiogram from a patient with a normal heartbeat. The isoelectric baseline has been estimated and subtracted from the waveform resulting in a new baseline (marked by a dotted line) with zero slope and constant. Each peak comprising the PQRST complex is labeled (by hand).

The experiment was executed by an IBM 360/91 computer operating under OS MVT/LASP using an optimized FORTRAN H program based on the last algorithm in Subect. 2.2.3. The preprocessor ran in approximately fifty ms[1] and reduced the 275 points to thirteen segments of four values apiece, plus two values for (x_{ns+1}, y_{ns+1}), yielding a 550:54 (roughly ten to one) compaction

[1] CPU time (excluding I/O) accurate to within an observed thirty percent.

```
ETOL = 360.0

 J      X(J)       Y(J)        A(J)          B(J)

 1       1.0        1.0     -3.769E-02     9.300E-01
 2      25.0        0.0      6.678E-01    -1.679E 01
 3      36.0        7.0     -9.473E-01     4.162E 01
 4      49.0       -5.0      3.281E-02    -6.835E 00
 5      80.0       -3.0      2.700E 00    -2.202E 02
 6      84.0        8.0      7.036E 00    -5.842E 02
 7      91.0       55.0      3.000E-01     2.900E 01
 8      94.0       56.0     -1.013E 01     1.011E 03
 9     103.0      -28.0      2.262E 00    -2.649E 02
10     114.0       -7.0      1.924E-01    -2.909E 01
11     159.0        0.0      4.063E-01    -6.611E 01
12     211.0       19.0     -6.216E-01     1.511E 02
13     247.0       -2.0      7.291E-02    -2.055E 01
14     275.0       -1.0
```

(a)

```
BTOL =   2.0        V = **+--+++--****

STOL =   0.2        W = 0/\0///\/0/\0

 K     XL(K)      YL(K)      XM(K)      YM(K)      XR(K)      YR(K)

 1      25.0        0.0       36.0        7.0       43.9        0.0
 2      43.9        0.0       64.5       -4.0       81.6        0.0
 3      81.6        0.0       94.0       56.0       99.8        0.0
 4      99.8        0.0      103.0      -28.0      114.0       -7.0
 5     159.0        0.0      211.0       19.0      247.0       -2.0

                          PEAK ANALYSIS

POS. PEAK AT  36.0   AMP. =   7.0   DUR. =  11.0 +  7.9 = 18.9
NEG. PEAK AT  64.5   AMP. =  -4.0   DUR. =  20.6 + 17.1 = 37.6
POS. PEAK AT  94.0   AMP. =  56.0   DUR. =  12.4 +  5.8 = 18.2
NEG. PEAK AT 103.0   AMP. = -24.5   DUR. =   3.2 + 11.0 = 14.2
POS. PEAK AT 211.0   AMP. =  20.0   DUR. =  52.0 + 36.0 = 88.0
```

(b)

Fig. 2.2a and b. IBM 360/91 FORTRAN H computer program (bipolar algorithm) output: a) from the preprocessor: column (1) segment number j; (2) abscissa of left segment endpoint x_j; (3) ordinate of left segment endpoint y_j; (4) segment slope a_j; (5) segment constant b_j. (b) from the peak detector: column (1) peak number k; (2) abscissa of left peak boundary x_k^{l}; (3) ordinate of left peak boundary y_k^{l}; (4) abscissa of peak extremum x_k^{m}; (5) ordinate of peak extremum y_k^{m}; (6) abscissa of right peak boundary x_k^{r}; (7) ordinate of right peak boundary y_k^{r}

ratio (Fig. 2.2a). The accompanying piecewise linear approximation is displayed in Fig. 2.3. The recognizer ran in approximately seven ms and identified five peaks (Fig. 2.2b) which are displayed in Fig. 2.4 (squares and crosses denote peak boundaries and extrema, respectively) and match the labeled PQRST components in Fig. 2.1.

The "peak analysis" offers additional useful information about peak direction, absolute location, relative amplitude, and duration (split into rise-time/fall-time pairs). These are calculated from the values of $[(x_k^{\mathrm{l}}, y_k^{\mathrm{l}}), (x_k^{\mathrm{m}}, y_k^{\mathrm{m}}), (x_k^{\mathrm{r}}, y_k^{\mathrm{r}})]$ as follows: A peak

1) is either positive if $y_k^{\mathrm{l}} > y_k^{\mathrm{m}}$ (and $y_k^{\mathrm{r}} > y_k^{\mathrm{m}}$) or negative if $y_k^{\mathrm{l}} < y_k^{\mathrm{m}}$ (and $y_k^{\mathrm{r}} < y_k^{\mathrm{m}}$);
2) is located at x_k^{m};
3) has amplitude $y_k^{\mathrm{m}} - (y_k^{\mathrm{l}} + y_k^{\mathrm{r}})/2$; and
4) has duration $(x_k^{\mathrm{m}} - x_k^{\mathrm{l}}) + (x_k^{\mathrm{r}} - x_k^{\mathrm{m}}) = (x_k^{\mathrm{r}} - x_k^{\mathrm{l}})$.

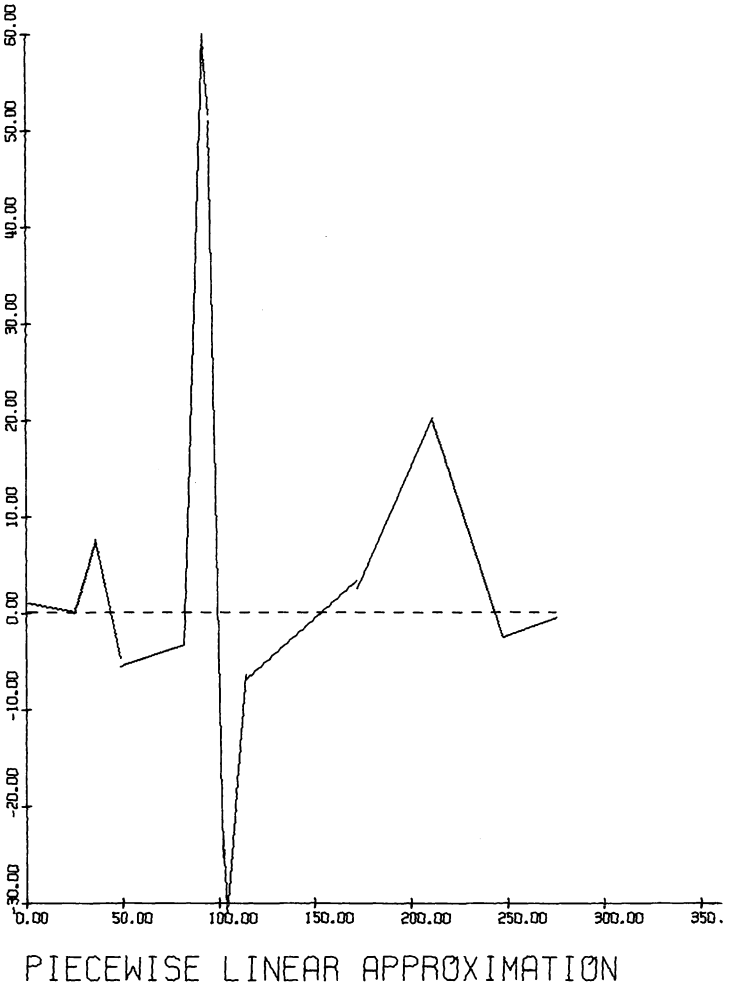


Fig. 2.3. The piecewise linear approximation of Fig. 2.1. Corresponding values are given in Fig. 2.2a

Thus peaks (as approximated by fixed triangles) may be matched with templates, compared to previously or subsequently generated peaks, and examined for important anomalies in orientation, size (height, width, and area), symmetry, and position (e.g. to facilitate arrhythmia diagnosis [2.16, 43, 44]).

Values of ETOL, STOL, and BTOL were derived by inspection of the scale and noise content of the electrocardiogram. In practice, these parameters are quite flexible, allowing for a fairly wide margin of error in their assignment. Once established for a given waveform and environment, ETOL must be linearly proportioned to the number of data points per input record, while STOL and BTOL remain unchanged. Three adjacent heartbeats consisting of 1100 points from the same data base produced thirteen peaks (Q was absent from the last two complexes—an inherent deficiency in lead I) using an ETOL

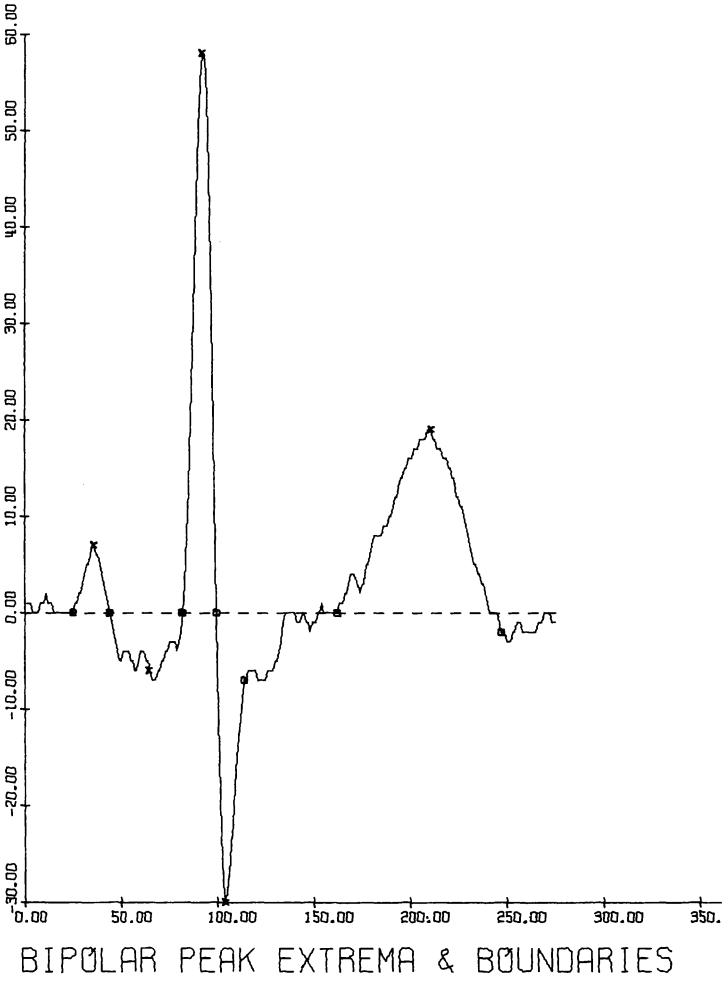


Fig. 2.4. Peaks recognized in Fig. 2.1 by bipolar algorithm (squares denote peak boundaries, crosses denote peak extrema). Corresponding values are given in Fig. 2.2b

multiplied by four and the same STOL and BTOL. The processing time was approximately one quarter second, which is eleven times less than the threshold necessary for real-time feasibility (2.75 s) stipulated by the digitization rate. Automatic calculations of the horizontal distances between consecutive peak extrema whose amplitudes and durations are within prespecified tolerances (five units apiece in this case) yielded a naive yet reliable average instantaneous pulse period of 375 points (equivalent to a pulse rate of 64 beats per minute).

Although the examples in this section were conducted off-line, a continuous, real-time, on-line system can be achieved. Three parallel processes utilizing three circular buffers (operating in "producer/consumer" configurations) comprise the basic design. The first processor reads data points $\{(x_i, y_i)\}$ from the A/D

converter into the first buffer. When n points have been collected, the second processor approximates the stored waveform and fills part of the second buffer with $ns+1$ segment points interleaved with ns slope symbols $\{(x_j, y_j), \omega_j\}$, (x_{ns+1}, y_{ns+1}). The final processor parses the string w and writes the $np \times 3$ peak points $\{(x_k^{\mathrm{l}}, y_k^{\mathrm{l}}), (x_k^{\mathrm{m}}, y_k^{\mathrm{m}}), (x_k^{\mathrm{r}}, y_k^{\mathrm{r}})\}$ into the third buffer, which may be accessed by some external output device.

The interface between consecutive records of data points is handled as follows: the preceding (x_n, y_n) becomes the next (x_1, y_1) and tentative values of $(x_{np+1}^{\mathrm{l}}, y_{np+1}^{\mathrm{l}})$, $(x_{np+1}^{\mathrm{m}}, y_{np+1}^{\mathrm{m}})$ and the final state become $(x_1^{\mathrm{l}}, y_1^{\mathrm{l}})$, $(x_1^{\mathrm{m}}, y_1^{\mathrm{m}})$ and the start state, respectively. If an extra first segment is generated because of the break, the parser will consider it a part of the last segment from the previous block.

To minimize processor idle time, the three buffers should be integer multiples in length of the factors n, ns, and np, respectively. Conservative estimates for these factors are $n=1000$, $ns=200$, and $np=50$, times the appropriate bit size per buffer entry, determined by the digitization sample size and the number of values per buffer entry. The integer multiples depend on memory limitations and the relative speeds of the software and hardware implementations of the system.

2.4 Concluding Remarks

Syntactic pattern recognition has become a promising tool for the extraction of features from digitized waveforms. PAVLIDIS [2.1] provided the seminal concepts in 1971; since then, a number of specific waveform parsing schemes have been published: DEMORI [2.6] (speech), LEE [2.10] (electroencephalograms), STOCKMAN et al. [2.13] (carotid pulse waves), and ALBUS [2.22] (electrocardiograms). All of their grammars contain nonterminals that represent higher order features relevant to the designated class of waveforms.

HOROWITZ [2.25] proposed a lower level approach applicable to most waveforms when an a priori model is lacking or as a preliminary step before high order parsing (i.e., the primitives would be line segments and peaks rather than single points). In this chapter, a considerable degree of formalization has resulted in three algorithms for peak recognition in general classes of waveforms that are more efficient and rigorous than the previous work.

These new algorithms incorporate proven techniques from approximation theory (discrete piecewise linear function representation) and formal language theory (bottom-up, automaton-driven, syntax-directed parsing). Context is specified by choice of the appropriate algorithm (baseline-free, unipolar, and bipolar) and the assignment of two or three flexible parameters (according to scale and noise content). Other waveform characteristics (e.g., discontinuities and inflections) may be detected through minor changes within the structure of the routines (e.g., approximating function, error norm, grammar productions, parser type, and semantic routines).

As they now stand, the algorithms are fast enough to allow real-time processing of digitized graphic and sampled analog data, as demonstrated

by recognizing the significant peaks in an electrocardiogram. An on-line system adaptable to microprocessor technology has been outlined, suggesting front-end biomedical and industrial applications in signal monitoring, feedback control, and waveform analysis.

Acknowledgments. The author would like to thank Professor Theodosios Pavlidis of the Department of Electrical Engineering at Princeton University for his initial suggestion to pursue the topic of peak detection in waveforms, and for his valuable assistance at various stages of this work.

Special thanks are also due to Dr. John C. Seed of the Montefiore Hospital of New York City for the digitized electrocardiograms.

This research was supported in part by National Science Foundation Grants GK 36180 and ENG 72-04133 while the author was with the Department of Electrical Engineering at Princeton University.

References

2.1 T. PAVLIDIS: in *Software Engineering*, Vol. 2, ed. by J. T. TOU (Academic Press, New York, London 1971), pp. 203–225
2.2 R. W. ERICH, J. P. FOITH: IEEE Trans. C-25, 725 (1976)
2.3 B. WIDROW: Pat. Rec. **5**, 175 (1973)
2.4 A. STEFANSKI: Hewlett-Packard J. **26**, 18 (1974)
2.5 M. BAUDRY, B. DUPEYRAT: In *Proceedings of the Second Joint International Conference on Pattern Recognition*, Copenhagen (1974), pp. 356–357
2.6 R. DEMORI: Syntactic Recognition of Speech Patterns; this publication, Chapter 4
2.7 J. R. COX, F. M. NOLLE, R. M. ARTHUR: Proc. IEEE **60**, 1137 (1972)
2.8 M. SERAFINI: Comp. in Biomed. Res. **6**, 187 (1973)
2.9 J. R. SMITH: IEEE Trans. BME-**21**, 1 (1974)
2.10 S. C. LEE: In *Proceedings of the Conference on Computer Graphics, Pattern Recognition and Data Structures*, Los Angeles (1975), pp. 135–138
2.11 F. K. KLEIN: IEEE Trans. BME-23, 246 (1976)
2.12 R. H. SWATZELL, W. H. BANCROFT, JR., J. MACY, JR., E. F. EDDELMAN, JR.: Comp. in Biomed. Res. **6**, 465 (1973)
2.13 G. STOCKMAN, L. KANAL, M. C. KYLE: In *Proceedings of the First Joint International Conference on Pattern Recognition*, Washington D.C. (1973), pp. 236–243
2.14 R. E. BONNER, D. SCHWETMAN: Comp. in Biomed. Res. **1**, 366 (1968)
2.15 T. C. HORTH: Hewlett-Packard J. **21**, 12 (1969)
2.16 E. KIMURA, K. OBAYASHI, T. AKAZOME, H. HAYAKAWA, G. KASAI, Z. E. ABE, T. SUZUKI, K. KATO: J. Electrocardiology **6**, 137 (1973)
2.17 T. Y. YOUNG, T. W. CALVERT: *Classification, Estimation, and Pattern Recognition* (American Elsevier, New York, London, Amsterdam 1974), pp. 277–307
2.18 M. L. SIMOONS, H. B. K. BOOM, F. SMALLENBURG: Comp. in Biomed. Res. **8**, 105 (1975)
2.19 M. G. GOOVAERTS, H. H. ROS, T. J. VAN DEN AKKER, H. SCHNEIDER: IEEE Trans. BME-23, 154 (1976)
2.20 S. J. HENGEVELD, J. H. VAN BEMMEL: Comp. in Biomed. Res. **9**, 125 (1976)
2.21 M. GRAHAM: IEEE Trans. BME-23, 220 (1976)
2.22 J. E. ALBUS: Electrocardiogram Interpretation Using a Stochastic Finite-State Model; this publication, Chapter 3
2.23 I. TOMEK: Comp. in Biomed. Res. **6**, 499 (1973)
2.24 R. O. DUDA, P. E. HART: *Pattern Classification and Scene Analysis* (John Wiley, New York, London, Sydney, Toronto 1973), pp. 298–341
2.25 S. L. HOROWITZ: CACM **18**, 281 (1975)
2.26 A. V. AHO, J. D. ULLMAN: *The Theory of Parsing, Translation, and Compiling*, Vol. 1 (Prentice Hall, Englewood Cliffs, N.J. 1972), pp. 83–138

2.27 K.S. FU: *Syntactic Methods in Pattern Recognition* (Academic Press, New York, London 1974), pp. 25–34
2.28 J.R. COX, H.A. FOZZARD, F.M. NOLLE, G.C. OLIVER: in *Computers in Biomedical Research*, Vol. 3, ed. by R.W. STACY, B.D. WAXMAN (Academic Press, New York, London 1969), pp. 181–206
2.29 C.S. WEAVER, J. VON DER GROEBEN, P.E. MANTEY, J.G. TOOLE, C.A. COLE, JR., J.W. FITZGERALD, R.W. LAWRENCE: IEEE Trans. AU-**16**, 350 (1968)
2.30 T.A. PRYOR: Comp. in Biomed. Res. **4**, 542 (1971)
2.31 W. GERSCH: Comp. in Biomed. Res. **6**, 281 (1973)
2.32 K. STEIGLITZ: *An Introduction to Discrete Systems* (John Wiley, New York, London, Sydney, Toronto 1974), pp. 24–138
2.33 H.C. ANDREWS: *Introduction to Mathematical Techniques in Pattern Recognition* (John Wiley, New York, London, Sydney, Toronto 1972)
2.34 N. AHMED, P.J. MILNE, S.G. HARRIS: IEEE Trans. BME-**22**, 484 (1975)
2.35 J.R. RICE: *The Approximation of Functions*, Vol. 2 (Addison Wesley, Reading, Mass., Menlo Park, Calif., London 1969), pp. 123–167
2.36 A. TALBOT, ed.: *Approximation Theory* (Academic Press, New York, London 1970), pp. 191–232
2.37 T. PAVLIDIS: IEEE Trans. C-**22**, 689 (1973)
2.38 T. PAVLIDIS, S.L. HOROWITZ: IEEE Trans. C-**23**, 860 (1974)
2.39 J.R. RICE: *The Approximation of Functions*, Vol. 1 (Addison Wesley, Reading, Mass., Menlo Park, Calif., London 1964), pp. 1–59
2.40 I. GUTMAN, S.S. WILKS: *Introductory Engineering Statistics*, 2nd ed. (John Wiley, New York, London, Sydney, Toronto 1971), pp. 339–346
2.41 R.R. GIBSON, M. GRAHAM, S.L. HOROWITZ: "Electrocardiogram Baseline Extraction Techniques", Department of Electrical Engineering and Computer Sciences, University of California at Berkeley (in progress).
2.42 R. PLONSEY: *Bioelectric Phenomena* (McGraw Hill, New York, Saint Louis, San Francisco, Toronto, London, Sydney 1969), pp. 299–304
2.43 J.L. WILLEMS, H.V. PIPEBERGER: Comp. in Biomed. Res. **5**, 263 (1972)
2.44 J.R. WHITEMAN, F.A. SIEGEL, J.B. BREINIG: In *Computers in Biomedical Research*, Vol. 4, ed. by R.W. STACY, B.D. WAXMAN (Academic Press, New York, London 1974), pp. 90–114

3. Electrocardiogram Interpretation Using a Stochastic Finite State Model

J. E. ALBUS

With 8 Figures

A syntactic approach to the problem of electrocardiogram rhythm interpretation is presented and illustrated with a simple application.

The interpretation problem is defined using a communication system model. A stochastic finite state automaton, called the rhythm model, is made for the time intervals between heart muscle stimulations. The output of the rhythm model is encoded into the continuous ECG signals by the heart muscles. The ECG is then decoded into a discrete sequence of symbols called the detector output. The transformation of the rhythm model outputs into the detector outputs is modeled as being a noisy channel.

The problem of interpreting a detector output sequence is to find the rhythm model state and output sequences which have the highest probability of having caused the observed sequence. These sequences are called the interpretation of the detector outputs. Given the initial state of the rhythm model and a sequence of detector outputs a tree search procedure may be used to solve this problem.

The purpose of this chapter is to present a syntactic approach to the problem of electrocardiogram (ECG) rhythm interpretation and to illustrate it with a simple application. The general use of syntactic methods for pattern recognition has been described by FU [3.1].

The interpretation problem is defined using a communication system model. A stochastic finite-state automaton (SFSA), called the rhythm model, is made for the time intervals between heart muscle stimulations. The output of the rhythm model is encoded into the continuous ECG signals by the heart muscles. The ECG is then decoded into a discrete sequence of symbols called the detector output. The transformation of the rhythm model outputs into the detector outputs is modeled as being a noisy channel.

The problem of interpreting a detector output sequence is to find, given the detector output, the rhythm model state and output sequences which have the highest probability. These sequences are called the interpretation of the detector output. Given the initial state of the rhythm model and a sequence of detector outputs a tree search procedure may be used to solve this problem.

The heart has two muscles; the atria and the ventricles. The atrial muscle is the smaller of the two. Its chambers act as filling pumps for the ventricular chambers which do most of the pumping work. In the normal heart cycle the atrial muscle is stimulated first. There is then a delay before the stimulation of the ventricles which allows the atria to complete mechanical

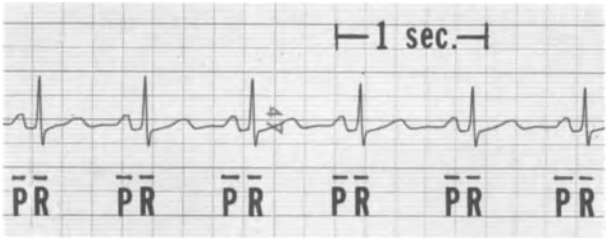


Fig. 3.1. Sample ECG with normal rhythm

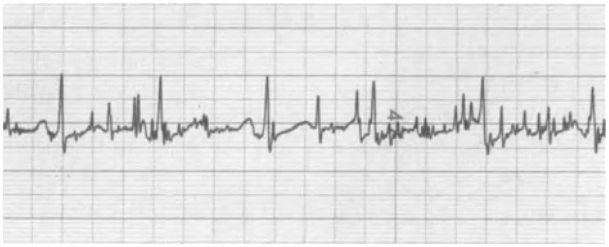


Fig. 3.2. Noisy ECG

contraction. The rhythm system of the heart is responsible for stimulating the muscles in this order.

When a heart muscle is activated by a stimulation, it generates electric fields which may be measured on the surface of the body by the electrocardiogram. A sample ECG showing a normal rhythm is shown in Fig. 3.1. A pulse shape due to an atrial activation is called a *P* wave, and the shapes generated by ventricular activations are called *R* waves. The locations of the *P* and *R* waves are illustrated on the figure. (The slower bump following each *R* wave is called a *T* wave and is generated by the repolarization of the ventricles following an activation).

In certain critically ill patients there is a high risk of rhythm system malfunction. If certain malfunctions are undetected death occurs quickly. The ECG's of these patients should be monitored continuously.

The main problem in monitoring the ECG is to decide what sequence of events occurred in the rhythm system to produce the observed ECG. This sequence is called an interpretation of the signals. Once the interpretation is found it is not difficult to decide whether the rhythm system is functioning properly.

A complication to the interpretation problem is the fact that potentials generated by muscles other than the heart appear on the ECG. Thus patient movement may produce signals such as shown in Fig. 3.2. A monitoring system should be tolerant to detection errors caused by noise.

The interpretation problem is defined using a stochastic model which has the form of a communication system from information science. The system is illustrated in Fig. 3.3.

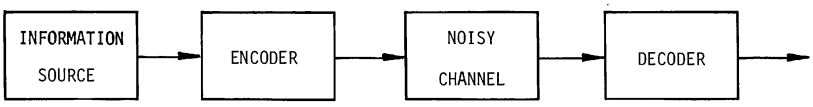


Fig. 3.3. Communication system model for the observation of the rhythm system

The information source in this case is the rhythm system, which is modeled by an SFSA. It operates in discrete time, and its outputs represent sources of stimulation to the heart muscles.

The heart muscles are represented as being an encoding device. When a muscle receives a stimulation from the rhythm model it generates a fixed pulse shape on the ECG signal during the interval following the stimulation.

This idealized ECG generated by the heart muscles is transmitted to the surface of the body, where the ECG electrodes are placed. Here the signal is combined with noise from several sources. This contamination of the signal is represented by the noisy channel.

Finally the ECG signal is decoded into a discrete sequence of symbols by a detector. The detector is used to locate the activation pulse shapes on the output of the channel. It is assumed that each pulse is detected with probability one, but false detections occur with nonzero probability. The transformation of the rhythm model output into the detector output is modeled as being a memoryless noisy channel.

The total system, composed of the rhythm model and the transformation of its output into the output of the detectors, has the form of an SFSA. The state of the total system is the state and output of the rhythm system. The output of the total system is the detector output.

The interpretation problem is defined in terms of this total system. Let the rhythm model be in state w_0 at time 0. Then during the time interval from 1 to N let $\underline{w}(\underline{a})$ be the sequence of rhythm model states (outputs), and $\underline{d}$ be the detector output sequence. Then given $\underline{d}$ and w_0 the interpretation problem is to find $\underline{w}$, and $\underline{a}$ such that $P(\underline{a}, \underline{w}|w_0, \underline{d})$ is maximum.

A tree search procedure is used to solve this problem. For a given detector output sequence a tree is defined. The root of the tree represents the initial state of the total system and each path through the tree represents a sequence of total system states which could have occurred to produce the observed detector output. A uniform cost tree search algorithm is used to find the state sequence with highest probability.

A simplified version of the ECG interpretation problem is used to illustrate the approach. Only the R wave of the ECG is taken into account. The procedure could be used in a monitoring situation to test for the regular occurrence of the normal R wave shape.

In this example the rhythm system consists of a single pacemaker and its output is connected to a single muscle. The detector then locates each activation of this muscle on the ECG, along with false detections. The interpretation problem then reduces to deciding which detections represent actual activations and which are false.

3.1 The Interpretation Problem

This section defines interpretation problem for the output of stochastic finite-state automata (SFSA). This type of model is described in Fu [3.1]. Note that in modeling physical systems the SFSA is being used as a generator of sequences rather than an acceptor; thus, the final state set is omitted from the definition.

SFSA are defined as follows: It is assumed that time is discrete, taking values from the integers 0, 1, 2, ... An SFSA M is the quadruple $M=(\Sigma, \Omega, P, w_0)$, where:

Σ is a finite set of symbols called the output alphabet. At each time exactly one symbol from this set is observed at the output of M.

Ω is a finite set of states called the state space. At each time M is in some state from this set.

w_0 is the initial state, the state of M at time zero.

P, the transition probability function, is a discrete conditional probability function such that $P(a, w'|w) \geqq 0$ for all w and w' in Ω, and for all $a \in \Sigma$. The function P must satisfy $\sum_{a \in \Sigma} \sum_{w' \in \Omega} P(a, w'|w) = 1$.

The operation of the automaton is as follows: If at time t the automaton is in state w, then at time $t+1$ it will produce an output, a, and enter state w' such that $P(a, w'|w) > 0$.

Sequences of states and outputs are denoted by underscored symbols. Let $\underline{a} = a_1 a_2 \ldots a_N$ be a sequence of output symbols. Let $\underline{w} = w_1 w_2 \ldots w_N$ be a sequence of states. The probability $P(\underline{a}, \underline{w}|w_0)$ may be computed by $P(\underline{a}, \underline{w}|w_0) = \prod_{i+1}^{N} P(a_i, w_i|w_{i-1})$.

Suppose the output of the machine during the time interval $t = 1, 2, \ldots, N$ is observed. Let $\underline{a}$ be the observed sequence of outputs. Let $\underline{w}$ represent the sequence of internal states of the machine during this interval. The problem of interpreting the output sequence $\underline{a}$ is to find the sequence of internal states $\underline{w}$ which produced the output $\underline{a}$.

There may not exist a sequence of states, $\underline{w}$, such that $P(\underline{a}, \underline{w}|w_0)$ is greater than zero. In this case the output sequence is rejected. On the other hand there may be more than one sequence of states for which this probability is greater than zero. In this case a minimum error rate decision rule may be used to choose the sequence of internal states $\underline{w}$ which maximizes $P(\underline{w}|\underline{a}, w_0)$. Since

$$P(\underline{w}|\underline{a}, w_0) = \frac{P(\underline{a}, \underline{w}|w_0)}{P(\underline{a}|w_0)}$$

and $P(\underline{a}|w_0)$ is the same for all choices of $\underline{w}$, this is equivalent to maximizing $P(\underline{a}, \underline{w}|w_0)$.

3.2 A Solution to the Interpretation Problem

For any given output sequence, $\underline{a} = a_1 a_2 \ldots a_N$, a tree is defined such that the path from the root to each node represents a sequence of states. With each node is associated the probability of the state sequence it represents. This tree may then be searched to find the sequence of states $\underline{w}$ for which $P(\underline{a}, \underline{w}|w_0)$ is maximum. If no such state sequence exists the output sequence $\underline{a}$ is rejected.

3.2.1 The Tree Structure

The set of nodes in the tree is designated V. The root of the tree is an element of V designated v_0. Each node in V is labeled with a state from Ω. The root is labeled with the initial state, w_0.

The nodes are arranged into levels. A node on level i, for $i=0, 1, \ldots, N$, labeled w corresponds to the possibility that at time i the automaton was in state w. Let V_i be the set of nodes on level i of the tree. V_0 consists of the root node only.

For a given output symbol a, define the set of output successors of state w by $S(w, a) = \{w' : P(a, w'|w) > 0\}$. The successor nodes of a node $v \in V_i$, labeled w, are in V_{i+1} and represent the output a_{i+1} successors of w, for $i=0, 1, \ldots, N-1$. Thus, for each $w' \in S(w, a_{i+1})$ there is a node v' in V_{i+1}, labeled w'. A directed arc is drawn from node v to node v'. This arc represents the transition from state w at time i to state w' at time $i+1$ with output a_{i+1}. Node v is called the father of node v', and v' is called the son of v. Note that there are no successors of the nodes in V_N, the highest level in the tree.

There may be two nodes in V_i, v and v', with labels w and w', respectively, for which some state w'' is in both $S(w, a_{i+1})$ and $S(w', a_{i+1})$. Then, in order to preserve the tree structure, two nodes labeled w'' are constructed on level $i+1$, one representing the son of v, and the other representing the son of v'. It will turn out later that only one of these nodes need have its successors constructed.

3.2.2 The Yield of a Node

There is a unique sequence of arcs and nodes, called a path, which connects the root to each node in the tree. The sequence of state labels on the nodes on a path from the root to a node is called the yield of a node. The yield of a node may be defined recursively. The yield of the root is $Y(v_0) = e$, the empty sequence. If node v', labeled w', is not the root and node v is the father of v', then $Y(v') = Y(v)w'$.

3.2.3 The Probability of a Node

With each node in the tree, v, is associated a probability value, $\varrho(v)$. For the root, $\varrho(v_0) = 1$. Let the first i symbols in the sequence $\underline{a}$ be denoted $\underline{a}_i = a_1 a_2 \ldots a_i$. If $v \in V_i$ with yield $Y(v) = \underline{w}$, then $\varrho(v) = P(\underline{a}_i, \underline{w}|w_0)$. $\varrho(v)$ may be computed recursively as follows: If $v' \in V_i$, for $i = 1, 2, \ldots, N$, with label w', and if node v, labeled w, is the father of v', then $\varrho(v') = \varrho(v) P(a_i, w'|w)$.

3.2.4 The Tree Search

An ordered search algorithm, such as the uniform cost algorithm described in NILSSON [3.2], may be used to find a node v^0 in v_N such that $\varrho(v^0) = \max_{v \in V_N} (\varrho(v))$. The yield of this node is then the state sequence which is being

sought. Note that, except for practical reasons, an ordered search is not necessary. Since the tree is finite, v^0 may be found by simply constructing the tree.

Now there may occur two nodes, v and v', on level i of the tree which have the same state label. In this case it is not necessary to construct the successors of the node with the lower ϱ value. This is due to the following. The two subtrees formed by each node and its descendants are identical since the successors of a node depend only on the state label of the node and its level in the tree. Thus, if there is a path from node v to a node u on level N, with a sequence of labels $\underline{w}$, then there is also a path from node v' to a node u' on level N with the same sequence of labels. Let $\varrho(u)=\varrho(v)q$, where q is the probability of the portion of the path from v to u. Then $\varrho(u')=\varrho(v')q$ since the path from v' to u' represents the same sequence of states and outputs. Thus $\varrho(u)\geqq\varrho(u')$ if and only if $\varrho(v)\geqq\varrho(v')$.

3.3 A Simple Example

As an example of a stochastic finite state automaton, take the following case: $\Omega=\{A, B, C\}$, $\Sigma=\{0, 1\}$, and $w_0=A$. The transition probability function is defined by the state transition diagram in Fig. 3.4. The states of the automaton are indicated by the labeled circles. An arc from state w to state w' labeled (a/q) indicates $P(a, w'|w)=q$.

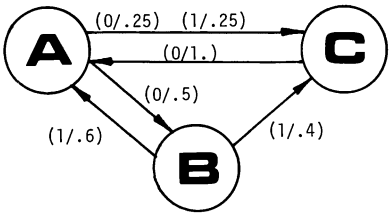


Fig. 3.4. A three state SFSA

Suppose the output sequence $\underline{a}=01011$ is observed from this automaton. The search tree for this output sequence is shown in Fig. 3.5. The squares on this figure represent the nodes of the tree. The state symbol in a node represents its state label. The number in a node represents its associated probability. The arcs are labeled with the output and probability of the transition from father to son.

The root of the tree is labeled with the initial state, A, and has a probability value of 1. Given the first output was zero, the possible states on level 1 are B and C, with probabilities as shown. Since the next output is 0, there are no successors of the node labeled C on level 1, and so on.

On level 4 of the output of 1 caused the generation of two state nodes having state label C. Since the successor of state B has a higher probability than the successor of state A, the latter node is deleted from the search. Finally, after the last output is considered, there is only one state node on level 5. Then the only possible sequence of states is ABABAC.

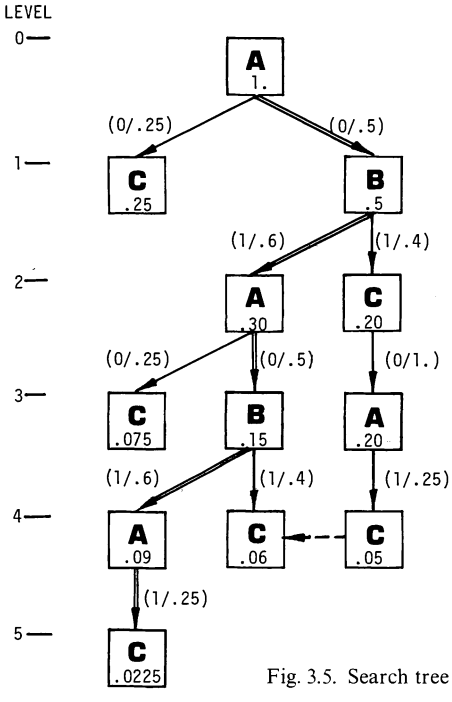


Fig. 3.5. Search tree for the SFSA of Fig. 3.4 and output sequence 01011

3.4 Application to Analog Signal Processing

The problem of analyzing the analog output of a physical system is now considered. Detectors are commonly used in the analysis of analog signals to covert the continuous data into a sequence of discrete symbols. Here, the interpretation procedure is not applied directly to the analog signal. It is assumed that a detector with certain properties exists. The procedure is applied to the discrete output of this detector. Figure 3.6 illustrates a model for the detector output sequence.

It is assumed that the physical system in question may be modeled by the combination of a basic model and a digital to analog encoding. The basic model is an SFSA, M, which passes through a sequence of states $\underline{w}$, and produces a discrete output sequence $\underline{a}$. This output sequence is transformed by some process into an analog signal. It is assumed that a detector which approximately performs the inverse of this transformation is available. Its output is a discrete sequence of symbols $\underline{d}$ from some finite alphabet. It is assumed that the transformation of the basic model output sequence $\underline{a}$ into the detector output sequence $\underline{d}$ may be modeled as a noisy channel, i.e., the conditional

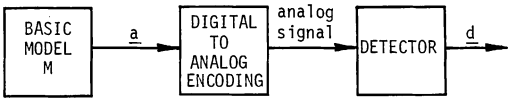


Fig. 3.6. Use of detector for analog signal interpretation

probability $P(\underline{d}|\underline{a})$ is given by $P(\underline{d}|\underline{a}) = \prod_{i=1}^{N} P(d_i|a_i)$, where $P(d_i|a_i)$ is independent of i and is known.

Now the entire system including the detector represents an SFSA whose output is to be interpreted. Since the model output $\underline{a}$ is not directly observed it becomes part of the unknown state of the system. Thus, given the detector output $\underline{d}$ and the initial state of the model w_0, the pair of sequence of model outputs and model states $(\underline{a}, \underline{w})$ must be found for which the probability $P(\underline{a}, \underline{w}|\underline{d}, w_0)$ is maximum. This probability may be rewritten

$$P(\underline{a}, \underline{w}|\underline{d}, w_0) = \frac{P(\underline{a}, \underline{w}, \underline{d}, w_0)}{P(\underline{d}, w_0)} = \frac{P(\underline{d}|\underline{a}, \underline{w}, w_0)\, P(\underline{a}, \underline{w}, w_0)}{P(\underline{d}, w_0)}$$
$$= \frac{P(\underline{d}|\underline{a})\, P(\underline{a}, \underline{w}|w_0)}{P(\underline{d}|w_0)}.$$

In the last step it was assumed that the detector output is independent of the state of the model given the model output. Since $P(\underline{d}|w_0)$ is the same for all pairs $(\underline{a}, \underline{w})$ it is necessary only to maximize the quantity $P(\underline{a}, \underline{w}|w_0)P(\underline{d}|\underline{a})$. This may be computed from

$$P(\underline{a}, \underline{w}|w_0)\, P(\underline{d}|\underline{a}) = \prod_{i=1}^{N} P(a_i, w_i|w_{i-1})\, P(d_i|a_i)\,.$$

As before, a tree search procedure may be used to find the interpretation $(\underline{a}, \underline{w})$.

3.5 An Example Application

This section presents an example of the application of these ideas to an analog signal interpretation problem. A clock having a random period emits pulses on an analog signal. A detector for these pulses is available; however, the detector occasionally produces false detections. A tree search method for deciding which detections represent true clock pulses and which are false detections is described.

3.5.1 The Clock Model

The basic model in this example is a clock having a random period, T. It is assumed that the density of the random period $P(T=t)$ is known. The output of this clock is a binary string; a one indicates a clock pulse, a zero no pulse. (This clock represents an idealized source of stimulation for heart muscle.)

As the clock cycles, a one is output followed by $T-1$ zeros followed by a one indicating the start of the next cycle. It is assumed that the range of the clock period T is finite, that is $P(T=t)$ is zero for t outside the integers $1, 2, \ldots, t_{\max}$. A clock model with this period may be constructed which is similar to the Markov chain known as the renewal chain.

Table 3.1. State transitions for the clock model

Present state	Output	Next state	Probability
w	0	$w+1$	$P(T>w+1\|T>w)$
w	1	0	$P(T=w+1\|T>w)$

Table 3.2. Extended state transitions for the clock model

Starting state	Output sequence	Final state	Probability
w	$0^{n-1}1$	0	$P(T=w+n\|T>w)$
w	0^n	$w+n$	$P(T>w+n\|T>w)$

The state space for this model is the integers $0, 1, \ldots, t_{\max}-1$. The transitions for this model are shown in Table 3.1.

It will be useful to extend this table to output sequences as follows: Starting in state w the probability that the model will output a sequence of the form $0^{n-1}a$, where a is 0 or 1, and enter the final state w' is shown in Table 3.2.

3.5.2 The Clock Pulse Detector

The clock model described outputs a binary string $\underline{a}$. This string is transformed into an analog signal. When the clock produces a one output a pulse is generated on the analog signal. It is assumed that the shape of this pulse is known and that the time between pulses is large compared to the width of the pulse shape. (These pulses represent the deflections produced on the ECG when a heart muscle is activated by a source of stimulation).

It is assumed that a clock pulse detector is available which reliably detects the presence of the clock pulse shape on the analog signal, although false detections may occur. The output of the detector is a binary sequence $\underline{d}=d_1d_2\ldots d_N$, where a one indicates a clock pulse has been detected and a zero no detection. From the general discussion before it is assumed that $P(\underline{d}|\underline{a})=\prod_{i=1}^{N} P(d_i|a_i)$, and that $P(d_i|a_i)$ is independent of i. Now it is assumed that an actual clock pulse is detected with probability one, that is $P(d=1|a=1)=1$ and $P(d=0|a=1)=0$. There is a possibility of false detections. Thus $P(d=1|a=0)=P_n$ and $P(d=0|a=0)=1-P_n$. These conditional probabilities are summarized in Table 3.3. (Detectors which approximate these qualities exist for the ECG signal [3.3]).

Table 3.3. Conditional probabilities $P(d|a)$ for the clock pulse detector

		d	
		0	1
a	0	$1-P_n$	P_n
	1	0	1

3.5.3 The Total System

The configuration of the total system is as shown in Fig. 3.6. The operation of the total system is described as follows: The state of the system at time i is described by the pair (a, w), where a is the clock output and w is the clock state. Then at time $i+1$ the system makes the transition to state (a', w'), and the symbol d' is observed at the detector output, where $P(a', w'|w)\, P(d'|a') > 0$.

Given the detector output d', define the set of output d' successors of a state (a, w) by $S(w, d') = \{(a', w') : P(a', w'|w)\, P(d'|a') > 0\}$. Note that the set of output successors of state (a, w) is independent of the previous clock output a. For a given state (a, w) the output d' successors and their probabilities may be determined from Table 3.4.

Note that when $d = 0$ there is only one transition possible since $P(d = 0|a = 1) = 0$. It will be of use in defining the search tree to extend the definition of the output successor set to detector outputs of the form $0^{\Delta t - 1}1$. For a given state (a, w) the output $0^{\Delta t - 1}1$ successors may be determined from Table 3.5.

The probabilities given in Table 3.5 represent $P(\underline{a}, \underline{w}|w)\, P(\underline{d}|\underline{a})$ where $\underline{a} = 0^{\Delta t - 1}a'$, $\underline{w} = (w+1)(w+2)\ldots(w+\Delta t - 1)w'$, and $\underline{d} = 0^{\Delta t - 1}1$.

3.6 The Solution to the Clock Signal Interpretation Problem

3.6.1 The Search Tree

Given a detector output sequence $\underline{d}$ the search tree as previously defined may be used to find a pair of sequences $(\underline{a}, \underline{w})$ for which $P(\underline{a}, \underline{w}|w_0)\, P(\underline{d}|\underline{a})$ is maximum. Now, let $t_0 = 0$ and let t_i, for $i = 1, 2, \ldots, N$, be the i-th time the detector output was nonzero. Then let $\Delta t_i = t_i - t_{i-1}$. The detector output may be specified by a string of integers $\Delta t_1 \Delta t_2 \ldots \Delta t_N$ which give the times between nonzero detector outputs.

Using the previous definition of the search tree each node is labeled by a clock output and state pair (a, w). The label on the root is (a_0, w_0).

Table 3.4. Output successors of state (a, w)

Present state (a, w)	Output d'	Next state (a', w')	Probability $P(a', w'\|w)P(d'\|a')$
(a, w)	0	$(0, w+1)$	$P(T > w+1\|T > w)P(0\|0)$
(a, w)	1	$(0, w+1)$	$P(T > w+1\|T > w)P(1\|0)$
(a, w)	1	$(1, 0)$	$P(T = w+1\|T > w)P(1\|1)$

Table 3.5. Output $0^{\Delta t - 1}1$ successors of state $(a, w) : S(w, 0^{\Delta t - 1}1)$

State at time t (a, w)	State at time $t + \Delta t$ (a', w')	Probability
(a, w)	$(0, w + \Delta t)$	$P(T > w + \Delta t\|T > w)P(0\|0)^{\Delta t - 1}P(1\|0)$
(a, w)	$(1, 0)$	$P(T = w + \Delta t\|T > w)P(0\|0)^{\Delta t - 1}P(1\|1)$

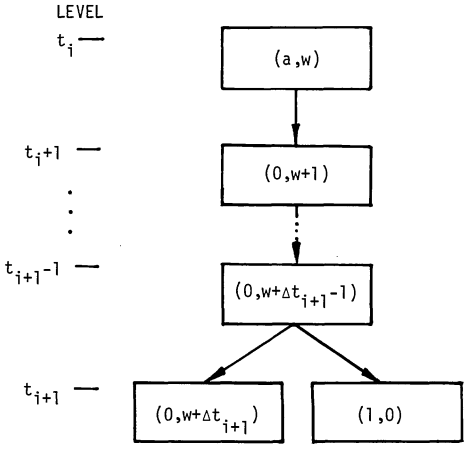


Fig. 3.7. Descendants of a node labeled (a, w) on level t_i of the search tree

A typical portion of the tree is illustrated in Fig. 3.7, showing the transitions from a node v labeled (a, w) on level t_i, to its descendant nodes on level t_{i+1}.

Note that on each of the levels t_i+1, $t_i+2, \ldots, t_{i+1}-1$, there is only one descendant of the node v. This is because no uncertainty about either the model state or model output occurs while the detector output is zero. These nodes need not be included in constructing the search tree since their presence may be inferred.

3.6.2 The Abbreviated Tree Structure

An abbreviated search tree is defined as follows. The set of nodes is designated V. The root is an element of V designated v_0. Each node $v \in V$ labeled with a clock output and state pair (a, w). The label on the root is (a_0, w_0). (Note that it is not required to specify the value of a_0.)

The nodes in the tree are arranged into levels. Let V_i denote the set of nodes on level i, for $i=0, 1, \ldots, N$. V_0 consists of the root only. A node on level i of the tree labeled (a, w) indicates the possibility that at time t_i the clock output and state pair was (a, w). The successors of $v \in V_i$ are in V_{i+1}, for $i=0, 1, \ldots, N-1$. There are no successors of the nodes in V_N. If (a, w) is the label of a node $v \in V_i$ then for each (a', w') in the set $S(w, 0^{\Delta t_{i+1}-1}1)$ there is a node $v' \in V_{i+1}$ labeled (a', w'). An arc is drawn from node v to node v'. This arc represents the sequence of transitions $(0, w+1)(0, w+2)\ldots(0, w+\Delta t_{it1}-1)(a', w')$.

3.6.3 The Probability of a Node

The problem of assigning a probability value to a node in the tree may be simplified by the following considerations. Let $T=\{1, 2, \ldots, t_N\}$ be the set of times t that the detector output $\underline{d}$ was observed. Let $T_1=\{t_1, t_2, \ldots, t_N\}$ be the set of times t that the detector output was one. Let $T_0=T-T_1$ be the times t that the detector output was zero. Now suppose $\underline{a}$ is a clock output sequence such

that $P\,\underline{d}|\underline{a} > 0$. since $P(d=0|a=1)=0$ we must have $a_t=0$ for $t \in T_0$. Then

$$P(d|a) = \prod_{t \in T} P(d_t|a_t) = \prod_{t \in T_0} P(0|a_t) \prod_{t=T_1} P(1|a_t)$$
$$= \prod_{t \in T_0} P(0|0) \prod_{t \in T_1} P(1|a_t).$$

Now $\prod_{t \in T_0} P(0|0)$ is constant for each clock output sequence $\underline{a}$ of length t_N for which $P(\underline{d}|\underline{a}) > 0$. Thus, it need not be included in the computation of the probabilities. It is sufficient to consider only strings $\underline{a}$ for which $a_t=0$ for $t \in T_0$, and to maximize the quantity $P(\underline{a}, \underline{w}|w_0) \prod_{t \in T_1} P(d_t|a_t)$.

Then for each node v in the tree the probability value associated with v is $\varrho(v)$ defined as follows: For the root $\varrho(v_0)=1$. Let v be in V_{i-1}, for $i=0, 1, \ldots, N-1$, with label (a, w), and let $v' \in V_i$ be a successor of v labeled (a', w'). Let $(\underline{a}', \underline{w}')$ be the sequences represented by the arc from v to v'. The probability associated with node v' is the given by $\varrho(v') = \varrho(v) P(\underline{a}', \underline{w}'|w) P(d_{t_i}|a')$.

3.6.4 Summary of the Tree Structure

In summary, if (a, w) is the label on a node v on level $i-1$ of the tree with associated probability $\varrho(v)=q$, then the node has two potential successors on level i. Let $S=p(d=1|a=1)=1$ and $\bar{S}=P(d=1|a=0)=P_n$. Let $A=P(T=w+\triangle t_i|T>w) = P(T=w+\triangle t_i)/P(T>w)$ and $\bar{A}=P(T>w+\triangle t_i|T>w)=P(T>w+\triangle t_i)/P(T>w)$.

The first potential successor is $(1, 0)$ and has associated probability $q_1=qSA$. The second is $(0, w+\triangle t_i)$ with associated probability $q_2=q\bar{S}\bar{A}$.

As before, if two nodes occur on the same level of the tree having the same clock state label w then it is necessary only to consider the successors of the more probable node.

3.7 A Sample Clock Signal Search Tree

A numerical example has been generated to illustrate this type of interpretation problem. The clock model random period T was chosen to range from 17 to 25 time units. The density $P(T=t)$ is graphed on the bottom of Fig. 3.8. The probability of a false detection was chosen to be $P_n=1/10$.

The times when the detector output was one are shown in Fig. 3.8. This data represents 5 clock pulses with periods selected randomly according to the density $P(T=t)$. There are also four false detections present. The locations of these detections were selected randomly on the interval 0 to 100 with a uniform density. Figure 3.8 shows the search tree which is generated in the interpretation of this data. Logarithms of the ϱ values are used to simplify the computations by replacing multiplication with addition and to reduce the range of the numbers involved.

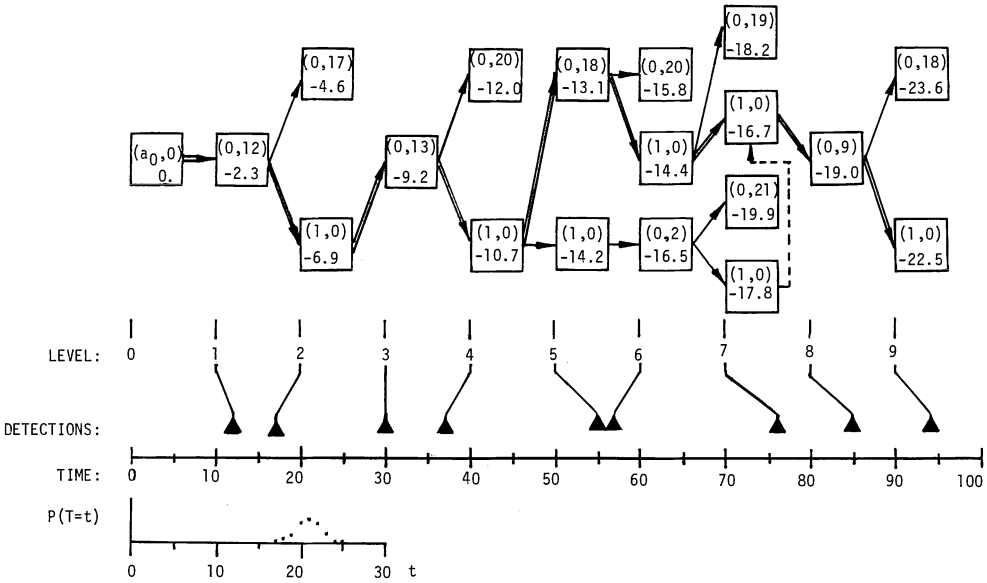


Fig. 3.8. Search tree for the clock pulse interpretation example. The density of the clock period, $P(T=t)$, is shown at the lower left. The dots indicate the times t for which $P(T=t)>0$

The i-th level in the tree corresponds to the i-th time the detector output was one. The first time that this occurs is at time $t=12$. Since the clock was in state 0 at time $t=0$, this detection could not represent a clock pulse. Thus, the only successor of the root on level 1 is the node labeled (0, 12), indicating clock output $a=0$ and clock state $w=12$. The next detection occurs 5 time units later. This may or may not indicate a clock pulse. Thus, there are two successors of the node on level 1. However, since the next detection occurs 13 time units later it must assumed that the second detection actually indicated a clock pulse. In a similar way it is concluded that detection number 3 is a noise artifact, and detection number 4 indicates a clock pulse. Now detections 5 and 6 occur close together in time and either one may be a clock pulse. The interpretation of detection 5 as the clock pulse is represented by the lower nodes on levels 5 and 6, and the interpretation of detection 6 as a clock pulse is reprensented by the upper nodes. When detection 7 occurs some time later these two paths converge on the same clock state and the one with the lower probability is rejected. This results in choosing detection number 6 as representing the true clock pulse. Since detection 8 occurs too late after detection 6 and too early after detection 7, it is concluded that this detection is a noise artifact. Finally, the node on level 9 with the highest probability indicates that detection 9 is a true clock pulse.

After considering the possibility that each detector output is either a clock pulse or a noise artifact, the procedure chooses detections 2, 4, 6, 7, and 9 as representing actual clock pulses. In generating this data, detections 2, 4, 5, 7, and 9 were the actual clock pulses. Thus, when the ambiguity between detections 5 and 6 occurred the procedure made an error.

3.8 Conclusion

The clock signal decision procedure can reduce the effects of occasional noise artifacts on the analysis of a periodic signal. Both the width of the search tree and the error rate will increase with the rate of false detections.

More complex models may be constructed by using several clocks which interact through mutual inputs and outputs. A model for the genesis of a common set of ECG rhythms, which consists of three interacting clocks, has been constructed. This model has been tested for use in interpretation intra-cardiac ECG signals. These initial tests have demonstrated the ability of the procedure to correctly interpret the signals in the presence of noise.

Acknowledgement. This study was supported in part by Contract No. PH 43-68-1333 under Myocardial Infraction Research Program, National Institutes of Health, National Heart and Lung Institute, Healt, Education and Welfare.

References

3.1 K. S. FU: *Syntactic Methods in Pattern Recognition* (Academic Press, New York 1974)
3.2 N. NILSSON: *Problem-Solving Methods in Artifical Intelligence* (McGraw-Hill, New York 1971)
3.3 COX, JR.: Proc. IEEE, **60**, 1137 (1972)

4. Syntactic Recognition of Speech Patterns

R. DeMori

With 9 Figures

Syntactic methods can be profitably used for automatically interpreting speech patterns because most of the phonetic knowledge of a language is available in terms of descriptions of spectral patterns or linguistic events.

This chapter deals with the problem of representing and using such knowledge in a digital computer.

4.1 Background

Automatic speech recognition has been the object of several investigations in the last two decades. The reasons for this interest reside in the importance of such potential applications as voice-controlled telephone dialing, phonetic typewriters, man-machine voice communication, monitoring of voice commands for air traffic controllers, information retrieval through telephone lines, question-answering systems and data base management.

Another attractive aspect of research in speech recognition is the attempt to conceive a model for speech perception where several sources of knowledge like phonetic rules, lexicon, syntax, semantics and pragmatics are used for gathering the meaning of a verbal message. The work done in speech recognition from the beginning until 1970 has been mostly concerned with techniques for signal analysis, feature extraction and adaptive pattern matching.

An excellent critical survey of the work has been provided by Hyde [4.1]; his conclusions share some points made by Pierce [4.2] which stimulated new directions for research in this area. Later, a concrete proposal for new investigations in speech understanding systems was formulated in a book [4.3] reporting the considerations elaborated by a qualified study group. The main findings are that advances in automatic speech recognition will come from a better use of the knowledge attained by research in natural language processing, speech perception and phonetics.

Some of the aspects that make automatic speech recognition a difficult problem concern the variability of the features extracted from the same message pronounced many times by the same talker, the talker and the context dependency of such features, the selection of speech units, and the ambiguities of the features that can be extracted. Such ambiguities make speech understanding a process by which the meaning of a sentence belonging to a spoken natural language has to be grasped from a set of corrupted and partially undecoded hypothetic transcriptions of the verbal message.

The use of syntactic methods in speech recognition and understanding has some important motivations related to experiments in speech perception.

Important works by Liberman [4.4] and Studdert-Kennedy [4.5] show that speech patterns are related to linguistic items by a complex set of rules belonging to a "grammar of speech". Such rules mainly concern the pattern complexity due to the effects of coarticulation.

The description of the vocal tract behavior, its models, and the coarticulation effects can be found in a book by Flanagan [4.6].

Investigations on coarticulation [4.7] made evident an aspect that motivates the use of syntactic methods in recognizing speech patterns. It was established that an utterance is organized as a succession of vocal tract states. The states are not individually finalized during the pronounciation of a sentence, continually exhibiting transients in the representative parameters.

These facts have been described by phoneticians more in terms of relations between subpatterns extracted from spectrograms than in terms of sets of numerical measurements taken from the speech waveforms. At least two fundamental studies related to "vowel reduction" [4.8] and "vowel-consonant-vowel coarticulation" [4.7] support this point. These phenomena are characterized by descriptions of forms extracted from the speech patterns, and the most suitable way for detecting and recognizing them is by syntactic methods. Moreover, syntactic approaches have been successfully applied to the automatic segmentation of continuous speech [4.9–11], to the recognition of words and sentences [4.12–15] and to processes of automatic emission and verification of syllabic hypotheses in continuous speech [4.16–18].

After having mentioned motivations and applications of syntactic methods to speech processing and recognition, it is important to notice that some pattern dimensions are also essential to the interpretation of the patterns themselves. Furthermore, knowledge of the statistics of the pattern forms and dimensions is also important for classification. Thus, syntactic speech recognition has to be studied in the framework of stochastic languages with associated semantic rules concerning pattern measurements.

Facing the problem with the use of stochastic languages it is possible to employ available learning techniques for constructing the sources of knowledge by grammatical inference [4.19]. Such sources of knowledge contain rules representing a structured description of the many different realizations of a phonemic sequence pronounced by different talkers and in different situations. Thus a concise set of rules takes the place of a large collection of template patterns.

The problem of computing the distance between the set generated by the grammar corresponding to a given sequence of phonemes and an unknown description not generated by the grammar is solved by algorithms developed recently [4.20] for computing similarities between strings and languages.

The arguments produced so far do not limit the problem of speech understanding only to the area of syntactic pattern recognition. Other sources of knowledge and related methodologies for emitting and verifying hypotheses are involved. They concern phonological rules, syntax, semantics, pragmatics and stochastic models of natural languages. These aspects cover a large area of artificial intelligence and will not be treated in this chapter. A good overview of them can be found in [4.21] and further details in other papers [4.22–44, 4.76].

An interesting experiment was performed by KLATT and STEVENS [4.27] to show how much the high level sources of knowledge may improve the recognition of words based on inspection of spectrograms. In their experiment the authors attempted to recognize words reading spectrograms of spoken sentences. Using their human knowledge of the language, they succeeded in 97% of the cases even if they were able to perform only a partial phonetic transcription of the spectrograms using phonetic rules; this operation was affected by an error of about 25%.

4.2 Preprocessing and Feature Extraction

4.2.1 Signal Processing

The speech message is generally available as a time waveform. The features that are significant for automatic recognition are generally extracted after some pre-processing consisting of filtering the signal with a number of filters varying from some units to some tenth and then taking the zero-crossing counts and/or the energy at the output of each filter. Applications that use such preprocessing are reported, for example, in [4.9] and [4.15]. More accurate techniques, now implemented by digital computers, use numerical methods for processing the sampled speech waveform. Such methods consider a running window on the digital waveform and compute a short-time spectrum for each "windowed" portion of the waveform by the fast Fourier transform (FFT) algorithm. The duration of the window can be made equal to a multiple of the pitch period (the period of the vocal cord vibrations) for those sounds that are called "voiced" because they are pronounced with the vocal cords exciting the vocal tract and, eventually, the nostrils.

Other speech analysis methods consist of modeling the vocal tract with a transversal filter or with a discrete model having a transfer function with poles and zeros. Using these methods, smoothed versions of the short-time spectra of the speech waveform can be obtained. Such smoothing should remove the influence of the glottal excitation evidencing the effects of the vocal configurations that mainly depend on the speech sounds composing the verbal message. Details of such methods can be found in [4.6, 45–49, 4.77].

The final product of a speech analysis system operating in the frequency domain is a spectrogram. A spectrogram is a plot of energy as a function of time and frequency where usually time and frequency are the main dimensions of a picture; energy is represented by gray levels. Spectrograms can be generated either by analog machines or by digital computers. Although the performance of the two methods is comparable, digital methods are more flexible and more suitable for further processing.

Digital processing of speech is generally very time consuming, but the processing can be sped up by using special-purpose processors. Available solutions and future trends can be found in [4.50–52, 4.69].

The main reason why spectrograms are considered the basis for extracting speech patterns is the fact that speech production can be described for many

sounds in terms of vocal tract resonant frequencies called formants. Evidence exists that the ear makes a crude frequency analysis in processing speech [4.6]. Furthermore most of the results obtained by research on experimental phonetics are formulated in terms of rules describing forms in the spectrograms.

4.2.2 Trajectories of Speech Parameters

Features extracted for automatic recognition depend on the purposes of the system. For the recognition of a limited vocabulary of isolated words, there are systems for which it is sufficient to extract the energies in some frequency bands. A spoked word is thus represented by a pattern made of frames. Each frame corresponds to a time interval of fixed duration (usually 10 to 20 ms), and a frame is a set of energies in fixed frequency bands. The source of knowledge is represented by one or more typical pattern for each word of the lexicon, and recognition is performed after computing a correlation of the pattern to be classified with the typical patterns. Usually the comparisons cannot be performed in simple ways because the speech units of a word, the phonemes, may have durations varying considerably for different utterances of the same word. For this reason, the pattern matching is better performed sequentially using algorithms based on dynamic programming. Interesting examples of such approaches can be found in [4.53–55].

Other approaches contain a step of feature extraction from spectrograms to reduce information before recognition. In some cases this process is the same for every type of phoneme: in other cases feature extraction depends on a precategorical classification of speech intervals into broad classes [4.75].

Among the applications where the reduction of dimensions is performed in the same way for all the frames, it is interesting to recall briefly a recognition system for isolated words proposed by Pols [4.56]. Here a word is represented as a sequence of points in a multidimensional space. The 17 coordinate values of the points are the logarithms of the energies detected at the output of 17 band-pass filters every 15 ms. Using a method called "principal component analysis", a three-dimensional subspace is derived which explains most of the variance (78%). The unknown word is represented as a sequence of points in a three-dimensional space; the points define a trace that is linearly time normalized and compared with reference patterns using probability scores. Traces obtained in a similar way are used in [4.13] for recognizing, by a syntactic procedure, liquid and glide linguistic sounds.

A similar procedure for reducing dimensions was proposed by Pau [4.57], resulting in an efficient method for removing most of the talker-dependent information. The papers [4.56–58] also contain some interesting non-syntactic methods to evaluate the similarity between speech traces and reference sets of points.

Speech traces of syllables and words are also suitable to be described by an artificial language and interpreted with syntactic methods using the procedure proposed in Section 4.4.

4.2.3 Distinctive Features

More sophisticated feature extraction can be made by considering that the phonetic properties of a spoken language have been often expressed in terms of the so-called "distinctive features". The basic principle underlying such an approach is that each phoneme can be represented by a concatenation of binary variables, each one expressing the value of a distinctive feature. For example one of such features could be vocalic-nonvocalic, and the corresponding variable assumes value one for the vowels and zero for the consonants. All the phonemes having a feature in common belong to a group characterized by some property in terms of the human production process. This aspect in relation to speech understanding was recently discussed by KLATT [4.59].

A generative model of a spoken language can be built up starting from a set of distinctive features. The same model could be used as basis of a strategy for understanding a spoken message after representing a phoneme and the co-articulation effects on it in terms of distinctive features. Such an attempt would be a syntactic method because recognition is controlled by the rules that relate words and phrases to sets of distinctive features. One of the most interesting efforts in this direction was done by HUGUES and HEMDAL [4.60]. They proposed a set of distinctive features on the basis of which phoneme recognition were performed according to a binary tree. Using such a scheme, automatic recognition is carried out by a set of processors operating independently; a single processor is required for each feature of the classifactory system.

Passing from theory to practice, it was found very difficult to extract the distinctive features from the spectrograms or from the speech wave. Methods based on energy measurements in some frequency bands were proposed by HUGUES and HEMDAL [4.60] only for some features.

Evidence available today appears to indicate that there is no simple transformation from the cues directly obtainable by processing the speech signal to the phonetic features. Rather, a complex encoding takes place where information regarding a particular feature of a segment may be carried by neighboring segments [4.10]. Nevertheless, some features like vocalic-nonvocalic exhibit very often some aspects that are affected very little by the context.

For example, the nonsonorant sounds (i.e., all the consonants except nasals n, m, η, liquids l, r and glides j, w) have considerable energy in the $5 \div 10$ kHz band while the sonorants do not. Furthermore, vocalic sounds exhibit peaks in the signal energy and have considerable energy in the $3 \div 5$ kHz band while the sonorant nonvocalic ones do not. Such measurements can be performed after preprocessing the speech waveform with band-pass filtering, and the results may be used for driving the successive steps of analysis and feature extraction. In fact, for the sonorant sounds, except the nasals, most of the linguistic information is contained in the formants; thus for such sounds the use of inverse filtering for computing short-time spectra is more appropriate. This method smooths the spectra, giving peaks of energy concentration at formant frequencies and making them easily detectable. Among the sonorant sounds, the nasals exhibit in the spectra the effects of some antiresonances due to the nasal tract. Nasality can be detected in many cases by a procedure that is insensitive to the context,

which consists of analyzing the behavior of the first formant frequency and its amplitude [4.61–62]. In fact, all the voiced consonants except nasals exhibit large dips in the first formant amplitude. For nasal segments and their neighboring segments, a more appropriate analysis can be performed by using models of the vocal apparatus having zeros as well as poles [4.46] or an all-poles model having more poles (22 for example) than the model used for the other sonorant sounds (8 to 16 poles) [4.63].

Finally, important linguistic information for the nonsonorant sounds is spread in a frequency band varying from 200 Hz to 10 kHz and over, while the sonorant sounds have their main features in the band from 200 Hz to 5 kHz.

An analysis procedure driven by features that can be detected extracting cues that are little affected by the context is complex and ought to be used only for the recognition of a large vocabulary or for continuous speech. There are applications with small lexicons for which syntactic methods can be applied after a simple preprocessing that can be done in real time and with partial use of analog circuits. Nevertheless, the extraction of easily detectable features can be fruitful also for applications with limited lexicons. Some features can also be extracted by syntactic analysis of signals obtained by preprocessing the speech waveform. Examples of syntactic feature extraction with methods requiring small computation and working close to real time will be introduced in the next section.

Recent significant efforts in feature extraction are reported by Stevens [4.64]. Interesting speech recognition systems, based on distinctive features, has been designed by Bobrow and Klatt [4.65] and Itahashi et al. [4.66]. An other interesting application using more complex features extracted by networks of threshold elements has been proposed by Martin [4.67].

4.3 Syntax-Controlled Segmentation of Continuous Speech

Segmenting speech before automatic recognition may not be necessary if the task is limited to a small vocabulary of isolated words. In such case, it is possible to assume the word as a unit and to consider the source of knowledge used for recognition made of a set of rules for each word. Avoiding segmentation may be advantageous because this operation introduces errors as does every other step involved in speech recognition. Nevertheless, segmentation cannot be avoided if the lexicon is large or if continuous speech has to be recognized. Furthermore, if segmentation is performed on the data available after preprocessing the speech waveform, the segments obtained can be used to select an analysis technique that is more effective for a particular segment. A crucial point is the selection of the units to be considered as segments. The difficulties concern the reliability of detecting the bounds and the efficiency of classification of such segments.

One solution is that of considering phonemes as units. A recognition system based on this approach was proposed by Reddy [4.9]. The advantage of attempting to recognize phonemes is that the set of units is relatively small (less than 100). Moreover there is a disadvantage because phoneme features are remarkably affected by coarticulation, and the context plays an important

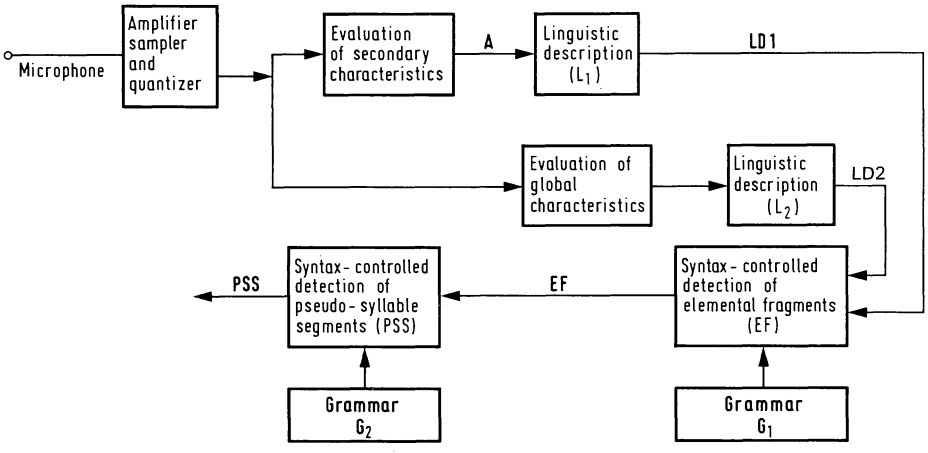


Fig. 4.1. Block diagram of a syntactic procedure for segmenting continuous speech into pseudo-syllable segments

role in their extraction and classification. For this reason, it seems more appropriate to consider as units some segments having the dimension of a syllable. These segments should contain the coarticulation effects appearing as transients between two stationary portions of the speech parameters. Let call such segments "pseudo-syllable segments" (PSS).

A syntax-controlled procedure for segmenting the Italian language into pseudo-syllable segments will be described in the following. It is implemented according to the block diagram of Fig. 4.1 proposed in [4.11].

Another procedure for extracting syllable nuclei from messages spoken in English has been suggested by MERMELSTEIN [4.10]. The parameters used for segmenting Italian sentences are the logarithm $A(n)$ of the rms amplitude of the speech waveform; the durations, the logarithm $L_v(n)$ of the energy in the 3–5 kHz band normalized with respect to the amplitude in an interval of 100 ms around the considered time frame; and the logarithm of the ratio $R_v(n)$ between the energy in the $200 \div 1000$ Hz band and the energy in the $5 \div 10$ kHz band. Each frame has a duration of 10 ms and the variable n is assumed as index of the frames.

The curve $A(n)$ is described by a sequence of horizontal (h), ascendent (a) and descendent (d) lines, using a procedure that is proposed in [4.11]. The evolution of $A(n)$ when a sentence is pronounced is thus represented by a string of symbols belonging to the alphabet:

$$V_1 = \{\mathrm{h}, \mathrm{a}, \mathrm{d}\}\,.$$

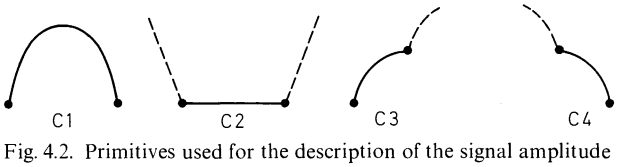


Fig. 4.2. Primitives used for the description of the signal amplitude

Each symbol is followed by the time reference of its last frame. Operating on the string corresponding to a sentence, the four primitive forms represented in Fig. 4.2 are researched by a finite state automaton whose design is based on the following rewriting rules.

P1. C1→ahd

C1→ad

dC2a→dha

C3a→aha

dC4→dhd.

Once a primitive has been detected, the corresponding piece of the $A(n)$ waveform, which will be referred to as an atom, is described by the primitive symbol followed by some attributes. Such description with attributes allows one to reconstruct the amplitude waveform with polynomial approximation. The description so obtained will be indicated by LD1 and the corresponding language will be referred to as L_1.

The two parameters $L_v(n)$ and $R_v(n)$ will be referred to as global spectral characteristics (GSC). They can be obtained by filtering the time waveform. This operation can be performed by digital filters or Fourier transformation. Reference [4.52] describes a special purpose computer that performs such operations in real time.

Earlier experiments in acoustic phonetics showed that there are significant high frequency components for unvoiced consonants and vowels, so $L_v(n)$ is used for distinguishing unvoiced consonants and vowels from other consonants. Moreover $R_v(n)$ is used for distinguishing between voiced and unvoiced tracts.

The time evolutions of $L_v(n)$ are classified into ascendent, descendent and horizontal tracts and each tract has the attribute "voiced" or "unvoiced" according to the corresponding value of $R_v(n)$. Thus the GSC evolutions of a spoken sentence are represented with a string of symbols belonging to the following alphabet:

$$V_2 = \{A1, A0, D1, D0, H1, H0\};$$

where the symbols have the following meaning:

A1: voiced ascendent tract;

A0: unvoiced ascendent tract;

D1: voiced descendent tract;

D0: unvoiced descendent tract;

H1: voiced horizontal tract;

H0: unvoiced horizontal tract.

The description is then processed to make evident two primitive forms, indicated as S1 and S2, that have been found useful for isolating vowels and unvoiced tracts in connected speech. These primitives are evidenced by two automata operating in accordance with the following rules:

P2.

$$y \rightarrow (\mathrm{A1})$$
$$y \rightarrow (\mathrm{A1})\,\xi_1$$
$$v \rightarrow (\mathrm{A0})$$
$$v \rightarrow (\mathrm{A0})\,\xi_2$$
$$z \rightarrow (\mathrm{D1})$$
$$z \rightarrow (\mathrm{H1})\,(\mathrm{D1})$$
$$z \rightarrow (\mathrm{H1})\,(\mathrm{D1})\,\xi_3$$
$$w \rightarrow (\mathrm{D0})$$
$$w \rightarrow (\mathrm{H0})\,(\mathrm{D0})$$
$$w \rightarrow (\mathrm{H0})\,(\mathrm{D0})\,\xi_4$$
$$\xi_1 \rightarrow (\mathrm{H1})\,(\mathrm{A1})$$
$$\xi_1 \rightarrow (\mathrm{H1})\,(\mathrm{A1})\,\xi_1$$
$$\xi_2 \rightarrow (\mathrm{H0})\,(\mathrm{A0})$$
$$\xi_2 \rightarrow (\mathrm{H0})\,(\mathrm{A0})\,\xi_2$$
$$\xi_3 \rightarrow (\mathrm{H1})\,(\mathrm{D1})$$
$$\xi_3 \rightarrow (\mathrm{H1})\,(\mathrm{D1})\,\xi_3$$
$$\xi_4 \rightarrow (\mathrm{H0})\,(\mathrm{D0})$$
$$\xi_4 \rightarrow (\mathrm{H0})\,(\mathrm{D0})\,\xi_4$$

P3.

$$\mathrm{S1} \rightarrow yz$$
$$\mathrm{S1} \rightarrow y(\mathrm{H1})$$
$$v\mathrm{S1} \rightarrow vz$$
$$v\mathrm{S1} \rightarrow v(\mathrm{H1})$$
$$v(\mathrm{H0})\mathrm{S1} \rightarrow v(\mathrm{H0})\,(\mathrm{H1})$$
$$v(\mathrm{H0})\mathrm{S1} \rightarrow v(\mathrm{H0})\,z$$
$$\mathrm{S2} \rightarrow vw$$
$$\mathrm{S2} \rightarrow v(\mathrm{H0})$$
$$y\mathrm{S2} \rightarrow yw$$
$$y\mathrm{S2} \rightarrow y(\mathrm{H0})$$
$$y(\mathrm{H1})\mathrm{S2} \rightarrow y(\mathrm{H1})\,(\mathrm{H0})$$
$$y(\mathrm{H1})\mathrm{S2} \rightarrow y(\mathrm{H1})\,w\,.$$

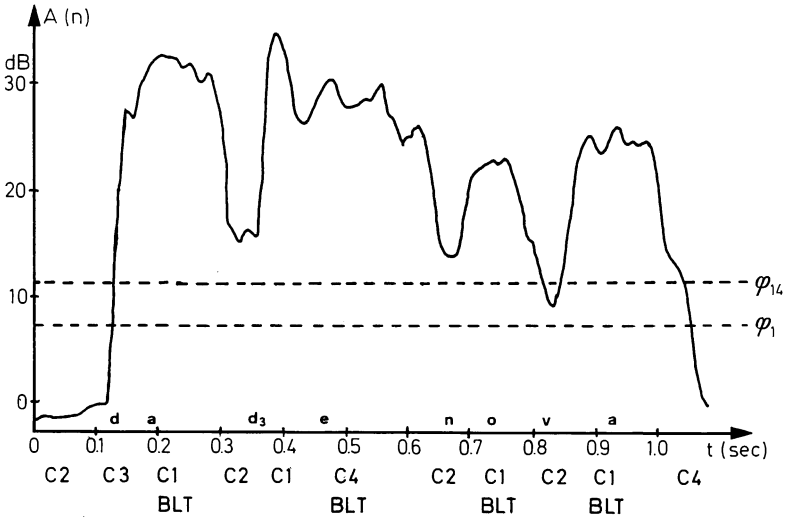


Fig. 4.3. Time evolution of the signal amplitude $A(n)$ and its description for the sample sentence *da dʒenova* (from Genoa)

The rules P2 and P3 are applied in a way such that the greatest possible form corresponding to the left side symbol is detected and described. A GSC description will be referred to as (*LD* 2) and the corresponding language will be indicated as L_2.

Figure 4.3 shows the time evolution of the signal $A(n)$ and its description. Figure 4.4 shows the time evolution of the signal $L_v(n)$ and its description. The sentence pronounced is "da Genova" (from Genoa) that can be written with the phonetic alphabet as *da dʒenova*.

The segmentation procedure ought to isolate coarticulation segments on the basis of the detection of some phonetic classes indicated as elemental fragments (EF). Such fragments, as well as the rules defining the coarticulation segments, depend on the language. For the Italian language it has been found that a satisfactory segmentation grammar can be designed using the following elemental fragments:

V: vowels;

UT: unvoiced tracts;

SL: silences;

VC: tracts of one or more voiced consonants.

Moreover, the rules of the segmenting grammar that will be introduced later on show that a VC tract needs to be detected only when a voiced consonant preceeds a silence or an unvoiced consonant.

The elemental fragments are related to the elements of the descriptions of the secondary characterics and the GSC by a set of productions that has

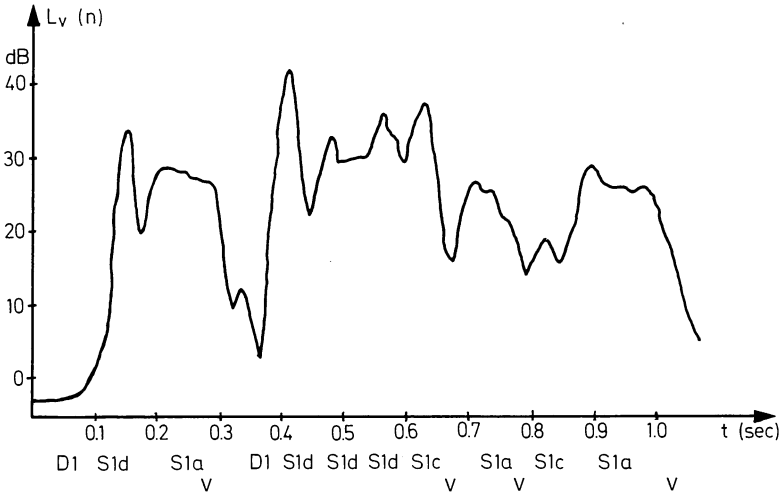


Fig. 4.4. Time evolution of the energy in the 3 ÷ 5 kHz band and its description for the sample sentence *da dʒenova* (from Genoa)

been inferred from experiments with a set of sentences belonging to a protocol for an automated reservation and travel information system. The details of such experiments are reported in [4.11]. The productions inferred by such experiments required that some conditions on the numerical attributes of the atoms be verified. Further experiments with other sentences and other speakers showed that while the numerical relations of the attributes are speaker dependent, the definitions of EFs in terms of qualitative symbols are not.

On the basis of the above-mentioned experiences, the following rules appear to be the best definitions of EFs.

P4. (SL)→CS

(UT)→((BLT), (UTA)); {inside (BLT), (UTA)}

(V)→((BLT), (VSC)); {inside (BLT), (VSC)}

(V) (VC) (UN)→(V) (VD1) (UN)

(VD1)→((BLT), (D1S)); {inside (BLT), (D1S)} .

The productions P4 contain new nonterminal symbols that will be defined later and the predicate: inside (X, Y). This predicate applies to a phrase X of L_1 and a phrase Y of L_2 and indicates that the event described by Y begins after the beginning of the event X and ends no more than two frames after the end of X. A comma in the productions separates a phrase of L_1 from a phrase of L_2, and it is intended that two or more different events Y_i that are inside X may generate more fragments, one for each pair (X, Y_i) that can be rewritten with an EF by P4.

As was mentioned, the VC fragment is defined only when it is useful for segmentation, that is in the context of a vowel and a silence or an unvoiced consonant. The other nonterminals in P4 are defined as follows[1]:

P5. $\mathrm{CS} \rightarrow \mathrm{C2}\ \{(\mathrm{amplitude} \leqq \varphi_1) \wedge (\mathrm{duration} \geqq \varphi_2)\}$

$(\mathrm{BLT}) \rightarrow (\mathrm{BLA})\ \{\mathrm{duration} \geqq \varphi_3\}$

$(\mathrm{BLA}) \rightarrow (\mathrm{C1V})$

$(\mathrm{BLA}) \rightarrow (\mathrm{BLR})\ (\mathrm{C1V})$

$(\mathrm{BLA}) \rightarrow (\mathrm{C1V})\ (\mathrm{BLP})$

$(\mathrm{BLA}) \rightarrow (\mathrm{BLR})\ (\mathrm{C1V})\ (\mathrm{BLP})$

$(\mathrm{C1V}) \rightarrow (\mathrm{C1})\ \{a_{\mathrm{m}} \geqq \varphi_4\}$

$(\mathrm{BLR}) \rightarrow (\mathrm{C3})$

$(\mathrm{BLR}) \rightarrow (\mathrm{C3})\ (\mathrm{BLR})$

$(\mathrm{BLP}) \rightarrow (\mathrm{C4})$

$(\mathrm{BLP}) \rightarrow (\mathrm{C4})\ (\mathrm{BLP})$

$(\mathrm{UTA}) \rightarrow \mathrm{S2}\ \{(\mathrm{duration} \geqq \varphi_6) \wedge (f_{\mathrm{m}} \geqq \varphi_7)\}$

$(\mathrm{VSC}) \rightarrow \mathrm{S1a}$

$(\mathrm{VSC}) \rightarrow \xi_5\,(\mathrm{S1c})$

$\xi_5 \rightarrow (\mathrm{VS1})$

$\xi_5 \rightarrow (\mathrm{VS1})\ \xi_5$

$\mathrm{S1a} \rightarrow \mathrm{S1}\ \{(f_{\mathrm{e}} < \varphi_{12}) \wedge (\mathrm{duration} \geqq \varphi_8) \wedge (f_{\mathrm{m}} \geqq \varphi_{11})\}$

$\mathrm{S1c} \rightarrow \mathrm{S1}\ \{(f_{\mathrm{e}} < \varphi_{12}) \wedge ((\mathrm{duration} < \varphi_8) \vee (f_{\mathrm{m}} < \varphi_{11}))\}$

$(\mathrm{VS1}) \rightarrow \mathrm{S1d}$

$(\mathrm{VS1}) \rightarrow \mathrm{D1d}$

$\mathrm{S1d} \rightarrow \mathrm{S1}\ \{f_{\mathrm{e}} > \varphi_{12}\}$

$\mathrm{D1d} \rightarrow \mathrm{D1}\ \{f_{\mathrm{e}} > \varphi_{12}\}$

$\mathrm{D1S} \rightarrow \mathrm{D1}\ \{(\mathrm{duration} \geqq \varphi_9) \wedge (f_{\mathrm{m}} \leqq \varphi_{10})\}$.

The productions P5 are valid only if some specified conditions on the attributes are verified; the φ_i values ($i = 1, 2, \ldots, 12$) are thresholds determined experimentally. The attributes used in P5 have the following definitions:

f_{e}: frequency of the last $L_{\mathrm{v}}(n)$ in the atom,

f_{m}: maximum frequency of the $L_{\mathrm{v}}(n)$ in the atom,

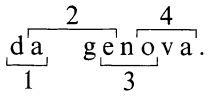
a_{m}: maximum amplitude of C1.

[1] $\wedge$ is the logical operator AND; $\vee$ is the logical operator OR.—The parentheses contain relations that have to be verified in order to make the rewriting rule valid.

Finally, segmentation is performed in accordance with the following rules:

P6.
(VLK)→(V)
(VLK)→(VC) (V)
(UN)→(SL)
(UN)→(UT)
(UN)→(SL) (UT)
(UN)→(UT) (SL)
UN→(UT) (SL) (UT)
(PSS) (VLK) →(V) (VLK) (VLK)
(PSS) (VLK)→(UN) (VLK) (VLK)
(PSS) (UN)→(V) (VLK) (UN)
(PSS) (UN)→(UN) (VLK) (UN)
(PSS) (UN)→(V) (VLK) (VC) (UN)
(PSS) (UN)→(UN) (VLK) (VC) (UN).

Referring to the example of Figs. 4.4 and 4.3, all the vowels are detected and the following four PSS are obtained:



The elemental fragments can be used together with other features to drive a more efficient analysis procedure and extraction of acoustic patterns.

A tree diagram for the sequential detection of distinctive-feature values is shown in Fig. 4.5. The detected fragments used for segmentation are processed

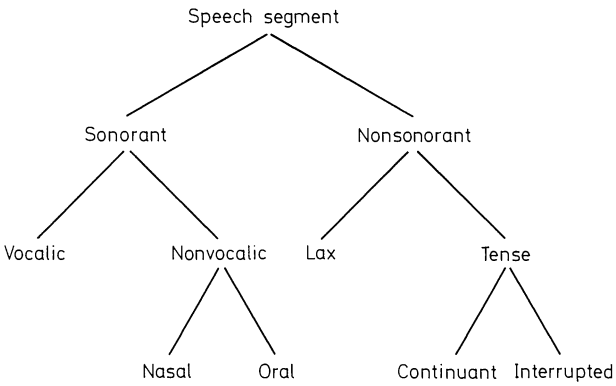


Fig. 4.5. Tree diagram representating distinctive features of speech sounds that can be detected with context-independent algorithms

under the control of the rules introduced in the following to give the binary values of each branch of the tree:

$$\begin{aligned}&(\mathrm{V})\quad [\text{nonsonorant-tense continuant}]\rightarrow(\mathrm{V})\,(\mathrm{UT})\\&(\mathrm{VC})\ [\text{nonsonorant-tense-continuant}]\rightarrow(\mathrm{VC})\,(\mathrm{UT})\\&\qquad [\text{nonsonorant-tense-interrupted}]\rightarrow(\mathrm{SL})\,(\mathrm{UT})\,.\end{aligned}\tag{4.1}$$

The nonsonorant-lax class is defined as follows. Let F_v be the normalized energy in the $5 \div 10$ kHz band. Let D_A be a measure of the dip of the amplitude curve in a logarithmic scale, then:

$$[\text{nonsonorant-lax}] = \mathrm{VC}\left\{\begin{matrix}(F_v > \varphi_{13}) \vee ((\varphi_1 < D_A < \varphi_{14}) \wedge \\ \wedge\,(\varphi_5 < R_v < \varphi_{16}))\end{matrix}\right\}\tag{4.2}$$

φ_{13}, φ_{14} and φ_{16} are thresholds; the second condition in the definition of the nonsonorant-lax class refers mainly to the sound [v] which has no considerably

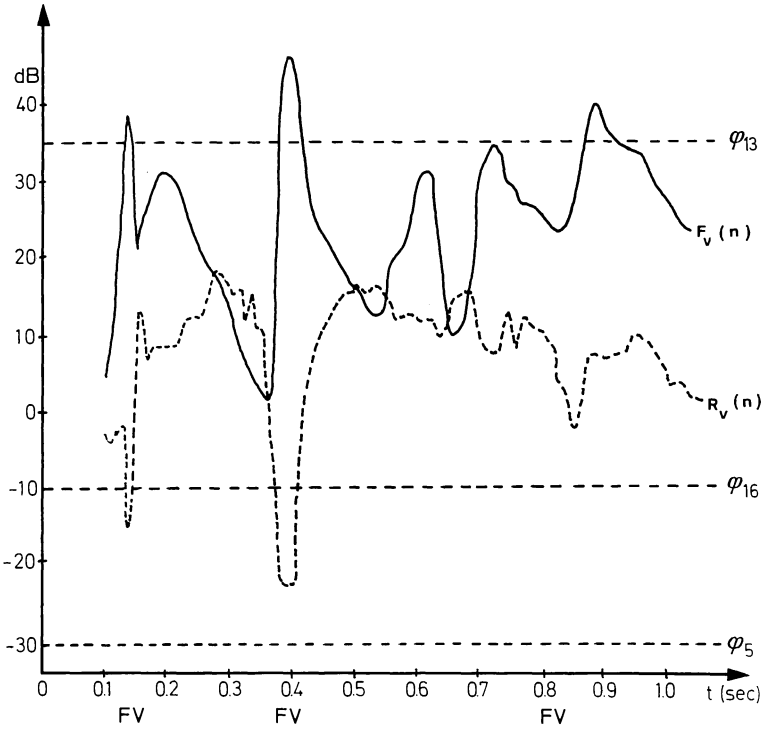


Fig. 4.6. Time evolutions of the ratio between low and high frequency energies ($R_v(n)$) and high frequency energy ($F_v(n)$) for the sample sentence *da dʒenova* (from Genoa)

high frequency components; φ_5 is the threshold used for distinguishing between voiced and unvoiced tracts.

Among the sonorants, the vocalic feature corresponds to the detection of a V fragment, while the nasal feature is detected considering the Boolean values corresponding to the truth of the following events:

— downward shift of the first formant frequency, or considerable dip in the amplitude curve;
— no dip in the corresponding interval of formant amplitude;
— presence of antiresonances or formant clusters depending on the method used for computing the spectrograms.

Figure 4.6 shows the time evolutions of $R_v(n)$ and $F_v(n)$ for the sample sentence "da Genova" and the detection of the nonsonorant sounds. The dip of [v] in the amplitude curve is shown in Fig. 4.3.

A more sophisticated procedure for classifying nonsonorant features is described in [4.75] and a more accurate algorithm for the distinction among nasals and orals is proposed in [4.78].

4.4 Syntactic Recognition of Syllables and Words

Syntactic recognition of syllables or words can be accomplished by starting from a phonemic or phonetic transcription of the unknown utterance or from a description of the acoustic pattern obtained by the spectral analysis of the utterance.

In the first case, the speech waveform or its representation must be segmented into phonemic units. This operation has been performed by studying the time evolution of some parameters like zero crossing counts and energies in some frequency bands or the spectral derivative defined as the average made over the entire frequency range of the time derivative of the spectral energy for each frequency of the spectrum. Some results of such approach are reported in [4.9, 62, 68]. Once segments corresponding to phonemes have been delimited, each segment can be assigned one or more phonemic transcriptions by comparing the average spectrum of the segment with a set of template spectra [4.62, 68]. The comparison is usually performed by cross correlation, and the phonemes for which high scores have been obtained are retained as candidates for the interpretation of the segments.

Improvements to such principles consist of allowing ambiguities on the segment bounds and of using phonological constraints, distinctive features and a priori phoneme confusion probabilities to correct or reorder the phonemic hypotheses.

Syllables and words are recognized by comparison of the noisy phonemic transcriptions obtained after phoneme recognition and the phonemic representations of a given syllable or word. A single word generally has many phonemic representations reflecting the phonological variations of the pronounciation of the word in different situations or by different talkers. A discussion on phonological rules can be found in [4.74].

Another possible approach is proposed in [4.61] and consists of extracting patterns from spectrograms and describing them by a picture language. The

descriptions of such patterns are concatenations of strings referring to different types of speech intervals. Such types are defined by the following rules.

$$
\begin{aligned}
&\langle\text{description}\rangle \rightarrow \alpha \\
&\langle\text{description}\rangle \rightarrow \langle\text{description}\rangle\ \alpha \\
&\qquad\qquad \alpha \rightarrow \text{SL} \\
&\qquad\qquad \alpha \rightarrow (\text{UT}) \\
&\qquad\qquad \alpha \rightarrow (\text{VCT})\,(\text{VCT})^* \\
&\qquad\quad \text{VCT} \rightarrow \text{FV} \\
&\qquad\quad \text{VCT} \rightarrow \text{NS} \\
&\qquad\quad \text{VCT} \rightarrow \beta \\
&\qquad\ \ \text{FV} \rightarrow (\text{F1T}) \\
&\qquad\ \ \text{NS} \rightarrow \text{F4T} \\
&\qquad\ \ \text{NS} \rightarrow \text{F5T} \\
&\qquad\ \ \beta \rightarrow \text{F2T} \\
&\qquad\ \ \beta \rightarrow \text{F3T} \\
&\qquad\ \ \text{UT} \rightarrow \text{FDES}
\end{aligned}
\tag{4.3}
$$

where

SL: silence
UT: unvoiced tract
VCT: voiced tract
FV: voiced non sonorant
NS: nasal
FDES: frication noise description ($1 \div 10$ kHz)
FiT: description of a portion of spectrogram having i formants.

The descriptions of intervals having formants are structured as follows:

$$
\begin{aligned}
&\text{F1T} \rightarrow (\text{F1D})\,(\text{A1D})\,(\text{FDES}) \\
&\text{F2T} \rightarrow (\text{F1F2D})\,(\text{A1A2D}) \\
&\text{F3T} \rightarrow (\text{F1F2D})\,(\text{A1A2D})\,(\text{F3D})\,(\text{A3D}) \\
&\text{F4T} \rightarrow (\text{F1F2D})\,(\text{A1A2D})\,(\text{F3F4D})\,(\text{A3A4D}) \\
&\text{F5T} \rightarrow (\text{F1F2D})\,(\text{A1A2D})\,(\text{F3D})\,(\text{A3D})\,(\text{F4F5D})\,(\text{A4A5D}),
\end{aligned}
\tag{4.4}
$$

where FhD is the description of the time evolution of the h-th formant frequency;
AhD is the description of the time evolution of the h-th formant amplitude;
FiFjD is the description of the evolution of F_i and F_j in the F_iF_j plane;

AiAjD is the description of the evolutions of the amplitudes A_i and A_j in the A_iA_j plane.

Formants can be extracted by algorithms proposed in [4.61, 70]. All the descriptions apply to monodimensional or bidimensional parametric graphs having time as a parameter.

This method represents stationary sounds by clusters of points performing a time normalization because the dimensions of the area containing the points are independent of the duration.

A relatively dense set of points in one of the planes is generated by the pronounciation of a vowel or a semivowel. When points lie within a surface of relatively small and fixed dimensions and their number is higher than an established threshold, a primitive, called stable zone (SZ), is assumed to be present in the graph. Such stable zones, that represent a quasi-stationary portion of the speech waveform, are detected by an algorithm described in [4.15].

A stable zone is described by the following message:

$$S, x_1, x_2, x_3, x_4$$

where S is the symbol used for denoting stable zone, x_1 is the time reference, x_2 is the duration, x_3 and x_4 are the coordinates of the center of gravity.

Nonstationary portions generally lead to lines of various shapes in the planar graph. These lines may be approximated by a succession of straight segments. An algorithm for detecting such segments is proposed in [4.15]. These lines are described by the following message:

$$L, x_1, x_2, x_5$$

where L is a slope symbol belonging to a set of 8 elements, x_1 and x_2 are the same as for the SZ and x_5 is the line length.

The description FDES is obtained by computing the gravity center of the spectrum energy in the frequency interval from 1 to 10 kHz and issuing a symbol according to its frequency. Two other symbols are then issued, corresponding to the initial and final frequency of the spectral interval where the energy is concentrated.

Finally three other symbols are included in the description according to the positions of the first three relative maxima belonging to three different classes. Symbols for the frication noise description are assigned according to Table 4.1.

Figure 4.7 shows the spectrum of the nonsonorant sound [$d\text{ʒ}$] and its description. A simple syntactic recognition scheme for words spoken in isolation has been proposed in [4.15]. The acoustic patterns are the parametric graphs described by two parameters related to the zero crossing densities of the outputs of two filters having approximately the bandwidth in the range of the first and the second formant, respectively.

These parameters, indicated as $B1(nT)$ and $B2(nT)$, are sampled every 20 ms. The pronounciation of a word gives a graph in the B1–B2 plane. The graphs have shapes similar to those that could be obtained in the F1–F2 plane after

Table 4.1. Symbols for coding describing the spectra of nonsonorant sounds

Frequency bounds (kHz)	Symbol
1 ÷2	FA
2 ÷2,5	FB
2,5÷3	FC
3 ÷3,5	FD
3,5÷4	FE
4 ÷4,5	FG
4,5÷5	FH
5 ÷6	FK
6 ÷7	FL
7 ÷8	FM
8 ÷10	FN

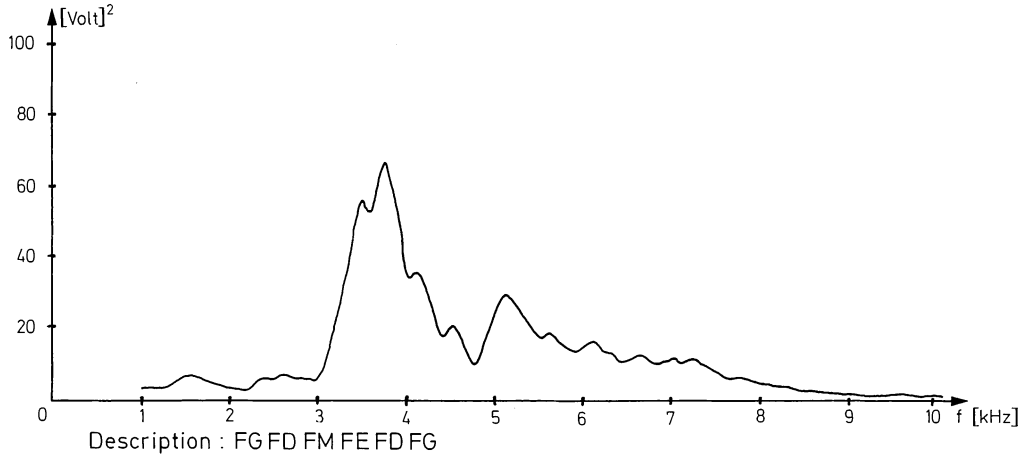


Fig. 4.7. Spectrum of the nonsonorant sound [dʒ] and its descriptions

extracting the first two formants. The advantage of using B1 and B2 is that these parameters can be computed in real time after a very simple preprocessing of the speech waveform. The simplicity has as counterpart a low accuracy in the distinction between words that are phonetically similar, making the use of these parameters suitable only for particular tasks.

The B1–B2 graphs are described in terms of lines (LN) and stable zones leading to a local aspect description. Then a global aspect description is obtained applying some composition rules defining the following global aspects forms:

$$\begin{aligned} &(\mathrm{IF}) \rightarrow \gamma(\mathrm{SZ}) \\ &(\mathrm{VF}) \rightarrow \gamma^2(\mathrm{SZ}) \\ &(\mathrm{ZF}) \rightarrow \gamma^3(\mathrm{SZ}) \\ &\gamma \rightarrow (\mathrm{LN}) \\ &\gamma \rightarrow (\mathrm{SZ}) \\ &\gamma \rightarrow (\mathrm{SZ})\,(\mathrm{LN})\,. \end{aligned} \tag{4.5}$$

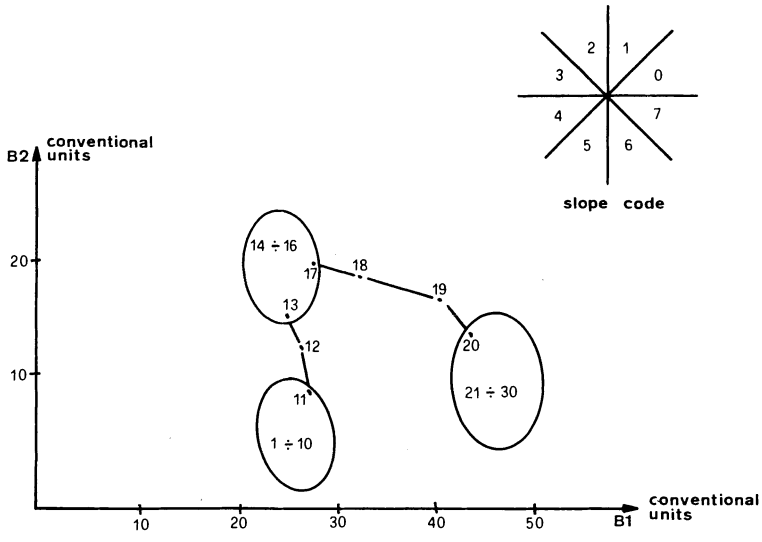


Fig. 4.8 Graph of the Italian digit UNO obtained after zero crossing measurements

The symbols of the global aspect descriptions have the following attributes:
— time reference of the portion of graph represented by the symbol,
— duration,
— composition code having a one for each SZ and a zero for each LN,
— the line lengths divided by 2^3,
— the line slopes represented by the code shown in Fig. 4.8,
— the coordinates of the centers of gravity of the last SZ.

Figure 4.8 shows, as an example, the graph of the Italian digit UNO (one). B1 is quantized on 60 levels and increases as the first formant increases; B2 is quantized on 30 levels and increases as the second formant increases. Stable zones are represented by rings in the graph and the numbers of the graph varying from 1 to 30 correspond to time intervals of 20 ms each. The local aspect description of the pattern of Fig. 4.8 is

S, 1. 11. 25. 2
S, 13. 5. 23. 19
*L*7, 17. 3. 17
S, 20. 11. 46. 8;

the global aspect description is

V. 1. 30. 1101. 22. 27. 46. 8.

For the fourth and fifth parameters of this description a line is considered to join the gravity centers of the first two SZ.

Recognition of each word is performed under the control of a simple grammar by which a description is translated into a word of the lexicon if it has proper symbols and its attributes satisfy some relations. A grammar for the ten digits spoken in Italian is given in [4.15].

4.5 Linguistic Interpretation of Speech Patterns

A general problem in speech recognition or speech understanding is that of emitting and verifying hypotheses about speech patterns that may correspond to syllables or words. In such cases it is important to consider possible ambiguities in feature extraction and classification. This means that a piece of spectrogram may admit more linguistic interpretations, and it is necessary to introduce a measure of correspondence between an interpretation and a piece of spectrogram. On this line a probability or a truth-value is assigned to the elements considered in the segmentation grammar and in feature extraction. Thus a speech interval α may be a silence (SL) with probability P (SL) or a sequence of unvoiced phonemes (UT) with probability P(UT) or a sequence of voiced tracts (VCT) (VCT)* with probability P((VCT) (VCT)*). Furthermore the probabilities must satisfy the following condition:

$$P(\alpha) = P(\mathrm{SL}) + P((\mathrm{UT})) + P((\mathrm{VCT})\,(\mathrm{VCT})^*)\,. \tag{4.6}$$

The probabilities are supposed to be proportional to the ratio between the value of a parameter typical of the feature corresponding to the symbol and the maximum value that can be assumed by such parameter. For each parameter a threshold is also fixed and if the parameter is not in a proper field, the corresponding probability is forced to zero.

The typical parameters used are the following:

for SL the log-amplitude fo the signal;

for UT the ratio between low and high frequencies energies;

for FV the normalized energy in the $5 \div 10$ kHz band;

for NS the sum of the Boolean values corresponding to the following events:

— downward shift of the first formant frequency;

— no dip in the supposed nasal interval on the first formant amplitude;

— presence of antiresonance dips in the nasal spectra;

— presence of formant clusters.

For the tracts with two or more formants an algorithm for probabilitistic formant tracking has been proposed in [4.18].

Admitting ambiguities implies that more patterns can be extracted from a piece of spectrogram Ω_l. Let d_{il} be the description of the i-th pattern extracted from Ω_l $(i = 1, 2, \ldots, \mathrm{N}_l)$. Algorithms for evaluating and verifying hypotheses will be presented in the following subsection.

4.5.1 Evaluation of Hypotheses

A system proposed in [4.17] is now described. In this system, the source of knowledge for the syllabic level of the recognition system is assumed to be a

grammar of speech generating a language recognized by a set of stochastic-finite-state automata (SFSA), with some auxiliary units that will be described later on. The use of stochastic automata is motivated by the following considerations. The formant patterns corresponding to a given vowel-consonant-vowel (VCV) utterance pronounced in various contexts cannot be random patterns; they must respect some rules underlying the relations between the articulatory commands and the spectra of the generated waveforms. On the other hand, the patterns are not equal because some distorsions on the archetypes do not alter the interpretation given by a listener to the original utterance and it is important to know the statistics of such distorsions. Thus a SFSA is a proper tool for representing the possible patterns of a given utterance. Implementation and learning are also feasible because the segments considered are short in time and their possible patterns are described by a concatenation of few symbols belonging to a very small vocabulary.

The case presented here for pseudo-syllable segments could be extended to other units (words, for example) depending on the purpose of the recognition system. In every case the patterns corresponding to the units are assumed to be described by a language. Once the descriptions are obtained, they are processed under the control of a grammar of speech and are translated into possible phonemic transcriptions, with associated the conditional probability $P(S_j|\Omega_l)$ that the unit, in our case the syllable, S_j corresponds to Ω_l, the l-th segment of the spectrogram of the spoken sentence.

The grammar of speech is a stochastic grammar representing the possible patterns for each coarticulation instance, corresponding to the bounds generated by the segmentation algorithm.

The terminal alphabet of the grammar of speech contains all the symbols with which the descriptions are made; the nonterminal alphabet contains all the possible concatenations of phonemes for which coarticulation affects the formant patterns even after the description approximations. The nonterminal alphabet contains also the symbols emitted by an auxiliary unit preceding each SFSA, and acting as translator of the descriptions made of symbols, attributes and probabilities into symbols and probabilities, provided that some relations hold between the attributes. Hypothesis evaluation is performed with a procedure whose block diagram is shown in Fig. 4.9.

When a syllable or a coarticulation segment is hypothesized on a certain portion of the spectrogram, the syllable description is processed by the description translator (DT). The DT translates a symbol with its attributes into a symbol and its associated probability. The output symbol is emitted only if some conditions on the attributes of the input description are verified. These conditions are stored for a talker and a typical lexical position of the segment and can be modified by some speaker-dependent modifiers, acting mainly on the formant loci for the stable-zone constraints. In addition, some lexical-dependent modifiers, acting mainly on the durations, change the constraints in accordance with the stress of the segment. The design of such modifiers that also involves the knowledge of probability distributions can be limited to spectral loci and durations. The input to the DT is the description d_{il}. The output of the DT is a string g_{il} of symbols without attributes obtained from d_{il} which is a string

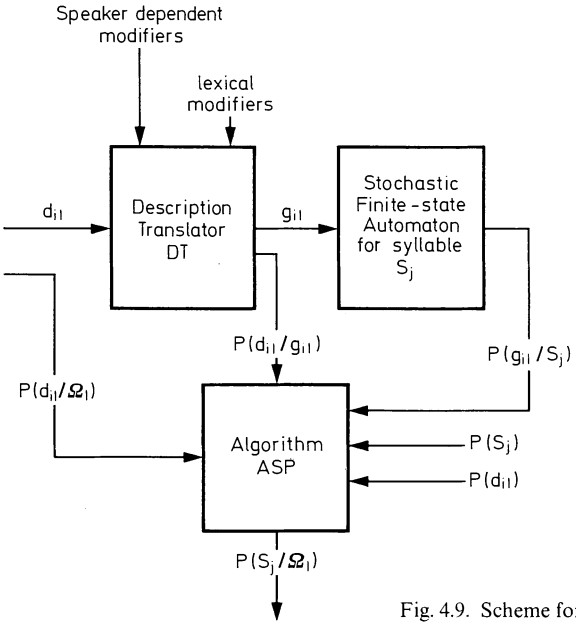


Fig. 4.9. Scheme for the evaluation of syllabic hypotheses

of symbols with attributes; together with g_{il}, the probability $P(d_{il}|g_{il})$ is obtained. Finally, the g_{il} is processed by the SFSA and, if the g_{il} is recognized, the probability $P(g_{il}|S_j)$ is given. All the obtained probabilities and the probability $P(d_{il}|\Omega_l)$ that the description d_{il} refers to features present in the spectrogram Ω_l are processed by an algorithm indicated in Fig. 4.9 as *ASP* that gives $P(S_j|\Omega_l)$. The details of this algorithm are given in [4.17].

In the case common in practice where only one description d_{il} is recognized and translated by the DT into a g_{il} that is recognized by the SFSA of the syllable S_j, the algorithm gives the following evaluation for the hypothesis S_j on the piece of spectrogram Ω_l:

$$P(S_j|\Omega_l) = P(S_j)\,\frac{P(d_{il}|\Omega_l)}{P(d_{il})}\,P(d_{il}|g_{il})\,P(g_{il}|S_j) \tag{4.7}$$

where the three conditional probabilities are given, respectively, at the outputs of the descriptor, the DT and the SFSA.

Let the description d_{il} be recognized by the automata of the syllables $S_1, S_2, \ldots, S_w$. The probability $P(d_{il})$ can be computed as follows:

$$P(d_{il}) = P(d_{il}|S_1)\,P(S_1) + P(d_{il}|S_2)\,P(S_2) + \cdots + P(d_{il}|S_w)\,P(S_w) \tag{4.8}$$

because $S_1, S_2, \ldots, S_j, \ldots, S_w$ are disjoint sets.

Each addend in (4.8) is obtained from the a priori knowledge of the system, as follows:

$$P(d_{il}|S_j)\,P(S_j) = P(S_j|d_{il})\,P\,(d_{il}) = P(d_{il}|g_{ij})\,P(g_{ij}|S_j)\,P(S_j)\,. \tag{4.9}$$

The probabilities in (4.9) are the a priori information that needs to be learned. There are several possible approximations for $P(d_{il}|g_{ij})$ that correspond to different reductions in the computation of (4.7) and (4.8).

The simplest approximation consists of considering $P(d_{il}|g_{ij})$ independent from j. A better approximation, that seems to be very realistic, consists of considering the probability of an attribute of d_{il} uniform over the range allowed for this attribute, in order to translate d_{il} into g_{ij}. In this case, one need learn only the intervals allowed for the attributes of d_{il}. Finally, the exact distributions of the attributes of d_{il} over the allowed intervals of the DT could be learned and used to compute $P(d_{il}|g_{ij})$. This last approach would require a very large number of experiments, probably without greatly improving the system performance.

4.5.2 Preselection Rules

In practice, it is time consuming trying to find whether the description of a PSS is recognized by any among all the SFSA of the grammar of speech. For this reason, the description is preprocessed in a preselection stage.

Preselection is performed analyzing first the gravity centers of the stable zones in the F1-F2 description belonging to a time interval previously labelled as vowel. Both the absolute values and the relative positions of such zones are considered. For VCV, VCCV, VCVC, VCCVC (V is for vowel, C is for consonant) pseudo-syllable segments, the vector joining the gravity center of the first vowel with the gravity center of the second vowel is used to restrict the set of candidates.

For example, if the first vowel has the second formant lower than the second formant of the second vowel, hypotheses having pairs of vowel like IU, IO, EU, EO, AU are not considered. A vector is translated into a symbol depending on the vector slope, its starting point and its length. A table gives, for each vector symbol, the possible candidates.

A preselection, for what concerns the consonants, is based on the context-independent distinctive features previously extracted and on the analysis of the portions of the F1-F2 plane where the lines joining the two vowel stable zones lie.

4.5.3 Verification of Hypotheses

The verification of a syllabic hypothesis may be requested by the control unit of the recognition system. Let S_j be the syllable to be verified. Generally the research of S_j is restricted to a given time interval t_e. Let D_e be the set of all the possible descriptions that can be extracted from the acoustic pattern Ω_e corresponding to the time interval t_e; a member of D_e can also be the description

of an acoustic pattern that is a piece of Ω_e. It is possible that a member of D_e is recognized as S_j in the verification step even if the hypothesis S_j was not emitted for some segment of Ω_e in the hypothesis emission stage. This can be due to errors in segmentation even after segmentation refinement.

In such a case the answer to a request for verifications is the following probability:

$$P_j = \operatorname*{Max}_{d_{be} \in D_e} p(S_j|\Omega_b) \tag{4.10}$$

where $\Omega_b \subseteqq \Omega_e$ is the acoustic pattern described by d_{be}.

If none of the members of D_e is recognized as S_j, a loss function $q(S_j, \Omega_e)$ is emitted as answer to the request of verification. The computation of $q(S_j, \Omega_e)$ can be viewed in the framework of the syntactic classification of noisy patterns [4.20].

Let D_{le} be the set of the extracted descriptions of the entire pattern Ω_e, then

$$q(S_j, \Omega_e) = \operatorname*{Inf}_{d \in D_{le}} \frac{q(d|S_j)}{p(d|\Omega_e)}. \tag{4.11}$$

The function $q(d|S_j)$ is the loss function representing the cost of translating the description d into a string g accepted by the SFSA of S_j. Let $L(S_j)$ be the language of all the strings accepted by the automaton. The loss function is defined in accordance with FUNG and FU [4.20] as follows:

$$q(d|S_j) = \operatorname*{Inf}_{g \in L(S_j)} \frac{q(d|g)}{p(g)}. \tag{4.12}$$

The function $q(d|g)$ represents the distance between the description d and the string g. This distance is assumed to be zero if the description d is translated into g by the description translator of the syllable S_j. The function $q(d|S_j)$ can be efficiently computed with the following considerations. For each element of d it is possible to establish a distance from each symbol of the terminal alphabet of $L(S_j)$. These distances are organized in a matrix and are computed with an algorithm that uses syllable-independent rules that are summarized below.

Rule 1. The distance between an element of the description and a symbol is zero if the element is translated into the symbol by the translator of S_j.

Rule 2. The distance between a symbol and an element of the description that cannot be translated into the symbol itself only because the element attributes do not lie into a specific interval is proportional to a linear combination of the smallest modifications that should be introduced on the attributes in order to make the description element translatable into the symbol. This rule applies, for example, to all the cases where the description element is a stable zone and the symbol is the translation of a stable zone.

Rule 3. The distance between two lines is proportional to a linear combination of the minimum differences between the member's slope and length and the allowed intervals for the symbol.

Rule 4. The distance between a line and a stable zone is a fixed constant.

The insertion of elements into d corresponding to some symbols is controlled by similar rules that are omitted for sake of brevity. At the beginning of the computation of the (4.12), the description d is "augmented" according to the insertion rules by elements that can be translated by the translator.

Examples of insertion rules are: a line of slope x can be interrupted by one or more stable zones; one or two lines, depending on the particular situations, can be inserted between two stable zones; a stable zone can be inserted between two lines, etc. The distance between an inserted element and the corresponding symbol is also defined by the insertion rules.

The augmented description is processed by the SFSA of S_j with an algorithm that allows only operations of deletion of members of the description or substitution of these members with symbols of the terminal alphabet of $L(S_j)$. The loss associated with the deletion of a member inserted only into the augmented description is zero, while the loss associated with the deletion of an element of the original description is given by some deletion rules. They establish, for example, that loss of a short stable zone between two lines is small.

Further details of the algorithm can be found in [4.18].

4.6 Automatic Learning of Speech Patterns

In learning structural descriptions of patterns there are two possible tasks. The first consists of learning the structures of a pattern class from a finite set of positive and negative information samples. In this way, one can infer a grammar that is capable of generating an infinite set of patterns having a structure that is assumed to have been learned by the presentation of a sufficient number of samples. The second possible task is that of building up a finite-state automaton that recognizes only the learned patterns.

The problem, in learning acoustic patterns, is that of representing by a limited set of rules all the possible acoustic realizations of an utterance that appear different with a given method of representation. These realizations are certainly finite in number, especially after they have been translated by the DT into strings of symbols.

Furthermore, different strings correspond to different possible articulations of the same utterance and these different possibilities are probably speaker- and context-dependent. Thus this plurality is adequately represented by a finite-state automaton. Furthermore, the transitions from state to state of this automaton have different probabilities and these probabilities may depend on both the speaker and the context.

From the generative point of view, the stochastic-finite-state-automaton representing a PSS generates a finite set of strings, each one of which can generate descriptions with different attributes leading to a very large number of acoustic realizations.

A SFSA is represented in memory as a deterministic acceptor with an associated data structure of transition probabilities. This data structure allows one to consider a single state of the deterministic acceptor as broken into many states of the stochastic automaton. These states are equivalent for what concerns

the symbols associated with the transitions, but they are not equivalent if the transition probabilities are also considered. The deterministic acceptor is used to test rapidly whether more than one syllable can correspond to a given description. Only in this case, one must compute the probabilities associated with each syllable that is a possible transcription of the description. When speaker or context dependency is particularly relevant, a table of modifiers can be introduced for the parameters of the translators or in the data structure of the SFSA.

The feature extraction procedure of the learning is performed through the following steps. The spoken sentence is segmented into PSSs. The segmentation can be corrected or refined by an editing routine operated by a human supervisor. For each segment, spectrograms are computed and for the voiced portions, formants are tracked. When several formant patterns are possible, the formant tracking algorithm selects the most probable, but all the candidates are displayed on a video and the supervisor can alter the selection. Finally, the selected formant pattern is described leading to a string d that can be corrected by the supervisor.

Segment bounds refinement including fusion of two PSSs or further subdivision of a PSS is performed according to the following rules:

$$
\begin{aligned}
&(\mathrm{PSS})\,(\mathrm{SZ}) \rightarrow \lambda(\mathrm{SV})\,(\mathrm{SZ})\\
&(\mathrm{PSS})\,(\mathrm{LN}) \rightarrow \lambda(\mathrm{SV})\,(\mathrm{LN})\\
&(\mathrm{PSS})\,(\mathrm{UN}) \rightarrow \lambda(\mathrm{SV})\,\xi_6(\mathrm{UN})\\
&\mathrm{UN} \rightarrow \mathrm{SL}\\
&\mathrm{UN} \rightarrow \mathrm{UT}\\
&\mathrm{UN} \rightarrow (\mathrm{SL})\,(\mathrm{UT})\\
&\mathrm{UN} \rightarrow (\mathrm{UT})\,(\mathrm{SL})\\
&\mathrm{UN} \rightarrow (\mathrm{UT})\,(\mathrm{SL})\,(\mathrm{UT})\\
&\quad \lambda \rightarrow (\mathrm{UN})\,\xi_6\\
&\quad \lambda \rightarrow \xi_6\\
&\mathrm{SV} \rightarrow (\mathrm{SA})\\
&\mathrm{SV} \rightarrow \xi_7\, y(\mathrm{SA})\\
&\mathrm{SA} \rightarrow \mathrm{SZ}\ \{(\text{duration} \geqq \vartheta) \wedge (\text{gravity centers in a vowel area})\};\\
&\xi_6 \rightarrow (\mathrm{LN})\\
&\xi_6 \rightarrow (\mathrm{LN})\,\xi_6\\
&\xi_7 \rightarrow yS\\
&\xi_7 \rightarrow yS\xi_7\,.
\end{aligned}
\tag{4.13}
$$

y is a line having a specific slope code;

S is an SZ that cannot be rewritten as SA:

ϑ is a duration threshold.

Finite state automata of each pseudo-syllable segment are represented in memory with a right linear grammar.

Let $G^j(n)$ be the grammar of the j-th PSS after the presentation of n samples of the j-th PSS. Let $I^j(n+1)$ be the string describing the $(n+1)$-th sample of the j-th PSS. If $I^j(n+1)$ is recognized by the automaton that accepts $L(G^j(n))$, the productions of the grammar are not altered and only the statistics associated with each production are updated.

If $I^j(n+1)$ is not in the language $L(G^j(n))$, then a new grammar is generated by the following algorithm AL1 proposed in [4.71].

Algorithm AL1

Step 1. The regular expression $R^j(n)$ is obtained from the automaton that recognizes $L(G^j(n))$ with an algorithm described by Booth [4.72].

Step 2. The regular expression

$$R^j(n+1)=R^j(n)+I^j(n+1) \tag{4.14}$$

is considered.

Step 3. The set

$$F[R^j(n+1)]=\{x_i/x_i \text{ is a symbol of } R_j(n+1) \text{ that is not preceeded by other symbols}\}, \text{ is derived}.$$

Step 4. Equation (4.14) is rewritten as follows:

$$R^j(n+1)=x_1D_{x_1}[R^j(n+1)]\ldots+x_iD_{x_i}[R^j(n+1)]\ldots + x_{F1}D_{x_{FN}}[R^j(n+1)] \tag{4.15}$$

where $D_{x_i}[R^j(n+1)]$ is the derivative of the regular expression $R^j(n+1)$ with respect to x_i and FN is the number of elements of the set $F[R^j(n+1)]$.

Step 5. Steps 3 and 4 are repeated for all the derivatives in (4.15) and for the derivatives of the derivatives and so on, until no more sets are found. At this point a deterministic finite state automaton can be found in a straightforward way from $R^j(n+1)$. Notice that steps 3, 4 and 5 of AL1 realize all the possible left factorizations on $R^j(n+1)$.

The automaton does not generally have the minimum number of states. State minimization is carried out with a simplified version of the general theory because the automaton recognizes only strings of finite length. Thus each state can be assigned a distance from the final state equal to the maximum length of the string that causes a move from this state to the last final state.

Step 6. Detection of equivalent states is carried out easily considering states having the same distance from the final state, starting from those having distance one and continuing with sets of states having increasing distance.

Step 7. A right linear grammar is obtained from the reduced automaton using the straightforward procedure that can be found in a book by AHO and ULLMAN [4.73].

4.7 Conclusions

The main problems encountered in speech recognition, namely speech analysis and preprocessing, feature extraction, emission and verification of hypotheses and automatic learning of spectral features, have been presented and directions for solutions with syntactic methods have been proposed.

Ambiguities are treated introducing probabilistic logic; the difficulties incurred in learning probability distributions can be avoided using fuzzy logic and allowing a subjective assignment of membership functions that could not be inferred with a satisfactory accuracy by experiments.

Remarkable simplifications on the general approach described in this chapter may be obtained if the task is limited to a small protocol or a small lexicon. In this case the grammars controlling the recognition process can tolerate much degradation in the input information making rules learned in laboratory conditions applicable to real-life situations.

References

4.1 S.R.HYDE in *Human Communication: a Unified View*, ed. by E.E.DAVID, P.B.DANES (McGraw-Hill, 1972), pp. 399–438
4.2 J.R.PIERCE: J. Acoust. Soc. Am. **46**, 1049 (1969)
4.3 A.NEWELL, J.BARNETT, C.GREEN, D.KLATT, J.C.R.LIKLIDER, J.MUNSON, R.REDDY, W.WOODS: *Speech Understanding Systems: Final Report of a Study Group* (North Holland 1973)
4.4 A.M.LIBERMAN: Cognitive Psychology **1**, 301 (1970)
4.5 M.STUDDERT-KENNEDY: Speech Perception: in *Contemporary Issues in Experimental Phonetics*, ed. by N.J.LASS (C. C. Thomas, Springfield, Il. 1976)
4.6 J.L.FLANAGAN: *Speech Analysis, Synthesis and Perception*, 2nd ed. (Springer Berlin, Heidelberg, New York 1972)
4.7 S.E.G.OHMAN: J. Acoust. Soc. **39**, 151 (1966)
4.8 B.E.F.LINDBLOM, M.STUDDERT-KENNEDY: J. Acoust. Soc. Am. **40**, 123 (1966)
4.9 D.R.REDDY: J. Acoust. Soc. Am. **34**, 1 (1966)
4.10 P.MERMELSTEIN: IEEE Trans. **ASSP-23**, 79 (1975)
4.11 R.DEMORI: Acustica **34**, 158 (1976)
4.12 A.KUREMATSU, M.TAKEDA, S.INOUE: Proc. 2nd Intern. Joint Conf. Artificial Intelligence. London (1971) p. 248
4.13 R.NEWMAN, K.S.FU, K.P.LI: Proc. Conf. Speech Commun. Process, Newton, MA (1972), p. 121
4.14 W.A.LEA: IEEE Trans. **AU-21**, 249 (1973)
4.15 R.DEMORI: IEEE Trans. **AU-21**, 89 (1973)
4.16 R.DEMORI: in *Information Processing 74*, ed. by J.ROSENFELD (North-Holland Publishing Co. 1974), pp. 753÷757
4.17 R.DEMORI, S.RIVOIRA, A.SERRA: Proc. 4th Intern. Joint Conf. Artificial Intelligence, Tbilisi, **2**, 468 (1975)
4.18 R.DEMORI, P.LAFACE, E.MICHELIS, E.PICCOLO: Proc. IEEE Conf. Cybernetics and Society, San Francisco, CA, **1**, 284 (1975)
4.19 K.S.FU: *Syntactic Methods in Pattern Recognition* (Academic Press, London, New York 1974)

4.20 L.W.FUNG, K.S.FU: IEEE Trans. **C-24**, 662 (1975)
4.21 D.R.REDDY (ed.): *Speech Recognition: Invited Papers presented at the* IEEE *Symposium* (Academic Press, New York, 1975)
4.22 L.D.ERMAN (ed.): IEEE Symp. Speech Recognition, Contributed Papers, April 15–19, 1974, Carnegie-Mellon Univ., Pittsburgh, PA, IEEE Catalog No. 74CH0878-9 AE (1974)
4.23 M.DERKACH: Speech Transmission Laboratory Quarterly Prog. Rep. 1/70, Royal Institute of Technology, Stockholm (1970), p. 39
4.24 M.DERKACH, R.GUMETSKY, L.MISHIN: Proc. 1972 Conf. Speech Communication and Processing, April 24–26, 1972, Newton, MA (1972) p. 338
4.25 M.DERKACH, R.GUMETSKY, B.GURIN, L.MISHIN: Proc. Speech Communication Seminar, Aug. 1–3, 1974, Stockholm, (Almqvist and Wiksell, Uppsala, 1976)
4.26 I.T.TURBOVICH: in *Recevoi ovshemie v automatisirovannik sistemak.* Isdatelstvo Nauka ed. – Moskow (1975) (In Russian)
4.27 D.H.KLATT, K.N.STEVENS: IEEE Trans. **AU-21**, 210 (1973)
4.28 D.R.REDDY: J. Acoust. Soc. Am. **41**, 1295 (1967)
4.29 D.R.REDDY: J. Acoust. Soc. Am. **42**, 329 (1967)
4.30 N.G.ZAGORUIKO: Speech Transmission Laboratory Quarterly Prog. Rept. 1/70, Royal Institute of Technology, Stockholm, 32 (1970)
4.31 M.BATES: IEEE Trans. **ASSP-23**, 112 (1975)
4.32 P.ROVNER, B.NASH-WEBBER, W.A.WOODS: IEEE Trans. **ASSP-23**, 136 (1975)
4.33 W.A.WOODS: IEEE Trans. **ASSP-23**, 2 (1975)
4.34 W.A.WOODS, J.MAKHOUL: Artificial Intelligence **5**, 73 (1974)
4.35 R.SCHWARTZ, J.MAKHOUL: IEEE Trans. **ASSP-23**, 50 (1975)
4.36 B.NASH-WEBBER: IEEE Trans. **ASSP-23**, 124 (1975)
4.37 D.R.REDDY, L.D.ERMAN, R.B.NEELY: IEEE Trans. **AU-21**, 229 (1973)
4.38 V.R.LESSER, R.D.FENNELL, L.D.ERMAN, D.R.REDDY: IEEE Trans. **ASSP-23**, 11 (1975)
4.39 D.E.WALKER: IEEE Trans. **ASSP-23**, 397 (1975)
4.40 J.-P.HATON: IEEE Trans. **ASSP-22**, 416 (1974)
4.41 W.A.LEA, M.F.MEDRESS, T.E.SKINNER: IEEE Trans. **ASSP-23**, 30 (1975)
4.42 F.JELINEK, L.R.BAHL, R.L.MERCER: IEEE Trans. **IT-21**, 250 (1975)
4.43 J.K.BAKER: IEEE Trans. **ASSP-23**, 24 (1975)
4.44 W.A.WOODS, M.BATES, B.BRUCE, J.COLARUSSO, C.COOK, L.GOULD, D.GRABEL, J.MAKHOUL, B.NASH-WEBBER, R.SCHWARTZ, J.WOLF: Tech. Rept. 2976, Bolt Beranek and Newman Inc., Cambridge, MA (1974)
4.45 R.W.SHAFER, L.R.RABINER: Proc. IEEE **63**, 662 (1975)
4.46 B.GARDINI, V.MAURO, A.SERRA: Informatica 74, Bled, Yugoslavia, (1974) Memo 6.6
4.47 F.ITAKURA, S.SAITO: Electron. and Comm. in Japan **53-A**, 36 (1970)
4.48 B.S.ATAL, S.L.HANAUER: J. Acoust. Soc. Am. **50**, 637 (1971)
4.49 J.D.MARKEL: Monograph 7, Speech Comm. Res. Leb., Santa Barbara, CA (1971)
4.50 J.ALLEN: Proc. IEEE **63**, 624 (1975)
4.51 L.R.RABINER, B.GOLD: *Theory and Applications of Digital Signal Processing* (Prentice Hall, Englewood Cliffs, NJ, 1975)
4.52 R.DEMORI, S.RIVOIRA, A.SERRA: IEEE Trans. **C-24**, 1022 (1975)
4.53 V.M.VELICHIKO, N.G.ZAGORUIKO: Intern. J. Man-Machine Studies **2**, 223 (1970)
4.54 G.M.WHITE: Proc. IEEE Conf. Cybernetics and Society, San Francisco, CA, **1**, 290 (1975)
4.55 F.ITAKURA: IEEE Trans. **ASSP-23**, 67 (1975)
4.56 L.C.W.POLS: IEEE Trans. **C-20**, 1972 (1971)
4.57 L.F.PAU: in *Machine Perception of Patterns and Pictures*, ed. by The Institute of Physics, London Bristol (1972), p. 126
4.58 R.B.NEELY, G.M.WHITE: in *Information Processing 74*, ed. by J.ROSENFELD North-Holland, Amsterdam, New York 1974), pp. 748–752
4.59 D.H.KLATT: Proc. Speech Communication Seminar, Aug. 1–3, 1974, Stockholm (Almqvist and Wiksell, Uppsala, 1976)
4.60 G.W.HUGUES, J.F.HEMDAL: Tech. Rep. AFCRL-65-681 (P 13552), Purdue University (1965)
4.61 R.DEMORI, P.LAFACE, E.PICCOLO: IEEE Trans. **ASSP-24**, (Oct. 1976)
4.62 C.J.WEINSTEIN, S.S.MCCANDLESS, L.F.MONDSHEIN, V.ZUE: IEEE Trans. **ASSP-23**, 54 (1975)
4.63 J.MAKHOUL, J.J.WOLF: Tech. Rept. 2304, Bolt Beranek and Newman, Cambridge, MA (1972)

4.64 K. N. STEVENS: MIT RLE Quarterly Progress Report **110**, 155 (1973)
4.65 D. G. BOBROW, D. H. KLATT: Proc. AFIPS Joint Computer Conf Thompson-Washington D. C., **33**, 305 (1968)
4.66 S. ITAHASHI, S. MAKINO, K. KIDO: IEEE Trans. **AU-21**, 239 (1973)
4.67 T. B. MARTIN: Acoustic Recognition of a Limited Vocabulary in Continuous Speech, PhD dissertation, Univ. of Pennsylvania, Philadelphia, PA (1970)
4.68 C. C. TAPPERT, N. R. DIXON, A. S. RABINOWITZ: IEEE Trans. **AU-21**, 225 (1973)
4.69 R. DEMORI: The Radio and Electronic Engineer **46**, 173 (1976)
4.70 S. S. MCCANDLESS; IEEE Trans. **ASSP-22**, 132 (1974)
4.71 R. DEMORI, S. RIVOIRA, A. SERRA: Automatic Learning of Spectral Features Extracted from Continuous Speech, Proc. 3rd Intern. Congress Cybernetics and Systems, Bucharest, Romania (in press)
4.72 T. L. BOOTH: *Sequential Machines and Automata Theory* (John Wiley, New York 1967)
4.73 A. V. AHO, J. D. ULLMAN: *The Theory of Parsing, Translation and Compiling* (Prentice-Hall, 1973)
4.74 B. T. OSHIKA, V. W. ZUE, R. V. WEEKS, H. NEU, J. AURBACH: IEEE Trans. **ASSP-23**, 104 (1975)
4.75 R. DEMORI, P. LAFACE: Proc. Third Intern. Joint Conf. Pattern Recognition, Coronado, CA, (1976), p. 609
4.76 J. J. WOLF: in *Digital pattern recognition*, ed. by K. S. Fu (Springer Berlin, Heidelberg, New York 1976)
4.77 J. D. MARKEL, A. H. GRAY: *Linear prediction of speech,* (Springer Berlin, Heidelberg, New York 1976)
4.78 R. DEMORI, P. LAFACE, P. TORASSO: Automatic recognition of liquids and nasals in continuous speech, Int. Rept. CENS Laboratory, Turin Italy (1976)

5. Chinese Character Recognition

W. W. STALLINGS

With 17 Figures

An approach to Chinese character recognition, based on a formal model of the pictorial structure of Chinese characters, is described. A program has been written which produces a description of a character on two levels: i) the internal structure of each connected part of the character, and ii) the arrangement in two dimensions of the connected parts. A numeric code is generated from the structural representation of a character, and is used for recognition.

In addition, a survey is made of recent formal models of Chinese characters. All of these models develop a two-dimensional grammar of Chinese characters. The application of these grammars to a number of machine input-output functions for Chinese characters is discussed.

5.1 Historical

5.1.1 An Approach to Pattern Recognition

This chapter reports on a study of the syntactic or structural approach [5.1, 2] to the recognition of Chinese characters. Using this approach, a scheme for automatic pattern recognition has been developed which includes the following tasks:

i) *Description.* A systematic scheme for the description of the pictorial structure of the patterns to be recognized is developed.

ii) *Analysis.* An algorithm is designed which analyzes the structure of the patterns, producing a representation of the structure conforming to the descriptive scheme.

iii) *Encoding.* From the structural representation of a pattern, a code is generated which uniquely identifies the pattern.

This method has been applied to the recognition of Chinese characters. A program has been written that analyzes Chinese characters; it produces a data structure which describes a character in terms of basic picture elements and the relationship among them. A procedure has been developed for generating a numeric code from the structural representation. Recognition is achieved by building up a dictionary matching characters with their codes; the code for any new instance of a character can then be looked up in the dictionary [5.3–5].

5.1.2 A Chinese Reading Machine

Two obstacles have hindered the access of interested non-Chinese groups to the vast body of written Chinese produced each year. The first is the difficulty of the

language itself. Chinese is very complex and takes so long to master that few Westerners ever learn it well. And second, of course, is the size of the printed output in Chinese. Manual translation is slow and tedious, and can never be relied on to handle more than a tiny fraction of the material.

To make available to Westerners the culture and technology of one-quarter of the human race, some form of automation must be introduced. A Chinese reading machine, which could scan printed Chinese and produce English output, would provide the most desirable means of improvement. Such a machine is a long way down the road, but individual steps which advance toward that goal are to be encouraged.

Considerable work has been done in the area of automatic translation of Chinese [5.6–9] but these efforts have been only partially successful. Even if a good translation device were available, however, the formidable problem of encoding Chinese characters for input would remain.

One answer to the problem would be the development of a practical Chinese character recognition machine, toward which the effort of this project is directed. It is hoped that advances in this area would provide additional incentive for work in translation devices. On a more modest scale, a Chinese character recognition device could be used as a type of on-line dictionary to speed up the process of human translation. Even this limited application would be a welcome advance.

5.1.3 Chinese Characters

Chinese characters, used to encode all the dialects spoken in China as well as the historically unrelated Japanese language, present an interesting collection of patterns for study. Written Chinese is a pictorial and symbolic system which differs markedly from written Western language systems [5.10]. Chinese characters are not alphabetic; they are of uniform dimension, generally square, and are composed of strokes, each one a line that can be drawn without lifting the pen. Chinese characters possess a great deal of structure. Many regularities of stroke configuration occur. Quite frequently, a character is simply a two-dimensional arrangement of two or more simpler characters. Nevertheless, the system is rich; strokes and collections of strokes are combined in many different ways to produce thousands of different character patterns. Figure 5.1 is a small sample.

The structure of a Chinese character may be specified hierarchically. Strokes are considered to be the basic picture elements. Nearly all strokes consist of one or a small number of straight-line segments which appear as either horizontal, vertical, or in a direction along one of the main diagonals. Strokes are combined to form units, called components in this chapter, which occur in many different characters. Finally, each character consists of a two-dimensional arrangement of one or more components.

The written Chinese language is very difficult to learn. There is the sheer size of the language: there are over 40000 Chinese characters of which an educated person would be expected to know about five to ten thousand. Each character corresponds roughly to a word in Western languages and the meaning of each character must be learned by rote. Each character has a fixed monosyllabic pronunciation which again must be learned by rote. Usually, one component of a

character gives a clue to its meaning and the rest gives a clue to its pronunciation, thus easing the above two tasks somewhat. Another difficulty is dictionary lookup. Since there is no alphabetic order to Chinese characters, a number of special systems have been devised to impose an ordering, none of which is terribly

支：支
攴：收改攻放政故效教敏救敗敍敝敢
散敦敬敲敵敷數整斃
文：文
斗：斗料斜
斤：斤斥斯新斷
方：方於施旁旅族旋旗
旡：既
日：日旦旨早旱旬昂昆昌昏明易昔星
映春昨是時晉晚晤晨普景晴晶智
暑暖暗暢暴暫暮曆曁曉昭

Fig. 5.1. Some Chinese characters

convenient. Finally, a student of Chinese is taught to draw the strokes of each character in a particular order and this too must be learned; a character may have from one to thirty strokes with eight to twelve being typical. There are some general guidelines but these do not absolutely apply in all cases: left-to-right, top-to-bottom, long horizontal strokes before vertical ones, and so on.

5.1.4 Preview

In Section 5.2, we survey recent attempts to develop a formal description of the pictorial structure of Chinese characters, and briefly mention their application in areas other than pattern recognition. Section 5.3 discusses a printed Chinese character recognition scheme based on one such formal description, and briefly discusses other approaches to Chinese character recognition.

5.2 The Morphology of Chinese Characters

5.2.1 Models

In recent years, a considerable amount of work has been done on the formal description of Chinese characters [5.11]. The job is nontrivial because of the already discussed complexity of the character system. The work is important for two reasons:

i) A complete formal description will yield a complete classification system, which could aid in dictionary lookup;
ii) A sound knowledge of the structure of Chinese characters contributes to their mechanization, as discussed in Subsection 5.2.2, and to their recognition, as discussed in Section 5.3.

Rankin et al.

RANKIN and his associates [5.12–16] have developed a two-level grammar for Chinese characters:

- a "generative" grammar which generates characters by means of a list of components and a set of rules for component combination. Components are defined to be "often-recurring character subparts".
- a "decomposition" grammar which explicates the process involved in the formation of components from strokes.

The arrangement of components in two dimensions to form characters is described using the concept of frame. Each character is viewed as occupying a hypothetical square. The segmentation of a character into components segments its square accordingly. The square, or frame □, may be segmented in one of three ways: a) East-West ◫, b) North-South ⊟, c) Border-Interior ⧈. Each of these segmentations corresponds to a two-component character. For example 化 would be represented by a), which decomposes the character into 亻 and 匕. 台 would be represented by b). Finally, either partial or complete enclosure, such

CHAR → COMP | IT
COMP → v(N,S) | h(E,W) | s(B,CHAR)
N → COMP | NT
S → COMP | ST
E → COMP | ET
W → COMP | WT
B → BT

Notes:
1. The grammar may be read as follows:
   - ○ a Character is a Composite or an Interior Terminal
   - ○ a Composite is the vertical combination of a North and South, a horizontal combination of an East and West, or a surround combination of a Border and a Character
   - ○ A North is a Composite or a North Terminal, and so on.
2. The terminal elements of the grammar are components. Each component may appear in one or more positions and hence appears on one or more lists of terminal elements.
3. The border or outside subframe may not be subdivided.
4. A character may consist of a single component.

Fig. 5.2. A grammar for component combination

as 句 and 困 would be represented by c). Frames for characters composed of more than two components are obtained by embedding a), b), or c) in one of the subframes of a), b), or c). The process of embedding is recursive, in that any subframe of a derived frame may be used for further embedding. The process of frame embedding is specified concisely by the grammar of Fig. 5.2. For example, the four-component character of Fig. 5.3 can be described by the frame arrangement in Fig. 5:3a. The frame description can be conveniently represented by a tree as indicated in Fig. 5.3b.

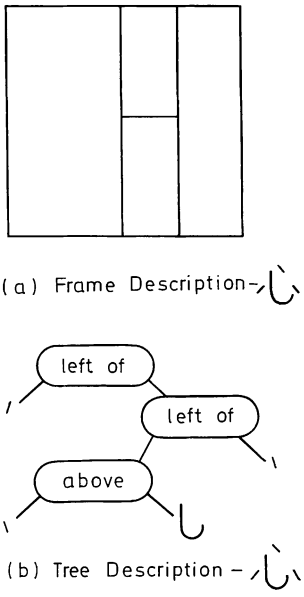


Fig. 5.3a and b. The structure of a Chinese character

The internal structure of components is specified by means of replication, attachment, and superposition of lexical elements. Lexical elements are arbitrarily chosen subpatterns which need not be connected. RANKIN's lexical elements are on the level of complexity of strokes. This work is mentioned only in RANKIN's thesis and is not as well developed as the work on component combinations.

Work on a similar approach has been reported by THOMPSON [5.17].

Fujimura and Kagaya

FUJIMURA et al. [5.18–20] describe a method of specifying the pictorial structure of Chinese characters using a generative grammar. The authors report that the "abstract representation of a character according to the generative rules make it possible to specify the patterns of essentially all Chinese characters completely in terms of elements (strokes) and operators (concatenators and compounders)".

For each of the 17 stroke types isolated by the authors, 3 functional points are designated: at the beginning, middle, and end of the stroke. Concatenators are defined which indicate positional interrelation between two strokes in terms of functional points. For example, two strokes may be concatenated by placing the end of one on the middle of the other. Configurations formed by concatenators are called "units". Figure 5.4 gives one example.

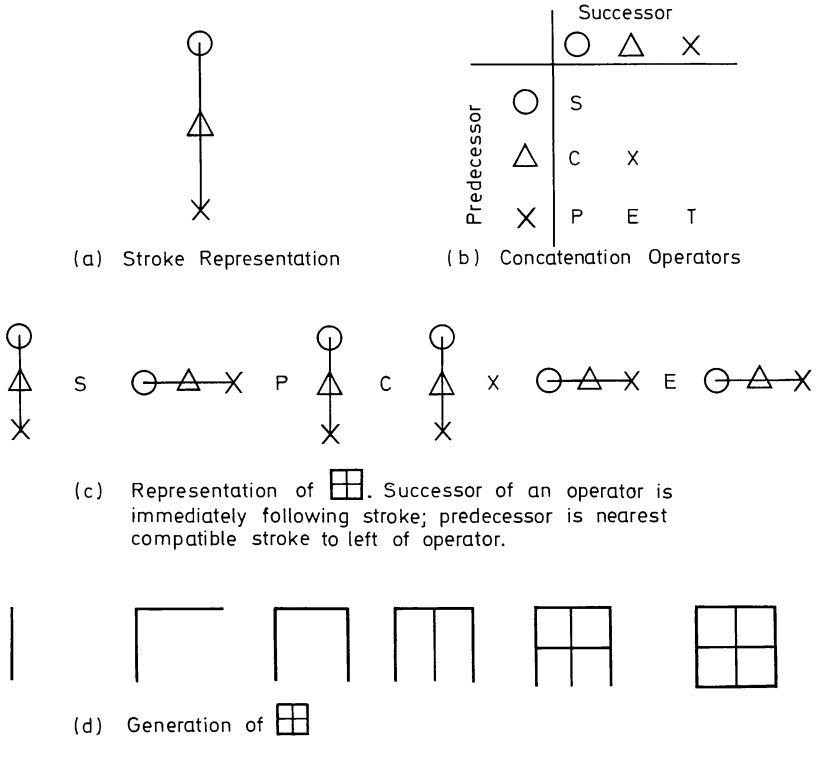


Fig. 5.4a—d. Fujimura's scheme

The process of compounding units involves three compounding operations; horizontal, vertical and surround. The system appears to be quite similar to RANKIN's frame-embedding scheme. To provide for certain complex characters, the authors introduce several "super-concatenators" which permit the superposition of additional strokes on already generated characters. Thus, the grammar is essentially three level, and appears to be better suited to a natural description of Chinese character structure than RANKIN's two-level scheme.

Other Systems

FINLEY [5.21–22] proposes a one-level context-free grammar for Chinese characters employing 15 basic strokes, each with 8 gradations of size, and 6 operators. The operators are: horizontal nontouching, horizontal touching, vertical nontouching, vertical touching, intersect, and touch at a point. The intention of this system, now in a preliminary stage, is to preserve stroke order. It does give a linear encoding, although a clumsy one, of the characters. The recognition problem is considered and a kind of design language proposed. His second paper gives a review of various traditional and modern ways of classifying Chinese characters and a summary of his recent work on simplification of his original encoding scheme.

BURKART [5.23] has done some preliminary work on specifying character structure in terms of white bands of space separating subparts arranged in a matrix. His initial report considers only horizontal and vertical bands of space.

SAKAI et al. [5.24–26] have achieved a description of Chinese characters using 250 subpatterns and 10 operators. The operators are: horizontal touching, vertical touching, horizontal nontouching, vertical nontouching, penetration, surround, and four special operators. The authors claim to be able to build up virtually all characters from these subpatterns and operators.

LEBAN [5.27–28] has developed a model similar in many ways to FUJIMURA's. LEBAN defines 36 elements and 5 composition operators. The elements are intended to correspond to strokes as used by native writers and the description of a character is intended to capture the typical stroke order used by native writers.

Comments

Three comments about structural models of Chinese characters are in order:

1) A distinction should be made, as BURKART points out [5.23, p. 36] between *pictorial* structure and *underlying* structure. Pictorial structure refers to taking a naive view of Chinese characters and attempting to find a logical efficient system for describing their two-dimensional structure. Underlying structure refers to looking at Chinese characters in terms of substructures that would seem "natural" to a user of the language.

Figure 5.5 makes this distinction clear. We find that certain units, though not connected in a topological sense, are found occurring together as intuitively recognizable components; other units are found to consist of more than one recognizable component; still others consist of parts of more than one component.

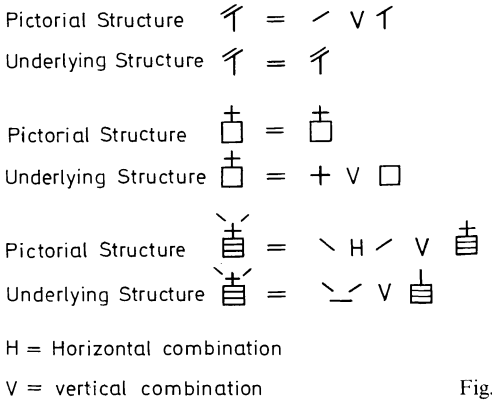


Fig. 5.5. Pictorial structure and underlying structure

It is a matter of taste whether one feels it important to aim at the underlying structure. Pictorial structure is adequate if you define it to be so. It is difficult to categorize the foregoing systems in terms of this distinction. It appears to the author that all of the above systems are a compromise to some degree between pictorial and underlying structures.

2) Another distinction can be made beetween the corpus of Chinese characters in use and the "language" of which that corpus is a sample. All of the grammars discussed hope to be able to generate all characters in use by Chinese writers, but clearly all of these grammars will also generate objects which look like Chinese characters but aren't. For example, any grammar which generates 他, 地, and 你, all of which are Chinese characters, would doubtless also generate 圤, which is not. The most that we can ask is that 1) the grammar generates all Chinese characters in use and 2) all other generated objects are well formed. RANKIN [5.13, p. 3] suggests a test for the acceptability of a grammar under criterion 2). To determine whether a generated object is acceptable, he would ask a native writer the following question: "If you saw the following putative character in a piece of written Chinese text and if you did not recognize it, would you consult a dictionary in order to determine its meaning rather than assuming that the putative character was no character at all?"

3) The order in which the strokes of a Chinese character are drawn is predictable. That is, when native Chinese are taught to draw Chinese characters they are taught a particular stroke sequence. With the exception of LEBAN'S and FINLEY'S, none of the discussed grammars reflects this fact, although it does provide an interesting means of characterizing pattern structure. But see below.

5.2.2 Some Applications

This section surveys some practical devices and systems which depend in some sense on the structure of Chinese characters [5.11]. In some but not all cases, the application described was inspired by one of the formal studies mentioned in Subsection 5.2.1.

Keyboard Input

Because of the many thousands of Chinese characters in common use, a keyboard (for typesetting, typewriting, on-line computer usage, etc.) for Chinese is an ungainly affair. A number of rather different approaches to reducing the keyboard size, all of which exploit some structural characteristics of Chinese characters, have been taken.

1) *Sinotype/Ideograph Composition Machine.* The proposal for this machine was originally made by LEE [5.29], who suggested the description and encoding of characters in terms of a sequence of strokes. CALDWELL [5.30] expanded the proposal into the development of a Sinotype. Work on the machine was continued at RCA, where it was called the Ideograph Composition Machine [5.31, 32].

One component of the Sinotype is a keyboard containing keys for all the basic strokes. To input a character, the operator types the keys corresponding to the sequence of strokes traditionally used to write the character. The sequence of keys typed generates a unique code for the character. This code is used by the machine to select the proper character for output. The stroke-sequence description used here is somewhat similar to the stroke-attachment scheme of RANKIN.

2) *Sinowriter/Chicoder.* The Sinowriter developed at IBM [5.7, 33] uses a geometric scheme which requires the operator to recognize particular shapes in the upper and lower portions of the character. In inputting a given character, the operator decides which of 36 upper configurations it contains and which of 30 lower ones. The appropriate keys are typed and the operator is presented with a family of no more than 16 characters from which final selection is made. A modified version called the Chicoder was developed by ITEK [5.34]. A similar device has been proposed by KIANG [5.35]. Instead of keying in an upper and lower pattern, the operator would key in the first and last strokes of the character and then be given the opportunity to choose the correct character from those displayed.

3) IBM *Chinese Keyboard.* This device [5.36] takes advantage of the fact that each Chinese character contains a subpattern, known as a radical. The total number of radicals for all characters is 214. The device consists of two keyboards. The smaller keyboard is a matrix containing the 214 radicals. The larger keyboard is a 128×128 matrix of characters. The operator first depresses a radical key. This causes the portion of the larger matrix containing characters with that radical to light up. It is then comparatively easy to select the desired character.

4) *The Chinese Computer Alphabet.* This rather ingenious system [5.37, 38] depends on decomposing each Chinese character into several "strokes". The 22 strokes are not the traditional strokes used in writing Chinese characters but are chosen to be easily recognizable shapes within the characters. Corresponding to the 22 strokes are 22 letters and special characters found on most keyboards. Each character has roughly the same shape as the stroke it signifies. To input a character, the operator spells out the alphabetic strokes of which it consists. A set of simple rules is used to determine the order in which strokes are entered.

5) *SINCoder.* This device was developed by LEBAN [5.27, 28] and is based on his structural scheme discussed in Subsection 5.2.1. The means of input is some-

what similar to the Chinese Computer Alphabet scheme. Each of LEBAN'S 36 basic symbols is assigned a letter code. Composition of a character is achieved by input of a sequence of these strokes alternating with his five composition operators. Certain simple rules are used to decide stroke order; these rules are similar to the traditional method of constructing characters.

6) *Stand-Alone Chinese Typewriter.* This device [5.39, 40] is similar to the SINCoder. It uses 78 strokes and radicals as the basic symbols and has composition operators similar to RANKIN'S.

7) *Dissembled Chinese Characters.* SU [5.41, 42] has developed a system based on the assumption that the spatial arrangement of the components of a Chinese character is not essential to either its unique representation or recognizability. SU contends that, with almost no training, a person who knows Chinese can read and write Chinese characters spelled out in terms of their components. A set of 210 components, together with three notational signs to resolve ambiguities, is used. SU is developing an input/output system including a keyboard containing the 210 symbols. A similar method has been developed by TING and HORNG [5.43, 44].

Chinese Character Storage and Retrieval

Here we survey three systems for computer storage and retrieval of Chinese characters.

1) *Harvard System.* A group of researchers at Harvard [5.42, 45, 46] has been working on an input/output system for Chinese characters and other non-alphabetic orthographies. Selection is achieved by keying in the radical and the number of strokes in the remainder of a character. All characters fitting this description are then displayed for final selection.

2) *Chang.* CHANG has developed a quite sophisticated system [5.47, 48] which makes use of a formal structural description of Chinese characters. The scheme developed by SAKAI et al. was adopted and modified by CHANG.

CHANG has developed a grammar for Chinese characters using 11 composition operators and a number of primitive components. Each character can then be described as a tree structure (see Fig. 5.3).

Based on this grammar, CHANG has created a hierarchically organized data base of Chinese characters. The organization makes use of the fact that a given Chinese character is either a simple component, or it is made up of subparts, which in turn are either simple or complex. Further, any subpart of a character may itself be a character. Hence, we can create a hierarchy of Chinese characters in which each character is "pointed to" by all those characters which contain it as a subpart and "points to" each of its subparts.

Taking advantage of this organization, character selection is done by means of *component and container searching.* A component of a character is any character below it in the hierarchy to which there is a path; a container is any character above it from which there is a path. To select a character, the operator keys in the codes for components and/or containers of the character. When sufficient codes

have been entered to select a unique character, the system responds by displaying the appropriate character.

For display, a character generation scheme was developed for generating a dot matrix from the structural description of the character. Thus the grammar is used as a generative grammar in addition to its function of organizing the data base.

3) *Intelligent Terminal for Chinese Characters.* This is a proposed system [5.49] built around the IBM Chinese keyboard described earlier. As this is simply a proposal, alternative features are discussed, some of which are similar to the Harvard system, some similar to CHANG's system.

5.3 Recognition of Chinese Characters

5.3.1 The Model

The model of Chinese characters used in this recognition scheme is that of a two-dimensional arrangement of one or more disjoint components, each component being a maximally connected subset of the character. Figure 5.6 shows a character with three compnents.

The structure of a Chinese character may therefore be specified on two levels:

i) a description of the internal structure of each component, and
ii) a description of the arrangement of components in two dimensions.

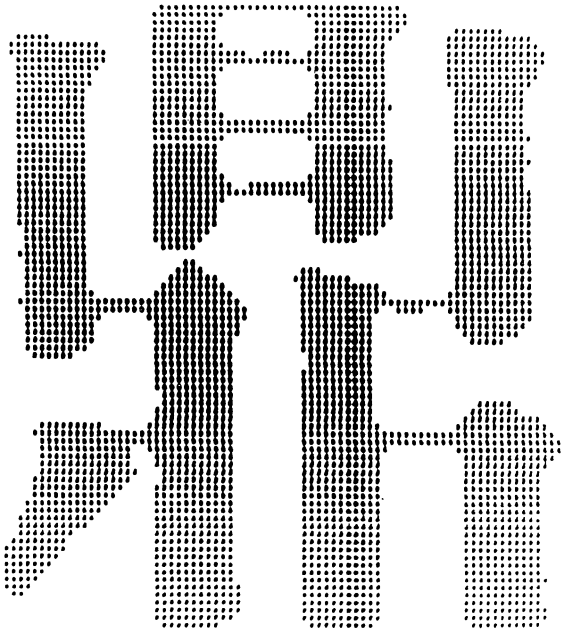


Fig. 5.6. Character with three components

Components

Two questions are involved in the decision of how to describe the internal structure of a component:

i) What class of objects shall be considered as the basic picture element?
ii) What sort of structure shall be used to indicate the relationship between elements?

Three criteria were used in answering these questions:

i) The structure mentioned in question ii) should be relatively easy to generate from the original pattern.
ii) It should be relatively easy to generate a unique numeric code from the structure.
iii) The structure should represent the pattern in a natural manner.

A quite natural method of representing the internal structure of a component would be in terms of strokes. This indeed is the approach taken by several previous recognition schemes [5.50, 51]. These schemes make use of on-line input, in which strokes are drawn one at a time. The difficulty with taking this approach for printed characters is that strokes do overlap and are not easily isolated. Further, the description of the relationship between strokes is not straightforward.

A much more promising approach is to describe components in terms of stroke segments. This can best be understood with reference to Fig. 5.7. As can be seen, a component can be depicted as a graph. The branches of the graph correspond to segments of strokes. These segments are bounded by stroke intersections and ends of strokes.

It will be shown in later sections that this representation satisfies criteria i) and ii). That is satisfies criterion iii) is fairly clear. To the human observer, the graph of a component is readily apparent.

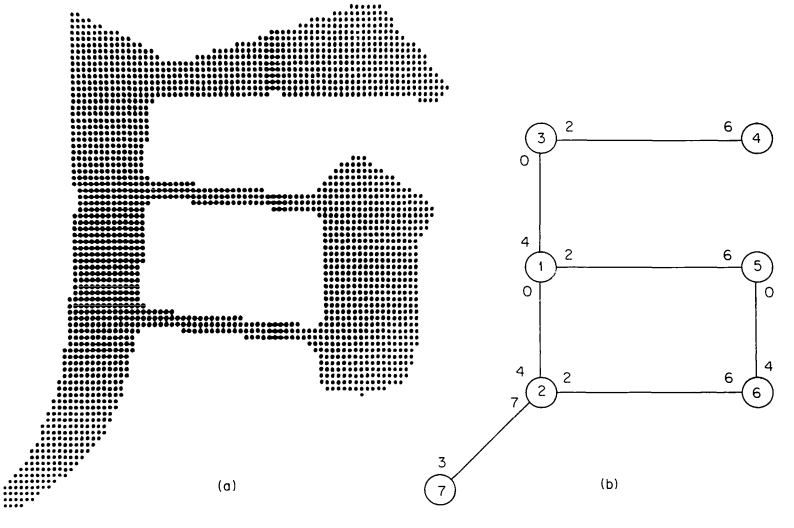


Fig. 5.7a and b. Component and graph

Characters

The arrangement of components in two dimensions to form characters is described using RANKIN's frame-embedding scheme. The definition of component used here is slightly different from that of RANKIN. Despite this, RANKIN's claim that the three relations used in his scheme are sufficient to describe accurately nearly all characters seems to apply.

5.3.2 Input

The program operates on a representation of one character at a time. The representation is in the form of a matrix whose entries have value zero or one corresponding to white or black in the original picture. The matrix is obtained by means of a flying-spot scanner. The printed characters used were taken from a number of different sources; the characters were all of roughly the same style but varied considerably in size.

Certain functions of the program depend on the fact that there are no gaps or holes in any of the strokes. This is not always the case due to the quality of the printed input. Accordingly, a smoothing operation is performed to fill in the gaps. The resulting matrix is used as the data base for the program.

The digitized form of a character can be displayed on a CRT. Figures 5.6 and 5.7a are photographs of such displays.

5.3.3 Analysis of Components

A program has been written to perform the analysis of components. For a given component, the output of the program is a graph in which branches correspond to stroke segments and nodes correspond to the endpoints of stroke segments.

To construct the graph of a component, one principal procedure, BUILD, is used. In addition, use is made of some auxiliary routines. It will be helpful to describe these first.

Contour Tracing

Contour tracing is the process of finding a series of black points on the boundary of a black region in a white field. Two routines are used: one which keeps the black region on the left as the tracing proceeds, and one which keeps the black region on the right.

To keep the black region on the left, the tracing proceeds from point to point, turning right after encountering a black point and left after encountering a white point. An additional rule is used to increase the speed of the algorithm. If three points of the same color are encountered in succession, the next point is assumed to be of the opposite color. Thus, two steps may be taken at once. The operation of the algorithm is depicted in Fig. 5.8. The last step shown is diagonal, indicating the effect of the 2-move rule.

The algorithm for keeping the black region on the right is similar. Both algorithms were developed by PRERAU [5.52].

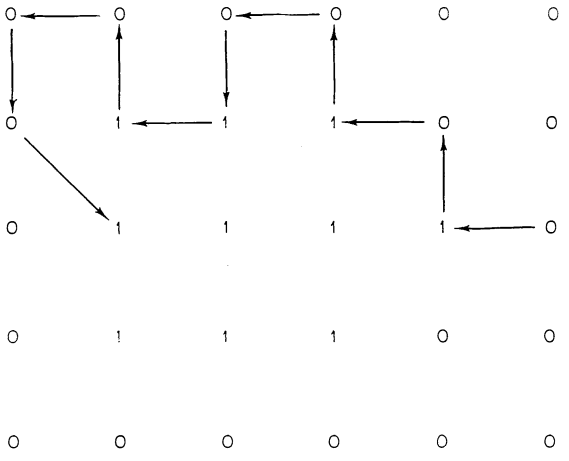


Fig. 5.8. Contour tracing

Search

The task of the SEARCH routine is to find some stroke segment to be used as a starting point. It is unimportant which particular segment of a component is found.

The output of the SEARCH routine is the coordinates of the endpoints of a strip of black points straddling a stroke segment. SEARCH proceeds by scanning alternately from left to right and from top to bottom along various rows and columns of the pattern. This continues until a series or strip of black points is encountered. If the strip is too long (more than $1/4$ the width of the pattern), it is assumed that the strip is lying along the length of a stroke. This is rejected and the scanning continues.

Figure 5.9 shows examples of all possible outcomes of scanning a single row.

Crawl

The CRAWL routine is used for "crawling along" a stroke segment. The routine proceeds along a stroke segment in a given direction, halting when a node is encountered, i.e., when an intersection or the tip of a stroke is reached.

The input to CRAWL is i) a location on a segment, in the form of the two endpoints of a horizontal or vertical strip of points straddling the segment, and ii) one of four directions (left, right, up, down) in which the crawl is to proceed. The output is the location on the segment where the crawl halted, again in the form of two endpoints of a strip.

The crawl is accomplished by moving from each of the input points along the contour of the segment. Tracing from the left-hand input point (with respect to the direction of the crawl) is done keeping the black region on the right and conversely for the right-hand input point. The crawling proceeds by advancing both "tracers" one unit in the specified direction at a time. This is depicted in Fig. 5.10. For each move from one line to the next, each tracer goes through one or more contour points.

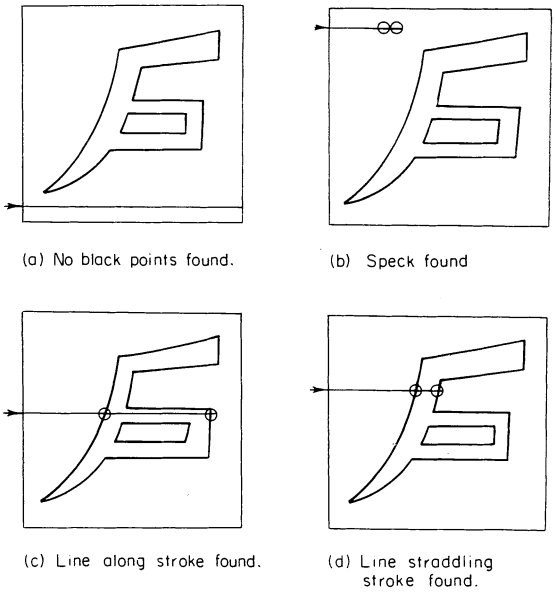


Fig. 5.9a—d. Outcomes of a scan by SEARCH algorithm

Figure 5.11 shows the four conditions under which a crawl will be halted. All four cases correspond to a node being encountered.

i) If the two tracers, instead of advancing, meet each other, then the tip of a stroke has been encountered.
ii) If the two tracers do advance, but not all of the points between them are black, then a fork has been encountered.
iii) If the new strip of black points on which the two tracers sit is significantly longer than the previous strip, then an intersection has been encountered.
iv) If one of the two tracers reverses direction, then again a fork has been encountered, but this time by coming up one of the two arms rather than the main road.

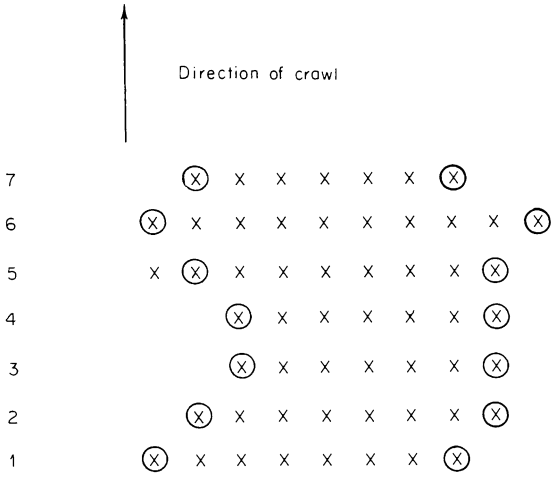


Fig. 5.10. Crawling along a stroke

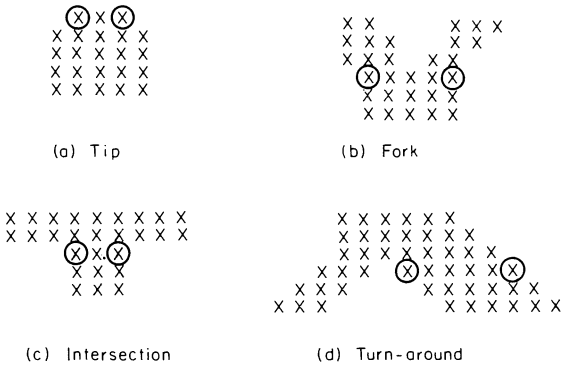


Fig. 5.11a—d. Conditions for halting crawl procedure

Although only horizontal and vertical directions of crawl are specified, the routine works on diagonally oriented segments. Notice that in Fig. 5.10 both tracers move diagonally from line 1 to line 2. This could continue along the entire length of a diagonal segment.

Node

After CRAWL has encountered a node, NODE is called to investigate it. The input to NODE is the output of CRAWL: the endpoints of a strip of points which marks the termination of a segment or an intersection. The task of the NODE routine is to find all other stroke segments radiating from this intersection. For each segment found, NODE returns the endpoints of a strip straddling that segment at the intersection. Also, the direction of the segment away from the node is indicated.

The operation of NODE is shown in Fig. 5.12. The routine starts at one of the input points and proceeds along contour points around the intersection. This continues until a contour point is found which is the endpoint of a horizontal or vertical strip straddling a segment (i.e., the endpoint of a small black strip). This strip and the direction perpendicular to it away from the node are noted. The routine then continues from the other endpoint of the strip. This process of going a few contour points, finding a segment, crossing it, going a few contour points, etc., continues until the other input point is encountered.

In addition to locating the segments leading from a node, the routine assigns a position to the node. This is done by averaging the X and Y coordinates (with respect to an origin in the upper left-hand corner of the matrix) of the endpoints of all the strips found, including the input points.

Build

The construction of a graph can now be described. As a graph is a collection of interconnected nodes, it is represented in the computer as a collection of interconnected blocks of data. For each node in a graph, a block of contiguous memory

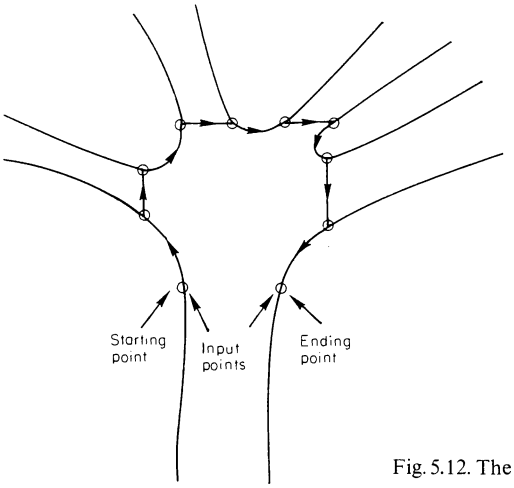


Fig. 5.12. The NODE algorithm

words is allocated. The length of a block depends on how many branches there are at the corresponding node. If two nodes are adjacent in a graph, their data will contain pointers to each other. Each of these pairs of pointers represents a branch.

To begin construction of a graph for a particular component, SEARCH is called to find some initial stroke segment. SEARCH returns a position somewhere along the length of a segment. From this position, CRAWL is used to crawl along the segment in both directions to its two endpoints. Thus two initial nodes are found. NODE is called once for each endpoint to determine the segments leading from them. Storage blocks are allocated for each node. Pointers are placed in each block linking the two together.

From this start, the graph is completed using BUILD. BUILD is called once for each segment leading from each of the two initial nodes. The arguments to BUILD are i) a pointer to a block of data corresponding to a node (the input node), and ii) the starting point of some segment (the input segment) leading from the input node. BUILD performs the following operations:

1) The input segment is crawled along to reach its endpoint, using CRAWL.
2) NODE is called to examine this endpoint, or node. The coordinates of the node and the segments leading from it are determined.
3) a) The coordinates of this node are compared to those of all previously encountered nodes (those for which data blocks already exist). If a match is found, then a pointer to the existing block for this node is placed in the block of the input node, and the routine stops.
b) If the encountered node is new, then a block is allocated for it, and it is linked back to the block of the input node. BUILD is then called once for each segment leading from the new node. Then the routine stops.

It can be seen that BUILD is a recursive routine. BUILD is described more formally in Fig. 5.13. As an example, the analysis of the component of Fig. 5.7 will be described. The two nodes initially found are marked 1 and 2. The branch between them corresponds to the initial segment found by SEARCH. Blocks of

```
Procedure build (block, stroke):

begin node:=find node at end of stroke;
      n:=number of other branches at node;
      branch:=n-vector of other branches at node;
      if node=oldblock*
      then place pointer to oldblock in block
      else begin
  newblock:=create block of length n+5;
  place pointer to newblock in block;
  place pointer to block in newblock;
  place number, x,y in newblock;
  for i:= 1 step 1 until n
  do build (newblock, branch (i))
            end
```

* i.e., node is compared to all nodes previously encountered. The value is true if node is the same as another node represented by the data block "oldblock".

Fig. 5.13. BUILD procedure

data are allocated for 1 and 2. Then, all the segments leading from 1 are examined, clockwise, by BUILD. Crawling along the first segment, node 3 is found. This is linked back to 1. The segment leading from 3 is examined next, finding node 4. The procedure unwinds back to node 1 and examines its next segment. As a result, 5 and 6 are found. From 6, node 2 is encountered. Node 6 is linked to node 2 and the procedure again returns to node 1, which is seen to be completed. Next BUILD is applied to node 2 which finds first 6 and then 7. At this point, 2 is complete and the analysis terminates.

5.3.4 Analysis of Characters

The algorithm for analyzing a character is in two parts:
1) A collection of graphs is produced, one for each component.
2) The relationship between components is determined.

Finding All Components

The first part of the algorithm involves a few modifications to the program discussed in the previous subsection. The objective is to keep track of which components in a pattern have already been analyzed.

To do this, the following procedure is employed. As a component is being analyzed, its outline is drawn on a separate pattern. That is, the contour points of a component are filled in on a new pattern as they are encountered. The new pattern contains, at any time, the outline of all components of a character which have been processed. The SEARCH routine is modified to test the endpoints of any strip of black points it finds against the new pattern. If the corresponding points are black in the new pattern, then the strip is rejected and SEARCH continues to scan. If no new strip is found after scanning a sufficiently large number of rows and columns, it is assumed that no new components remain to be

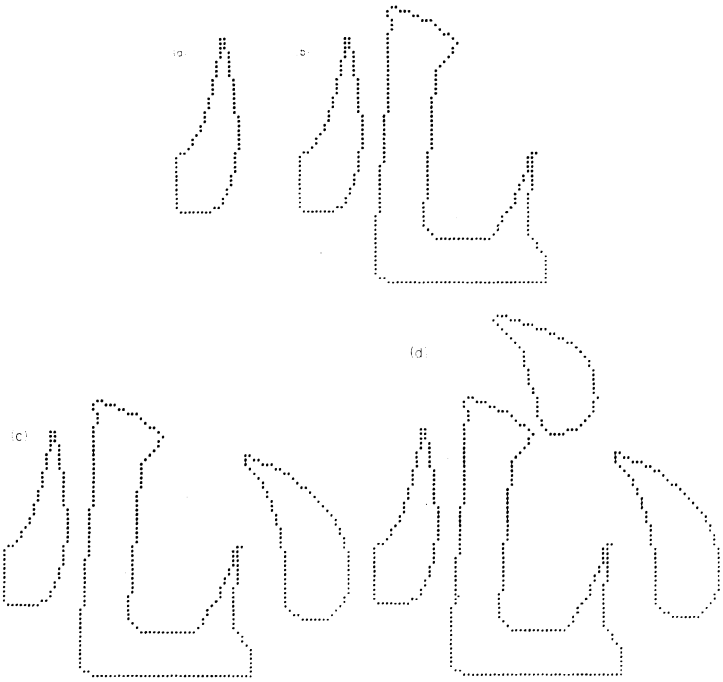


Fig. 5.14a—d. Outline of a character

found. After each component is analyzed, SEARCH is called to locate a stroke segment on a new component. The process of analyzing components continues until no new components can be found. The result is to produce a collection of connected graphs.

Figure 5.14 shows the result of applying the algorithm to the character of Fig. 5.3.

Constructing the Frame

Representation of the frame description of a character is done conveniently by means of a tree. The root node of the tree has as its value one of the three relations indicating how the overall frame is broken into two subframes. The two sons represent the structure of the two subframes. Terminal elements correspond to components (see Fig. 5.3).

The method of obtaining such a tree will be briefly described. First, each component in the character is inscribed in a rectangle. This is easy to do since the coordinates of each node are known. The relationship between all possible pairs of components is determined by determining the relationship between their rectangles. The one of the three permitted relationships (East-West, North-South, Border-Interior) which most nearly approximates the true relationship is chosen. Then it is determined whether one of the components has the same relation to all other components. This will usually be the case. If so, that component becomes

```
Procedure frame (list, tree);
begin list1:=first group of components;
      list2:=second group of components;
      node:=relation between two groups;
      if list1 is a list
        then frame (list1, tree1)
        else tree1:=list1;
      if list2 is a list
        then frame (list2, tree2)
        else tree2:=list2;
      tree:=tree1, node, tree2
end
```

Notes:
1. The input to frame is the argument list, which is a list of combinations of two or more components taken two at a time.
2. The output of frame is the argument tree which is a triple corresponding to the left son, node, and right son of a tree.
3. list1 and list2 represent disjoint groups of components such that the two groups have one of the three allowed relations between them. If either group contains only component, the corresponding variable (list1 or list2) is simply an identifier of that component and not a list.

Fig. 5.15. FRAME procedure

one son of the root node of the tree; the value of the node is the appropriate relation; the other son is a tree representation developed for the remaining components. This subtree is determined in the same way.

If no single component is found, a more complicated procedure is used to determine whether any two components have the same relation to all others, and so on. A procedure for constructing the tree representation of a frame description is described formally in Fig. 5.15.

5.3.5 Encoding of Components

For recognition purposes, a procedure has been developed for generating a numeric code for each character. The first step in this procedure is the generation of a code for each component in a character.

The code for a component is generated from its graph. To this end, the branches of a graph are labeled at each end. The label on a branch at a node indicates the direction or slope of that branch quantized into eight directions. All the branch labels at a node are stored in the data block of that node. An algorithm can then be specified for starting at a particular node of a graph and traversing all of its branches. The sequence of branch numbers encountered is the code produced. An example appears in Fig. 5.16.

The algorithm obeys the following rules:

1) Start at the node in the upper left-hand corner of the graph. Exit by the branch with the lowest-valued label. Mark the existing branch to indicate its having been taken, and write down the branch label.
2) Upon entering a node, check to see whether it is being visited for the first time. If so, mark the entering branch to indicate this.

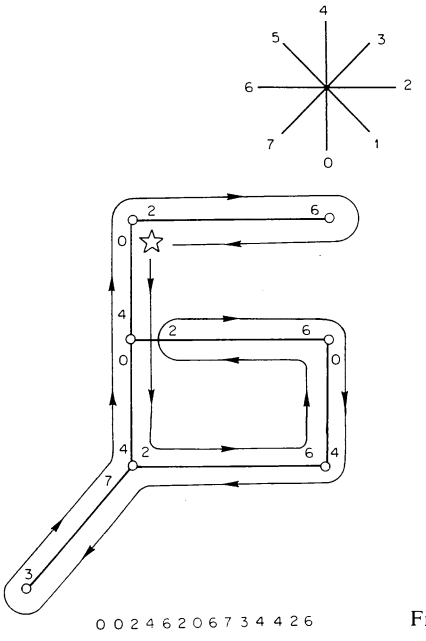


Fig. 5.16. Encoding a graph

3) Upon leaving a node, if there are available unused directions other than along the first entering branch, choose the one among these with the lowest-valued label. Leave by the first branch only as a last resort. Mark the exiting branch to indicate its having been taken and write down the label on the branch.

Since at each node there are just as many exiting branches as entering branches, the procedure can halt only at the starting node. At the starting node, all exiting branches have been used (otherwise the procedure could have been continued), hence all entering branches have been used since there are just as many of these. The same reasoning can be applied to the second node that is visited. The first entering branch is from the starting node and this branch has been covered both ways. But this branch would only have been used for exit from the second node if all other exits had been exhausted. Therefore all branches of the second node have been covered both ways. In this manner, we find that the branches of all nodes visited have been traversed both ways. Since the graph is connected, this means that the whole graph has been covered.

All branches are traversed exactly once in each direction by this procedure, so all labels are picked up. The code consists of the branch labels in the graph written down in the order in which they are encountered.

This algorithm is based on a procedure for traversing graphs described in Ore [5.53].

While this scheme will always generate the same code for a given component, the goal of generating a unique code for each component is not achieved. For example, 土 and 士 are represented by the same graph, hence the same code. Fortunately, this type of situation is rare. Characters with this property could be treated as special cases without seriously imparing the efficiency of the algorithm.

5.3.6 Encoding of Characters

The representation of characters is in the form of a tree. The nodes of the tree are binary relations; the terminal elements correspond to components. Considering the relations as binary operators, the tree can easily be flattened to prefix form. This is done by walking around the tree counterclockwise, starting from the root node, and picking up nodes and terminals the first time they are encountered. As is well known, the string generated in such a fashion is unique; the tree can readily be reconstructed from it. To generate a numeric string, the following code can be used:

$0 \leftrightarrow$ terminals (components)

$1 \leftrightarrow$ left node

$2 \leftrightarrow$ above node

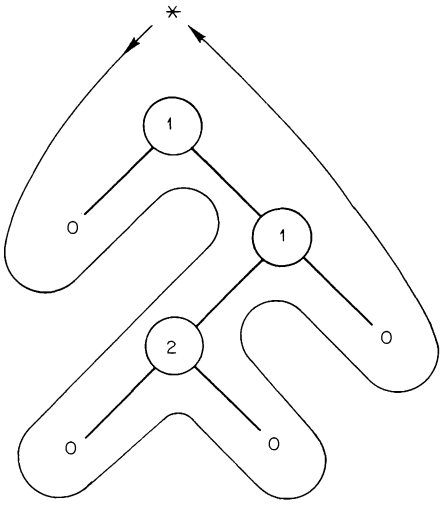
$3 \leftrightarrow$ surround node

Figure 5.17 shows the generation of code from the tree of Fig. 5.3.

We can consider that the code so generated defines a class of Chinese characters, all of which have the same frame description. Therefore, a Chinese character may be specified by first giving its frame description code and then giving the code for each of the components that fits into one of the subframes. A character having n components will have a code consisting of the concatenation of $n+1$ numbers

$$N_0,\ N_1,\ \cdots,\ N_n$$

where N_0 is the code generated from the tree and N_1 through N_n are the codes of the components listed according to the order in which the components were encountered in the tree flattening.



1 0 1 2 0 0 0

Fig. 5.17. Flattening a tree

5.3.7 Results

The algorithms discussed in this chapter have been implemented as a computer program. The program is written in FORTRAN augmented by a package of assembly language routines to permit structured data and recursive procedures. The program runs on a PDP-9 computer.

The program has been tested with a number of characters from several different sources. The tests were designed to consider four questions:

1) How successful is the program in analyzing the structure of Chinese characters?
2) Does the program generate consistent codes for characters of the same font? That is, will two instances of the same character from the same source yield the same code?
3) Does the program work for characters from different sources?
4) Do factors such as character size and character complexity affect program performance?

Initial results were obtained from a set of characters obtained from a Taiwan printer. A sample of this set appears in Fig. 5.1. To start, 225 different characters were processed. This was to provide a dictionary for later tests, and to test the pattern analysis capabilities of the program.

The results show a reasonable structural representation produced for about 94% of the characters. The failures were all due to a particular component not being analyzed; for all characters the relationship among components was correctly determined. The problems all occurred in the NODE routine, which is supposed to isolate a node and locate all segments leading from it. The NODE routine would sometimes make mistakes if, for example, two nodes were very close together or one node covered a large area. The characters involved were typically quite complex.

From the characters that were successfully analyzed, 25 were chosen for additional testing. Four additional instances of each character from the same source were processed, for a total of 100 new characters.

All new instances of the 25 characters produced reasonable structural representations. For five of the characters, one of the new instances produced a slightly different representation, hence a different code. No character generated more than two codes. In all cases, the discrepancy was caused by the fact that two strokes which were very close in one instance touched in another instance of the same character.

Additional testing was done using two other sources. Characters from issues of a Chinese magazine were used. These were approximately half the size of the characters in the original set. Also, some computer-generated characters [5.54] were used. These were about double the size of the originals. Both were of about the same style. Fifty instances were taken from each source. The percentage of instances generating the same code as the corresponding character from the original set was 89% for the magazine source and 95% for the computer source. Discrepancies mostly had to do with stroke segments appearing at somewhat different angles and with strokes touching in one case but not the other.

5.3.8 Conclusions

Pattern Analysis

A descriptive scheme for the structure of Chinese characters has been proposed and a program for computer analysis conforming to the scheme has been written. The description is on two levels: the internal structure of components, and the relationship among components.

The first level of description is straightforward: a connected part of a character is represented by a graph. This representation is adequate for the description of components; it is reasonable for the human percipient to think of components as graphs.

Analysis on this level works fairly well; difficulty is encountered with some complex characters. Some work has been done on modifying the described approach. The modification consists of "shrinking" a component to a skeleton and obtaining the graph from the skeleton. This procedure is sensitive to contour noise, and it seems that use of this method would result in many components generating several different graphs from different instances.

The second level of description is based on the work of RANKIN. With the exception of a very few characters whose components do not fit neatly into the frame description, it is an effective means of describing the structure of Chinese characters in terms of components. The analysis program for this level has been successful for all characters tested.

Character Recognition

Chinese character recognition is made difficult by the size of the character set and the complexity of the individual characters. Test results indicate that use of the approach described here would necessitate a dictionary in which some characters are associated with several codes.

Several possibilities exist which could improve the chances of constructing a practical character recognition device.

1) *High Standards of Print Quality.* A device restricted to use only with very high-quality print should be more consistent in code generation, thus reducing the size of the required dictionary.
2) *Stylized Font.* A specially designed font tailored to the recognition algorithm would improve the algorithm's performance.
3) *Language Simplification.* A particularly hopeful development in this regard in the program of the People's Republic of China to reduce the number of characters in general use and the complexity of individual characters [5.55, 56].

The results reported here lead the author to believe that pattern analysis can be a fruitful approach to Chinese character recognition.

5.3.9 Other Approaches

In recent years, a considerable amount of work has been done on Chinese character recognition [5.57]. This work falls into three main categories: printed Chinese

characters, hand-drawn Chinese characters, and on-line hand-drawn Chinese characters. A related body of work deals with analysis of on-line hand-drawn Chinese characters that falls short of a full recognition capability. These four areas are surveyed briefly.

Printed Chinese Character Recognition

Perhaps the most important area of investigation deals with printed Chinese characters. An effective printed character recognition system would permit the rapid processing of vast amounts of printed Chinese material. Not surprisingly, most of the work on Chinese character recognition has focused on printed characters.

The earliest reported attempt at printed Chinese character recognition is that of CASEY and NAGY [5.58]. Their method consisted of a two-stage template-matching process. Sixty-four "group masks" are used to divide the character set into distinct groups of characters with similar topological features. A given character is matched with all masks to determine the closest fit, thereby assigning it to one of the groups. Corresponding to each group are a number of "individual masks" which discriminate among members of the groups.

A group of researchers at the Hitachi, Ltd. in Japan have developed a recognition technique based on the projection of Chinese characters onto the X and Y axes [5.59, 60]. The authors observe that Chinese characters are typified mainly by the presence of vertical and horizontal strokes in specified positions and that this form is almost always well preserved in their projected profiles. Recognition is achieved by comparing the amplitude spectra of the profiles, the absolute value of the Fourier transform of the profiles, to those of known characters.

The research group at Hitachi has investigated another approach which they refer to as hierarchical pattern matching [5.60–62]. The approach, reminiscent of that of CASY and NAGY, involves creation of a 4×4, 8×8, and 16×16, and 32×32 gray-level matrix for each character in the set. These are then used in a four-stage template-matching algorithm.

Wang has porposed a three-stage recognition process based on topological properties of the characters plus the use of transformation algorithms [5.63–66]. As yet, the system is only partially implemented. For a three-stage system, the character set is divided into a number of groups based on a topological feature or set of features. Each group is in turn divided into subgroups based on the Fourier transform of a subset of the character matrix. Finally, a means is found for discriminating among the members of a subgroup. To recognize a character sample, it is first determined to which main group it belongs. Within that group, the appropriate subgroup is found. Finally, the sample must be assigned to one of the members of the subgroup. Compare this approach to the two-stage process of CASEY and NAGY, and to the four-stage process of the Hitachi group.

ANKENY [5.67] has developed a recognition scheme which is essentially an automated version of the Sinowriter or Chicoder. Predefined geometric shapes are searched for in the upper and lower portions of a character. Shape recognition is by means of two-dimensional discrete Fourier transform matching.

Handwritten Chinese Character Recognition

Somewhat more difficult than the recognition of printed Chinese characters is the recognition of handwritten Chinese characters. Not only is the variability of writing style in a given individual greater than that for a single font of printed characters, but differences in writing style from individual to individual are greater and less predictable than font-to-font differences [5.68].

YOSHIDA and EDEN [5.69] report on a method for handwritten Chinese character recognition based on an analysis-by-synthesis technique. The authors take the point of view that Chinese characters can be drawn as a sequence of a few fundamental strokes and that each stroke in a handwritten character can also be described in terms of a fairly simple model of penpoint movement during handwriting. The algorithm attempts to reconstruct the stroke-drawing process and recognize the character on the basis of the sequence of strokes.

MA [5.70, 71] has developed a method of recognition using a sequential decision procedure. For each character, a probability matrix depending on the relative frequency of stroke occurrence at each position is obtained by averaging over many samples. The sequential decision procedure is based upon a risk function, which is calculated by comparing the input pattern to the prototype patterns. The risk function is used to apply a "stopping rule" so that the amount of computation required to accept or reject a given character class for assignment to the input pattern is minimized.

An experiment in handwritten Chinese character recognition via diffraction patterns has been reported by a group at Grumman Aircraft [5.72]. The film transparency of a character is illuminated with spatially cleaned and collimated light from a laser and its farfield diffraction pattern displayed in the back focal plane of a lens. It was decided that the resulting diffraction patterns for Chinese characters were not useful for recognition.

On-Line Recognition

Several investigators have reported on work in on-line Chinese character recognition. Input for on-line character recognition is dynamic, using a suitable device such as a tablet or light pen. Two advantages accrue from such an approach.

1) Since the strokes are drawn one at a time, they are easily identifiable. The problem of isolating strokes which overlap is eliminated.
2) The sequence in which strokes are drawn is known. Traditionally, the strokes of a Chinese character are drawn in a fixed order.

LIU [5.51] reports on a real-time recognition scheme using a Rand tablet. The data used were the number of strokes of each type (LIU distinguishes 19 types) in a character and the type of the first and last strokes drawn. This was found to be sufficient for recognition.

A method of on-line recognition based on the sequence of pen movement directions has been reported by a group at the University of Pittsburgh [5.73–75]. As a character is drawn, the sequence of (X,Y) points encountered is stored. An algorithm has been developed for generating from the sequence of points a sequence of a fixed number of stroke segments—a pattern vector of fixed dimension.

This pattern vector is compared to the prototype pattern vector of each character class to determine closest fit. The comparison function is developed by means of a learning algorithm.

A system somewhat similar to the one just described has been reported by CHANG and LO [5.76].

On-Line Analysis

Less ambitious than the on-line recognition of Chinese characters is the on-line analysis of Chinese characters falling short of recognition. Several useful applications have been reported.

GRONER et al. [5.50] have developed a method that distinguishes five types of strokes. Characters are classified by stroke sequence. When a character is input from the tablet, its stroke sequence is determined and an array of all the characters with the sequence is presented on a display screen. The operator may then choose the intended character. This scheme is used as an aid in dictionary look-up characters.

A very interesting use of on-line techniques is reported by CHEN and CHUANG [5.77, 78]. Their system is one for teaching students how to write Chinese characters. Their characters are stored as structures which represent a two-dimensional arrangement of components, each component being a set of strokes in a specified sequence. This is very much like RANKIN'S representation. The system is capable of drawing a character in slow motion, so that the student may see the constituent strokes and the proper sequence for drawing them. More importantly, the system will accept a character drawn by a student on a tablet and compare it to the internal representation of that character. The system detects errors in stroke type, stroke sequence, relative position and size of strokes, cross points between strokes, and relative position and size of units. This is not true recognition, as the system knows ahead of time which character is being attempted.

Similar, but seemingly less ambitious CAI projects have been reported by TOU [5.79] and CHENG [5.80].

References

Abbreviations:

IOJC Preprints for Seminar on Input/Output Systems for Japanese and Chinese Characters, U.S.-Japan Committee on Scientific Cooperation, Tokyo (1971)

UJCC Proceedings of the First U.S.A.-Japan Computer Conference, Tokyo (1972)

PPR Proceedings of the First International Joint Conference on Pattern Recognition, Washington (1973)

CCIO Proceedings of the First International Symposium on Computers and Chinese Input/Output Systems, Taipei (1973)

5.1 U. GRENANDER: Advan. Computers **10**, 175 (1970)

5.2 K. M. SAYRE: Recognition: A Study in the Philosophy of Artificial Intelligence (U. Notre Dame Press, Notre Dame 1965)

5.3 W. STALLINGS: PhD Thesis, MIT (1971)

5.4 W. STALLINGS: "Computer Description and Recognition of Printed Chinese Characters", SJCC (1972)

5.5 W. STALLINGS: Comp. Graphics Image Proc. **1**, 47 (1972)
5.6 P. L. GARVIN: Defense Documentation Center Rept. AD 627763 (1966)
5.7 F. KING, H. CHANG: Sci. Am. **208**, 124 (1963)
5.8 W. WANG: Defense Documentation Center Rept. AD 850009 (1969)
5.9 L. YUTUNG: Defense Documentation Center Rept. AD 414397 (1963)
5.10 W. WANG: Sci. Am. **228**, 50 (1973)
5.11 W. STALLINGS: Computers & the Humanities **9**, 13 (1975)
5.12 B. RANKIN, W. SILLARS, W. HSU: NBS Tech. Note 254 (1965)
5.13 B. RANKIN: PhD Thesis, University of Pennsylvania (1965)
5.14 B. RANKIN, S. SIEGAL, A. MCCLELLAND: NBS Tech. Note 296 (1966)
5.15 B. RANKIN, S. SIEGAL: NBS Rept. 9537 (1967)
5.16 B. RANKIN, J. TAN: NBS Tech. Note 492 (1970)
5.17 R. M. THOMPSON: PhD Thesis, Indiana University (1972)
5.18 O. FUJIMURA, R. KAGAYA: "Structural Patterns of Chinese Characters", Intern. Conf. Computational Linguistics (1969); also: Ann. Bull., Res. Inst. Logopedics & Phoniatrics, University of Tokyo, No. 3 (1969); also: IOJC
5.19 O. FUJIMURA, R. KAGAYA: "A Pattern-Structural Code for Kanji", UJCC
5.20 O. FUJIMURA, A. ISHIDA: Computer Input-Output of Chinese Characters, The USA-Japan Cooperative Science Program Project No. 8 R-002 (1973)
5.21 M. FINLEY: "On the Formal Description of Chinese Characters", CCIO
5.22 M. FINLEY: "The Chinese Character Classification Problem", Proc. 2nd Open Conf. Inf. Sci. Canada, 163 (1974)
5.23 E. BURKART: PhD Thesis, University of Pennsylvania (1968)
5.24 T. SAKAI, M. NAGAO, H. TERAI: Inf. Proc. Japan **10**, 10 (1970)
5.25 T. SAKAI, S. SUIGITA, H. FUJITA: "Some Experiments on Kanji I/O Systems", IOJC
5.26 M. NAGAO: "Picture Recognition and Data Structure", In Graphic Languages, ed. by NAKE, ROSENFELD (North-Holland Publishing Co., Amsterdam 1972)
5.27 C. LEBAN, C. BAIRD: Computer Studies Hum. Verbal Behav. **2**, 133 (1969)
5.28 C. LEBAN: "Graphemic Synthesis: The Ultimate Solution to the Chinese Input/Output Problem", CCIO
5.29 F. LEE: Quart. Progr. Rept. Res. Lab. Electron., MIT **84**, 69 (1953)
5.30 S. CALDWDLL: J. Franklin Inst. **267**, 471 (1959)
5.31 W. HEAGERTY: "Ideographic Composing Machine", IOJC
5.32 F. SHASHOUA: RCA Tech. Paper, Camden, NJ (1964)
5.33 Anonymous: Machine Design **35**, 8 (1962)
5.34 A. ENG: Tech. Rept. RADC-TR-66-000 (Rome Air Development Center, Rome, NY 1966)
5.35 T. KIANG: "A Compatible Chinese Character Encoding System", Proc. 4th Ann. Princeton Conf. Inf. Sci. Systems (1970)
5.36 F. FANG, C. LIU, D. TANG: IBM Tech. Disclosure Bull. **13**, 3540 (1971)
5.37 H. TAO: MS Thesis, University of North Carolina (1966)
5.38 H. TAO: Comp. Stud. Human Verbal Behavior **2**, 1 (1969)
5.39 E. YHAP: IBM Tech. Rept. RC 4764 (1974)
5.40 E. YHAP: IBM J. Res. Develop. **19**, 60 (1975)
5.41 K. SU: Res. Rept. No. E 21-620-SU-1, School of Electrical Engineering, Georgia Institute of Technology (1972)
5.42 K. SU: "The Creation of a Set of Alphabets for the Chinese Written Language", CCIO
5.43 T. TING, C. HORNG: "Graphemic Analysis and Synthesis of Chinese Ideograms", Proc. 1975 Computer Sci. Conf., Washington (1975)
5.44 T. TING, C. HORNG: "Toward a Systematic Decomposition and Automatic Recomposition of Chinese Ideograms", Proc. Intern. Computer Symp., Taipei (1975)
5.45 S. DUNCAN, R. MUKAII, S. KUNO: Computer Stud. Human Verbal Behav. **2**, 113 (1969)
5.46 H. HAYASHI, S. DUNCAN, S. KUNO: CACM **11**, 613 (1968)
5.47 S. K. CHANG: IEEE Trans. SMC-**3**, 257 (1973)
5.48 S. K. CHANG, C. CHIU, M. YANG, B. LIN: "PEACE—A Phonetic Encoding and Chinese Editing System", CCIO
5.49 F. FANG, C. LIU, D. TANG: IBM Tech. Rept. RC4467 (1973); also CCIO
5.50 F. GRONER, J. HEAFNER, T. ROBINSON: IEEE Trans. EC-**16**, 856 (1967)

5.51 J. LIU: EE Thesis, MIT (1966)
5.52 D. S. PRERAU: PhD Thesis, MIT (1970)
5.53 O. ORE: Theory of Graphs (American Mathematical Society, Providence, RI 1962)
5.54 A. V. HERSHEY: Defense Documentation Center Rept. AD 622398 (1967)
5.55 Y. CHU: A Comparative Study of Language Reforms in China and Japan, Skidmore College Bulletin (1969)
5.56 F. SHIEN: RAND Tech. Rept. P-3078 (1965)
5.57 W. STALLINGS: Pattern Recognition **8**, 87 (1976)
5.58 R. CASEY, G. NAGY: IEEE Trans. EC-**15**, 91 (1966)
5.59 Y. NAKANO, K. NAKATA, Y. UCHIHURA, A. NAKAJIMO: "Improvement of Chinese Character Recognition Using Projection Profiles", PPR
5.60 K. NAKATA, S. NAKANO, M. YAMAMOTO, M. YASUDA, H. TERAI: "Problems in Chinese Character Recognition", UJCC
5.61 K. NAKATA, Y. NAKANO, Y. UCHIKURA: "Recognition of Chinese Characters", Proc. Conf. Machine Perception Patterns and Pictures (1972)
5.62 S. YAMAMOTO, A. NAKAJIMA, K. NAKATO: "Chinese Character Recognition by Hierarchical Pattern Matching", PPR
5.63 P. WANG, R. SHIAU: Pattern Recognition **5**, 303 (1973)
5.64 P. WANG, W. HODGKISS: Feature Extraction on a Finite Set of Binary Patterns", CCIO
5.65 P. WANG: "The Topological Analysis, Classification, and Encoding of Chinese Characters for Digital Computing Interfacing—Part I", CCIO
5.66 P. WANG: "The Topological Analysis, Classification and Encoding of Chinese Characters for Digital Computing Interfacine—Part II", PPR
5.67 L. ANKENEY: Defense Documentation Center Rept. AD 722852 (1971)
5.68 D. CROWN, T. SHIMAOKA: J. Police Sci. Admin. **2**, 279 (1974)
5.69 M. YOSHIDA, M. EDEN: "Handwritten Chinese Character Recognition by an Analysis-by-Synthesis Method", PPR
5.70 Y. MA: IEEE Trans. SMC-**4**, 223 (1974)
5.71 Y. MA: IEEE Trans. SMC-**4**, 575 (1974)
5.72 B. PERNICK, C. BARTOLOTTA, D. YUSTEIN: Appl. Opt. **6**, 1421 (1967)
5.73 C. LI, T. TEND, M. ZOBRAK: "On Recognition of Handwritting Chinese Characters", Proc. 1st Princeton Conf. Sci. Systems (1967)
5.74 M. ZOBRAK, T. SZE: "A Method of Recognition of Hand Drawn Line Patterns", Proc. 1st Princeton Conf. Info. Sci. Systems (1967)
5.75 M. ZOBRAK: MS Thesis, University of Pittsburgh (1966)
5.76 S. K. CHANG, D. LO: "An Experimental System for the Recognition of Handwritten Chinese Characters", CCIO
5.77 S. CHEN: ScD Thesis, Washington University, St. Louis (1973)
5.78 H. CHUANG, S. CHEN: "Computer Aided Instruction in Chinese Characters", CCIO
5.79 J. TOU, J. TSAY, J. YOO: "Interactive Processing of Chinese Characters and Texts", CCIO
5.80 C. CHENG: J. Chinese Language Teachers Assoc. **8** 75 (1973)

6. Shape Discrimination

TH. PAVLIDIS and H.-Y. F. FENG[1]

With 18 Figures

6.1 Basic Considerations

The problem of shape discrimination is a central one in pattern recognition. Many authors have discussed various aspects of it, but no satisfactory general theory exists [6.1–11]. Probably the major systematic effort towards shape theory is centered around the *medial axis transformation* or *skeletons* [6.1, 2, 6, 10].

In that approach a "full" figure is transformed into a line drawing consisting of those points which have a nonunique boundary neighbor. Figure 6.1 shows a few simple examples. The line drawing may be labeled with the distance of each of its points from the boundary and this allows the reconstruction of the original figure as an envelope of circles centered on the skeleton. The skeleton may be used to derive information about the shape of the original figure but except for some rather gross properties (e.g., elongation [6.2]) the process of doing so is by no means straightforward. The computation of the skeleton can be quite time consuming [6.2, 12] and very sensitive to noise as can be seen by comparing the first and last of the drawings in Fig. 6.1. Both difficulties may be reduced by performing first a *polygonal approximation* of the original contour. This can certainly remove noise and also allows a relatively fast computation of the skeleton [6.6]. However obtaining such an approximation can be quite sufficient in itself for shape description. Thus one can decide trivially whether the figure is a triangle, a square, or a rectangle.

In this chapter we develop a model for shape description based on polygonal approximations which seems to agree with various aspects of human intuition. It has also rather modest computational requirements (depending on the depth of analysis one desires) so that its implementation by computer programs is quite straightforward. We will not attempt to give here a complete theory of shape but will present only certain methodology which might be useful as a step towards such a general theory.

A central premise in this work is that shape perception is a hierarchical process and therefore it can be dealt appropriately with syntactic or, more generally, structural techniques [6.13].

If such an approach is to be followed we must define first what we shall use as the simplest forms of shape, namely the *shape primitives*. In earlier works [6.3, 9] it has been suggested to use convex sets as those of the simplest shape form. However this excludes such intuitively simple shapes as a minisque [6.8]. Here we will allow as primitives nonconvex sets provided that their boundary can be

---

[1] Now with the Pattern Analysis and Recognition Corporation, Rome, N.Y.

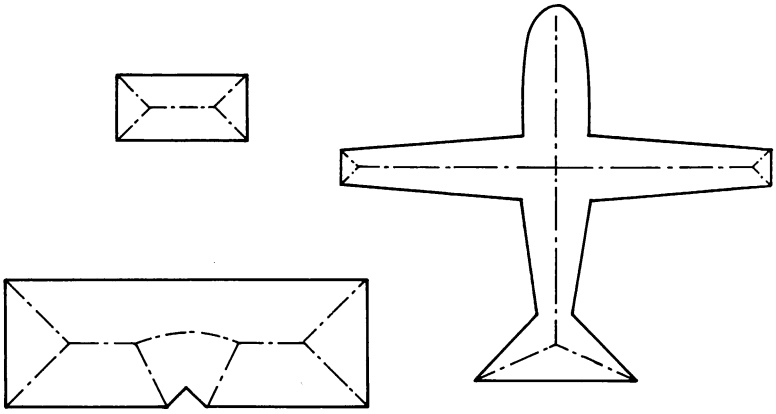


Fig. 6.1. Examples of the skeletons of certain simple shapes

divided into two contiguous arcs where the curvature does not change sign. A minisque has this property so it can now be admitted as a primitive element. The most complex figure satisfying this condition is obviously a winding spiral.

Given this assumption we are faced with two major tasks. One is the decomposition of complex shapes into simpler ones. The other is the further analysis of the primitive shapes. Both tasks can be facilitated if we limit our attention to polygons. This may seem a serious restriction but it is not actually so. Indeed any plane contour can be approximated by a polygon with as small an error as we want. Such an approximation can also result in noise removal. There exist a number of papers on this subject and certain very efficient schemes have been proposed recently [6.14, 15]. A polygonal approximation is bound to detect any "sharp corners" in the figure, and measures of fit can be used to detect whether a given corner reflects the original shape or approximates a smooth high-order curve.

In this chapter we shall discuss briefly the description of contours in terms of polygons and the description of the shape of polygons. We shall then dwell in detail on the hierarchical decomposition of polygons.

6.2. Description of Contours in Terms of Polygons

We assume that the use of an approximation algorithm has resulted in the following information being available:

1) A sequence of vertices $(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)$ described by their discrete coordinates.

2) For each vertex a measure of corner quality Z_i defined as following: Let L_i denote the length of the side of the polygon between vertices $i-1$ and i. Let e_i denote the pointwise error of approximation at the i vertex and d_i the absolute difference in slopes of the sides of the polygon at that vertex. Then

$$Z_i = 1 - (e_i)/[\max(L_i, L_{i+1})d_i] .$$

It is easy to verify that if the original curve is a parabola then the above measure equals always 1/3. It the original curve is a polygon then it is equal to 1 [6.16].

The latter information can be used to replace parts of the polygon by higher order curves. It is now possible to obtain a generalized chain encoding [6.17] of the boundary as a sequence of lines and higher order curves. This can also be considered as a generalization of the approach used by LEDLEY and LEE and FU [6.18, 19]. In this treatise we will not consider this point any more. Instead we shall concentrate on the use of the polygonal approximation only.

6.3 Description of the Shape of Polygons

The sequence of vertices can be used to provide information about the type of the angles of the polygon: convex or concave. We define a concave arc to be a sequence of concave vertices uninterrupted by convex vertices. If a is used to denote a convex corner and b a concave corner then the *angular characteristic* of the polygon P_n is defined as

$$\mathrm{AC}(P_n) = x_1 x_2 \ldots x_n$$

where x_i equals a or b. Without loss of generality we may assume that $x_1 = a$ always. Then a sequence of b's, a *b-string*, will correspond to a concave arc. A convex polygon has no b-strings while polygons with exactly one b-string (concave arc) are shown in Fig. 6.2. We shall call such polygons *spirals*.

In the next section we will describe ways of decomposing a polygon into primitive shapes. In this section we will show first how a certain amount of shape information can be elicited without decomposition. In many applications one deals with a limited variety of shapes each characterized by significant difference from the others. Alphanumeric characters are a prime example of this case, especially if one considers separately letters and numerals [6.20]. In such cases significant progress towards pattern recognition can be made by concentrating on salient features of the shape of characters in each class.

In the case of numerals the number of concave arcs is in itself a significant feature. In addition the orientation of written symbols is fixed, at least within

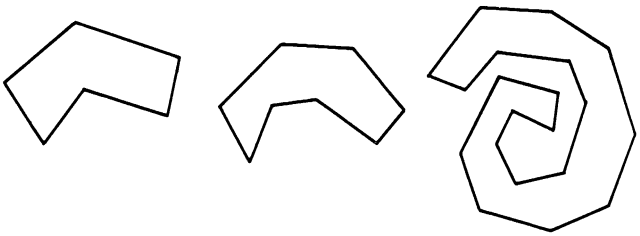


Fig. 6.2. Examples of spirals which are considered as primitive sets for the first decomposition. Allowing them to be nonconvex could be useful in the analysis of cursive script

Table 6.1. Numerals described in terms of concave arcs in their boundary

Total concave arcs	Left concave arcs				
	0	1	2	3	4
0	0,1	—	—	—	—
1	6	1, 7, 9	—	—	—
2	?	2, 4, 5, 8	3	—	—
3	?	?	3	?	—
4	?	4	?	?	?

certain limits. (This is not the case with, say, chromosomes). Thus we may distinguish concave arcs in the left or right part of a character. It is then easy to verify the relations in the matrix shown in Table 6.1.

This matrix is not yet sufficient for numeral classification, but one can easily build additional features on the basis of the same philosophy. For example the number of holes in the original character may be a feature. The following is a set of such features: $u(1)$ = number of holes; $u(2)$ = number of concave arcs; $u(3)$ = number of concave arcs on the left; $u(4)$, $u(5)$, $u(6)$ = location of first, second and third concave arcs; $u(7)$ = true (logical) if there is an upward-facing concave arc; $u(8)$ = true (logical) if the polygon has a side near the vertical before its first concave arc when scanned from the top, etc.

Similar features were used in tests with the IEEE Data Base 1.2.2 (source: J.H. Munson, Stanford Research Institute). A recognition rate of 90.6% was achieved [6.21]. The reader may find more details of this method in the paper cited.

6.4. Fundamentals of Decomposition

The main disadvantage of boundary encodings is that points which are geometrically close together can be encoded quite far apart in the string. In the example of Fig. 6.3 there Fig. 6.3 is no simple way to describe the neck *AB* through the boundary string. This difficulty may be overcome by considering the relative location of the concave arcs in the perimeter of a polygon. In particular we might be able to draw a line connecting two concave vertices (e.g., AB in the present example) and *decompose* the polygon into two parts using it as a dividing line.

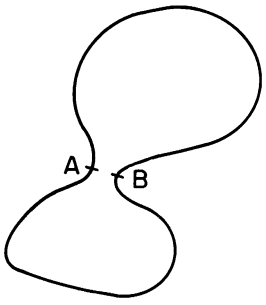


Fig. 6.3. Illustration of a defect of boundary encodings: Points A and B which geometrically close have distant encodings

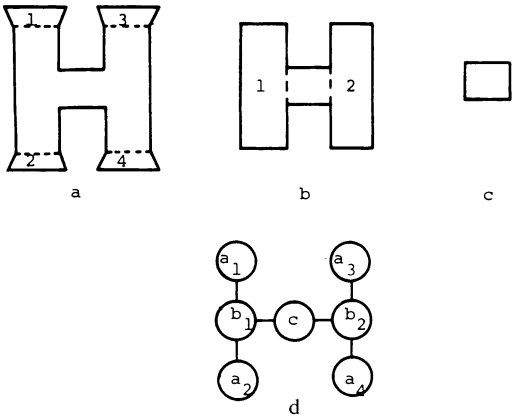


Fig. 6.4a—d. Illustration of the decomposition of the roman letter H

We may keep track of adjacency relations during such decomposition creating at the end a component adjacency graph.

We shall describe formally a decomposition scheme based on this idea starting with simply connected polygons (i.e., without holes). Figures with holes will be discussed later. The scheme is recursive and the term "nondecomposable element" will refer to a subset which is of simple form and will not be processed any more at a given step. It may well be analyzed further in subsequent steps. The term "*baseline*" will refer to the line dividing two components.

Figure 6.4 illustrates this approach applied to the polygonal approximation of a capital H. (a) is the input, (b) and (c) are intermediate and final results and (d) the adjacency graph. Note that the graph is generated recursively and therefore it is quite amenable for syntactic description. The dotted lines in the figure are the baselines.

The motivation for the following definition is obvious.

Definition 1: A simply connected polygon P_n is *decomposable* if its angular characteristic $AC(P_n)$ contains at least two b-strings.

This definition implies that if a polygon is convex or a spiral (see Sec. 6.1) then it is nondecomposable. The computational effort inherent in this method is given by the following proposition.

Proposition 1: In order to determine whether a subpolygon is decomposable one must check at most four vertices.

The proof of this proposition can be found elsewhere [6.22,23].

In general the decomposition will not be unique because there may be more than one legal dividing line between the vertices of two successive concave arcs. This ambiguity may be removed by choosing the shortest among such lines. However one more source of nonuniqueness remains if we remove subpolygons as they are found. For example the polygon of Fig. 6.5 can have either of the following three representations:

(ABCDEFJA) and (JFGHJ)

(ABCFGHJA) and (CDEFC)

(ABCJA) and (HJCDEFGH).

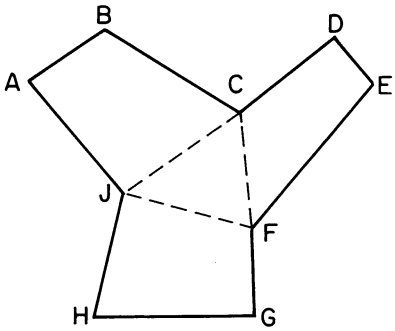


Fig. 6.5. Illustration of the nonuniqueness of sequential decomposition

This ambiguity may be removed if all legal dividing lines are found and then one proceeds with the decomposition in an order independent criterion, for example in order of the length of the dividing lines starting with the shortest. It is easy to verify on the basis of elementary geometrical considerations that no legal dividing lines ever cross each other and therefore such a procedure will not encouter any impasses. These concepts are implemented in the following algorithm.

Main Decomposition Algorithm

Input: ordered list of the corners of simply connected polygon P.

Output: Description of nondecomposable subpolygons and their adjacency relations.

Arrays: $i(*)$ starting vertex of concave arcs, $j(*)$ last vertex of concave arcs, $L(x, y)$ length of line joining vertices x and y, $I(X)$ vertex baseline between X^{th} and $X+1^{\text{th}}$ starts, $J(X)$ vertex where previous baseline ends, $M(X)$ length of previous baseline.

Functions: INSIDE(x, y) returns true if line x, y lies inside the polygon, INTERSECT(x, y) returns true if line (x, y) intersects the polygon.

Step 1: Find angular characteristic of P, rearrange (if necessary) vertices so that the first one is convex and fill arrays $i(*)$ and $j(*)$.

Step 2: Do block 21 while $m>1$.

Begin block 21;

1) For $k=1$ to m do block 22;
   Begin block 22;
   a) Set $I(k)=0$, $J(k)=0$, $M(k)=$maximum;
   b) For $x=i(k)$ to $j(k)$ do block 23;
      Begin block 23; For $y=i(k+1)$ to $j(k+1)$ do block 24;
         Begin block 24;
         If

$$\{L(x, y)<M(k)\}\&\{\text{INSIDE}(x, y)\}\&\{\text{INTERSECT}(x, y)\}$$

then

$$I(k)=x\,, \qquad J(k)=y\,,$$
$$M(k)L(x, y)$$

End blocks 24, 23, 22;

2) Sort $M(k)$ in increasing order and let $r(*)$ be the permutation mapping the new ordering onto the original.
3) Set $a=0$.
4) For $k=1$ to m do block 41 while $I(r(k))>0$.
   Begin block 41;
   a) Set $a=a+1$, $b=I(r(k))$, $c=J(r(k))$.
   b) Define the polygon $P(a)$ by the vertices

$$b, b+1, \ldots c-1, c.$$

   c) Establish the adjacency of $P(a)$ and P.
   d) Modify P by replacing vertices $b+1, b+2, \ldots c-1$ by the linear segment joining b and c.
   End block 41;
5) Inspect all vertices of P and redefine its angular characteristic.

End block 21;

This algorithm is guaranteed to terminate by the following result from elementary geometry (see also [6.22, 23]).

Proposition 2: If a polygon has at least two concave angles which do not share any sides then there exists a pair of such angles which can be joined by a legal dividing line.

In this way we know that as long as m exceeds 1 at least one decomposition will be performed and therefore the number of vertices will decrease by at least one. Because their initial number is finite the algorithm will eventually terminate.

6.5 Further Decomposition and Shape Description

In certain cases it might be necessary to further decompose a spiral into convex subsets. This can be performed easily by the following algorithm.

Decomposition of Spirals into Covex Subsets

Input: Ordered list of vertices of spiral S.
Output: Description of convex subsets of the spiral.
Step 1: For each concave corner find the bisectrix.
Step 2: For each bisectrix find the first point of intersection with the boundary of the polygon.
Step 3: Use the segements of the bisectrices lying inside the polygon as dividing lines.

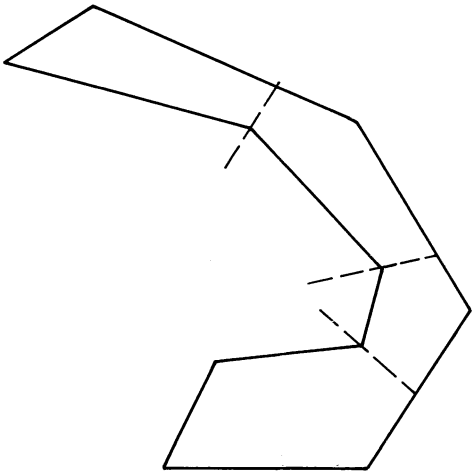


Fig. 6.6. Decomposition of a spiral

Figure 6.6 shows the dividing lines used by this algorithm. It can be easily shown that no two bisectrices intersect inside the polygon and therefore the above decomposition is well defined. It is also obvious that the resulting polygons are convex. Therefore the description of the shape of a spiral can be reduced to that of a sequence of convex sets together with the sequence of the sizes of the original concave angles. Of course subsets of this description may be used in the same manner as in Section 6.3.

The description of the shape of convex polygons is much more tractable than the general shape description problem. For one thing one may define for them concepts like width, elongation, etc. [6.2, 24–26]. For another it is possible to produce some rather simple grammars which will generate exactly all convex polygons [6.27].

6.6 Decomposition of Polygons with Holes

Up to this point we have ignored the case of polygons with holes. In this section we show how such polygons can be decomposed. Three general approaches are possible.

Approach 1

i) For each polygon which is the boundary of a hole find a vertex which is closest to one of the external boundaries. Then the pair of these vertices is joined by a dividing line and two additional "pseudo-vertices" are created as shown in Fig. 6.7. This step will yield a simply connected polygon.

ii) Apply the main decomposition algorithm.

iii) Establish the adjacency relation among all pairs of components which share a pair of pseudo-vertices (A–B and B–C in Fig. 6.7).

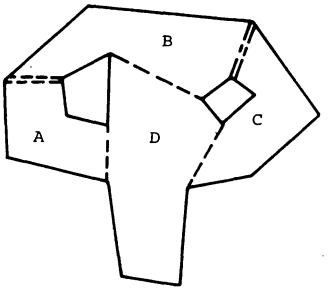


Fig. 6.7. Decomposition of a polygon with holes by Approach 1. "= = = =" are the pseudo-boundaries

Approach 2

i) Match pairs of vertices on different boundaries if their distance is below a threshold. Join such vertices by dividing lines obtaining, *possibly*, a simply connected polygon.

ii) As in Approach 1.

iii) As in Approach 1.

This method works when the picture is composed primarily of strokes because in such a case it is easy to find pairs of vertices which are close to each other. For example the Chinese character "the eye" can be decomposed as shown in Fig. 6.8.

Approach 3

i) Proceed with the main decomposition algorithm as though there were no holes.

ii) Describe the holes as independent simply connected regions.

iii) Generate positional relations between holes and components.

This approach is applicable when the holes are expected to fall within primitive components and do not play a major part in the description of the shape except possibly by their occurrence.

6.7 Implementation of the Decomposition Algorithm

The main decomposition algorithm and Approach 1 have been implemented in FORTRAN IV and run on an IBM 360-91 computer at Princeton University. They have been tested on the handwritten numerals described in Section 6.3 and on pictures of chromosomes and Chinese characters digitized at our laboratory.

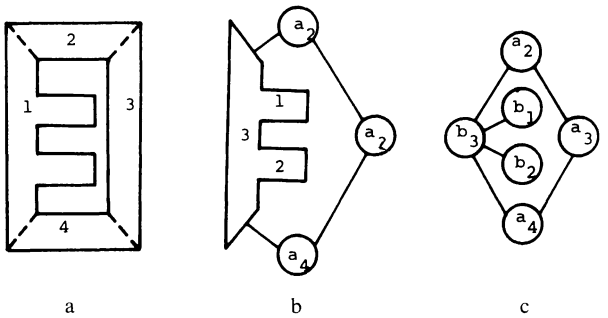


Fig. 6.8 a–c. Decomposition of a polygon with holes by Approach 2

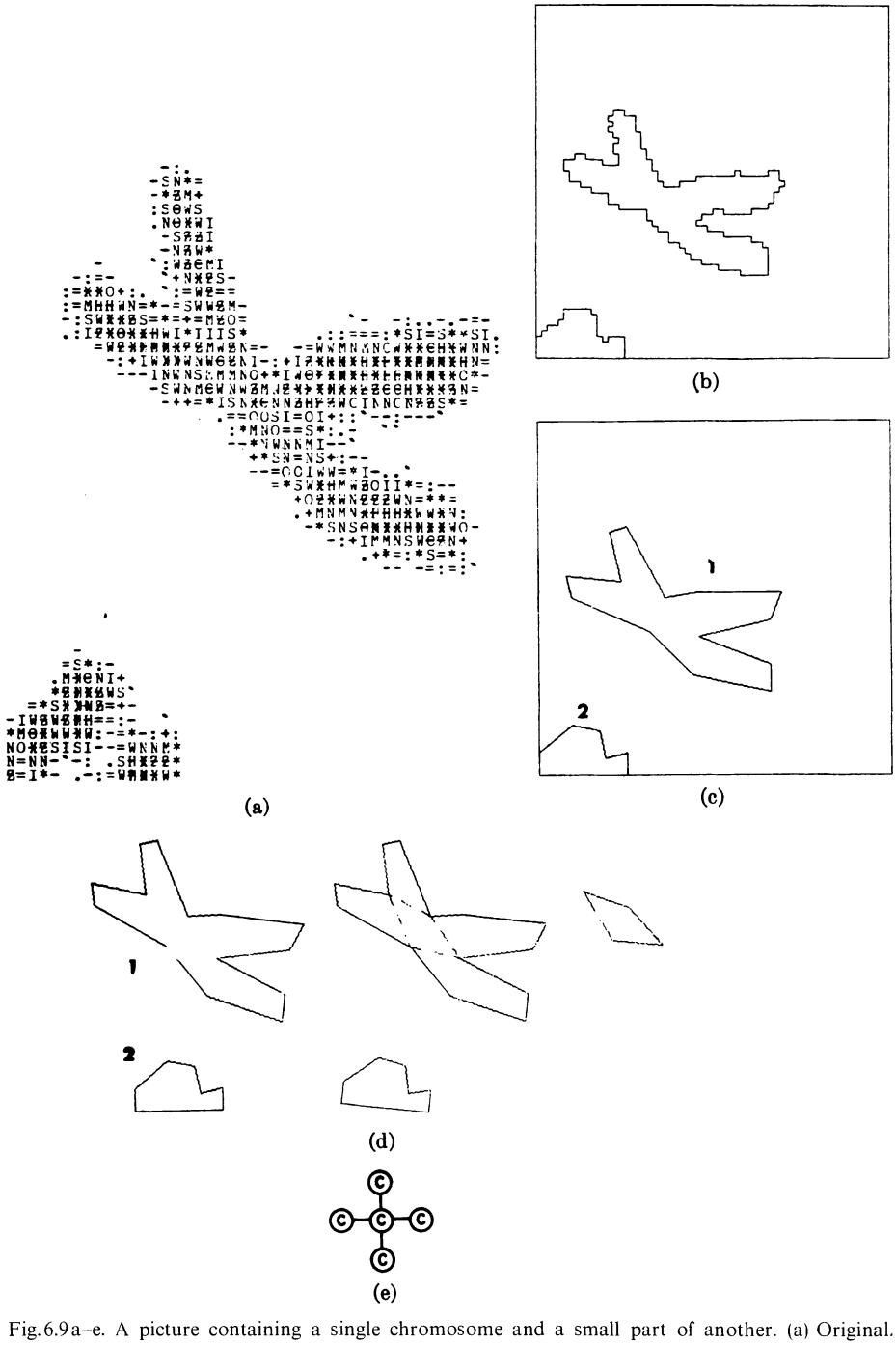


Fig. 6.9a–e. A picture containing a single chromosome and a small part of another. (a) Original. (b) Contours. (c) Polygonal approximation. (d) Decomposition (baselines are shown as dotted lines). (e) A labeled graph summarizing this description. The relative location of the nodes on the plane is important

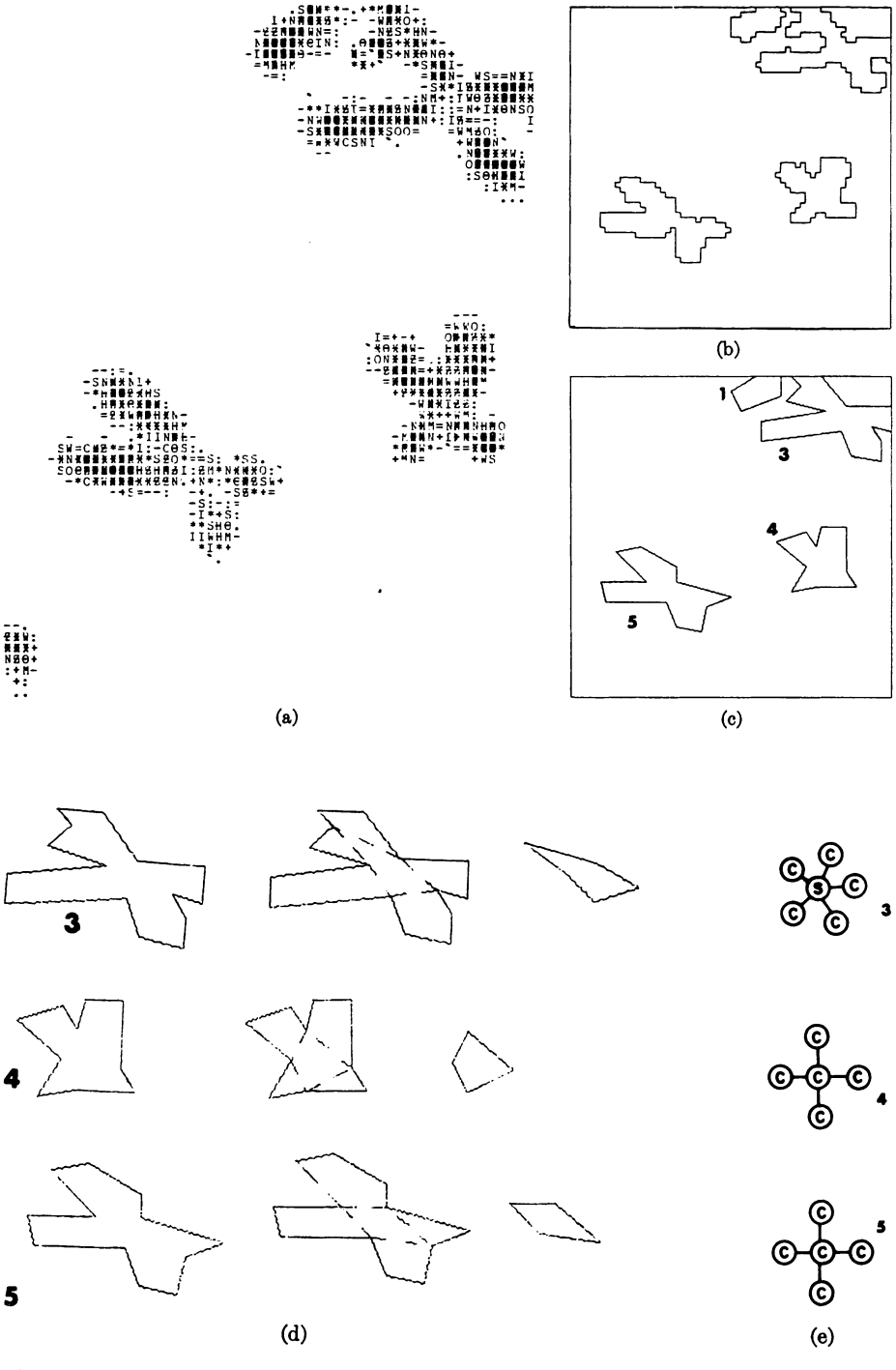


Fig. 6.10a–e. Part of a picture containing a group of chromosomes. Decomposition was performed on three of them indicated by the numbers on (c) and (f)

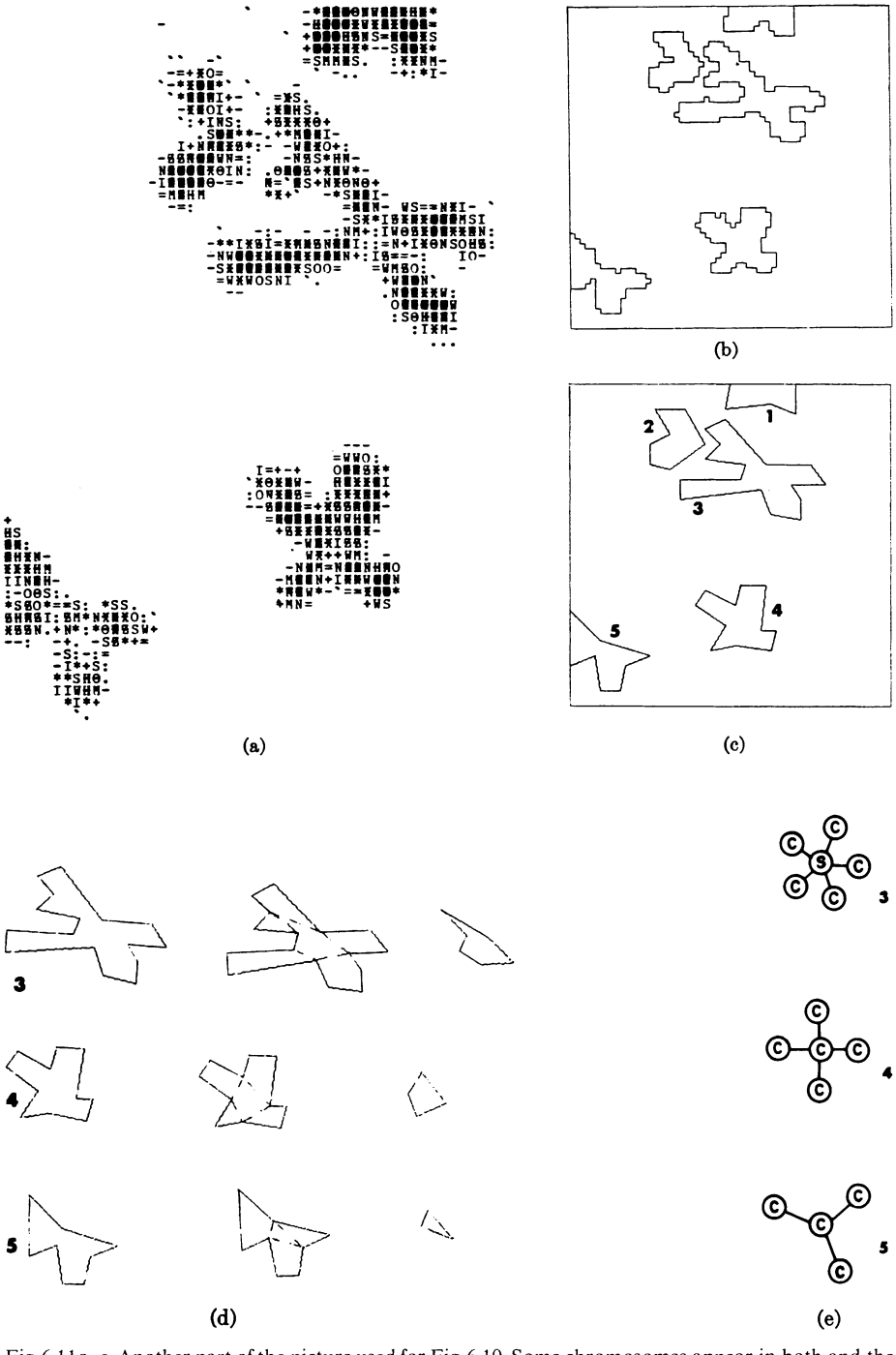


Fig. 6.11a–e. Another part of the picture used for Fig. 6.10. Some chromosomes appear in both and the repeatability of the decompositions illustrates the robustness of the method. Even when parts are missing (no. 5) the two forms are compatible

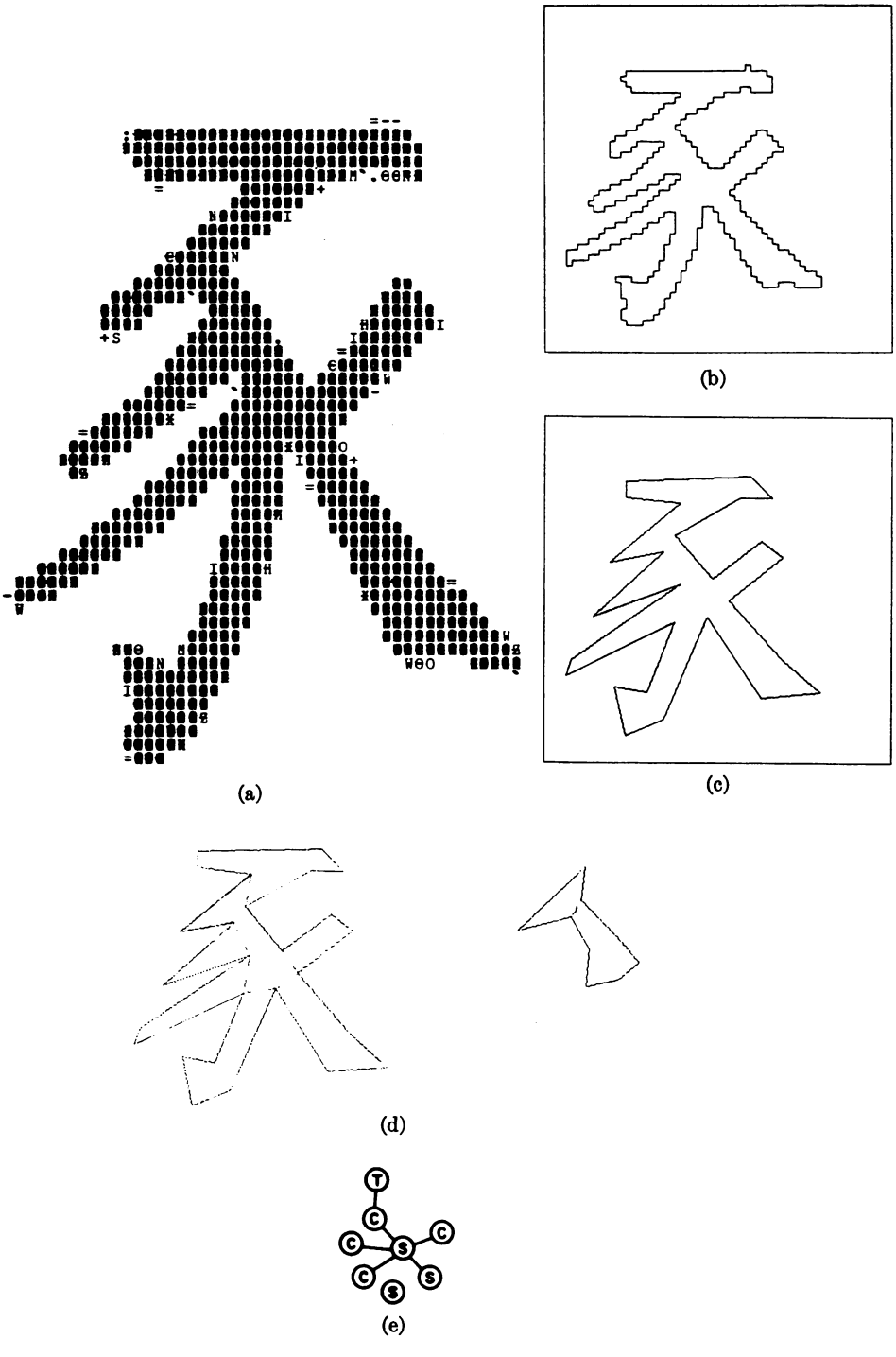


Fig. 6.12a–e. Analysis of the Chinese character for Pig

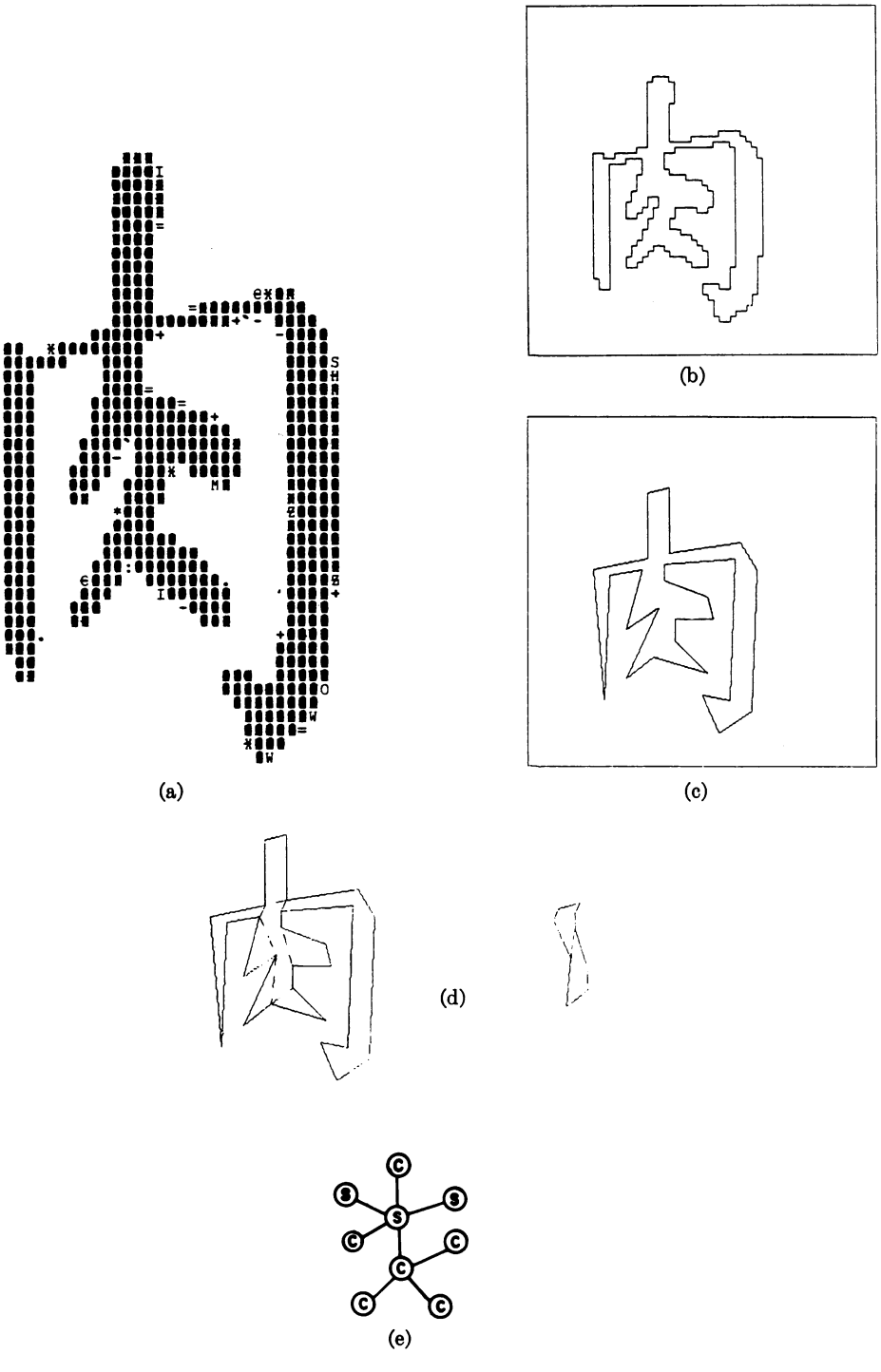


Fig. 6.13a–e. Analysis of the Chinese character for Meat

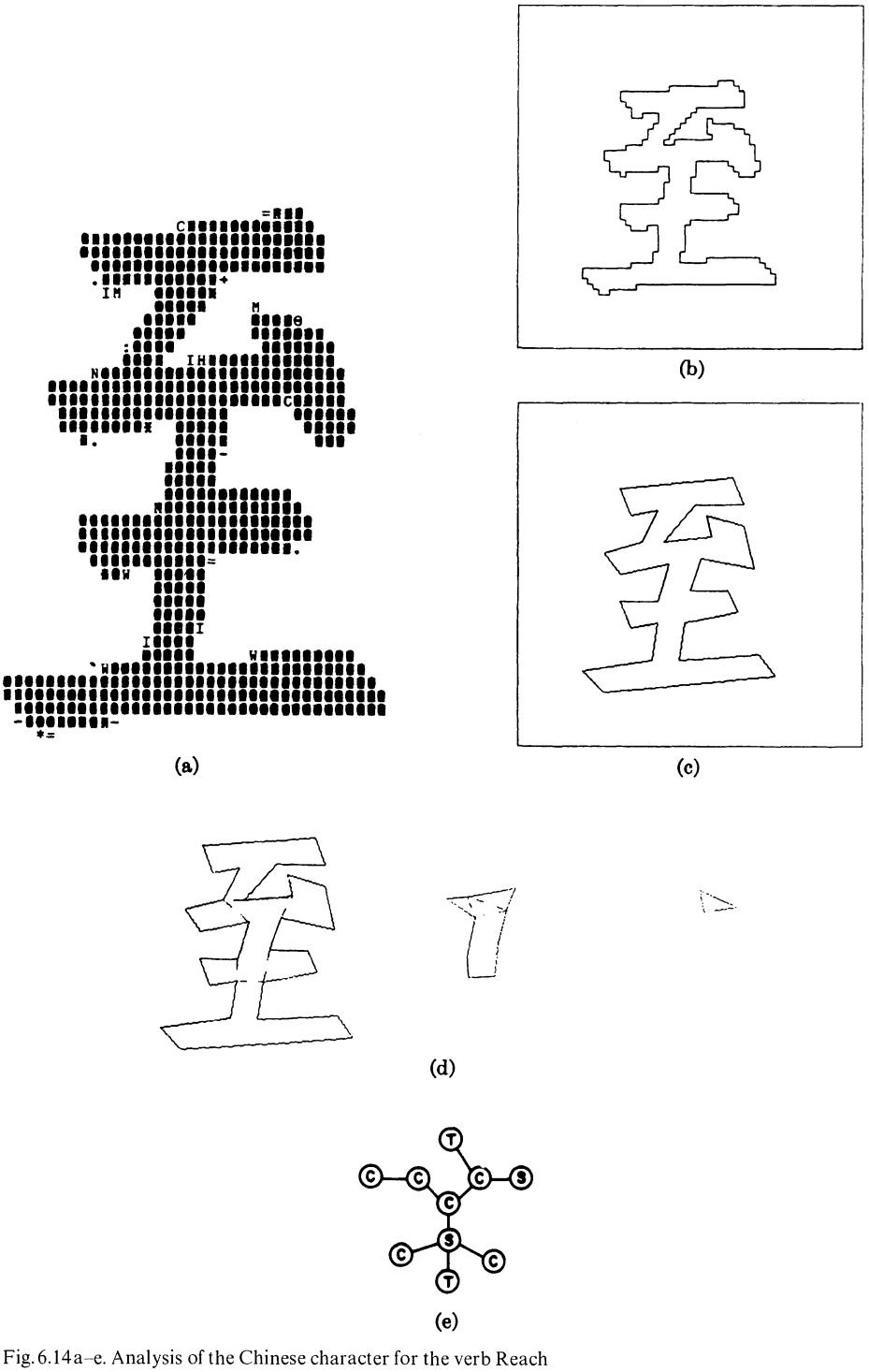


Fig. 6.14a–e. Analysis of the Chinese character for the verb Reach

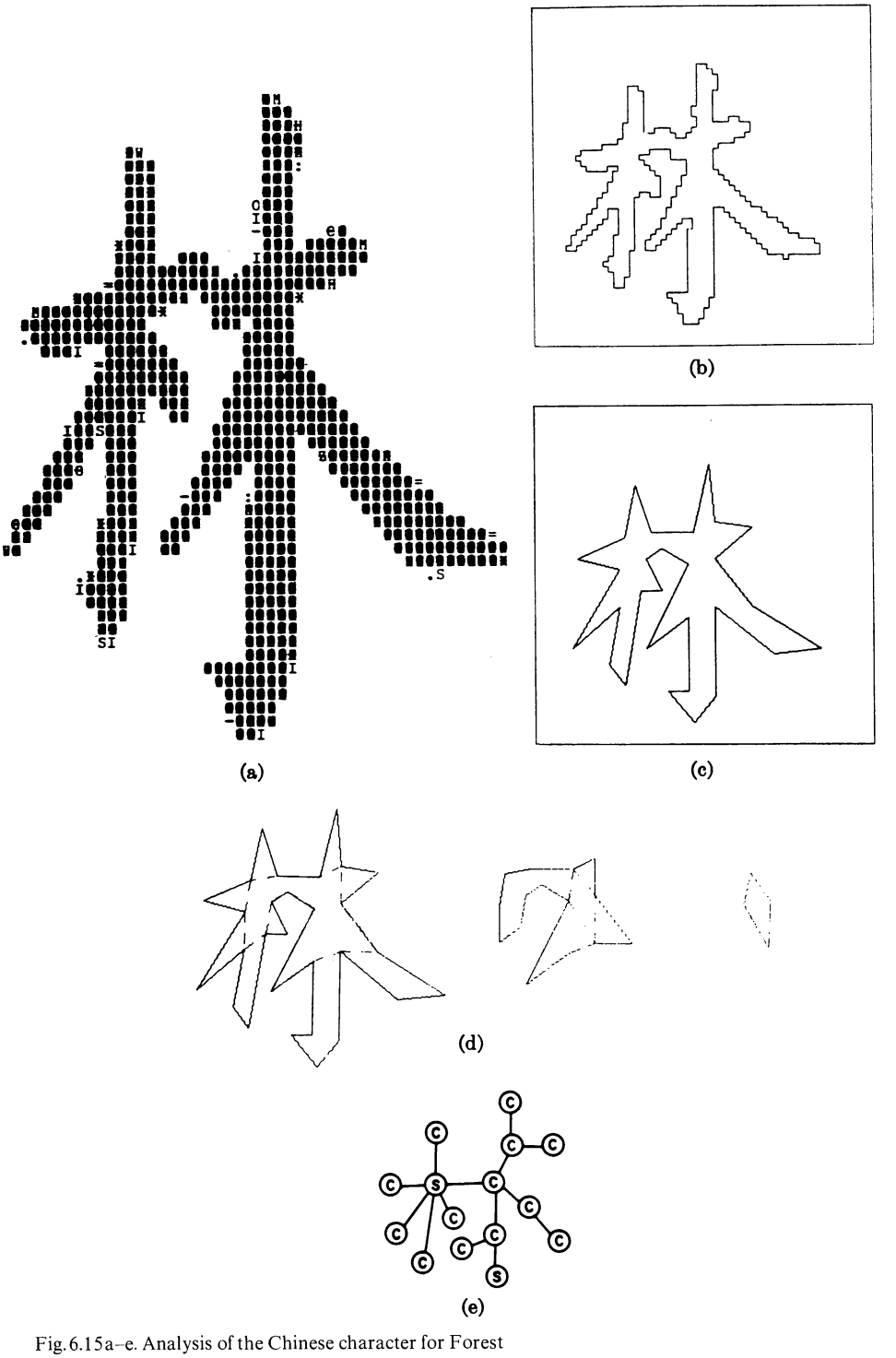


Fig. 6.15a–e. Analysis of the Chinese character for Forest

The system used consists of a closed-circuit TV camera connected to a minicomputer (HP2116) via a bandwidth compressor (by Colorado Video Associates) and an A/D converter. The pictures are stored on magnetic tape and then processed by the IBM 360-91. In the present tests a 64×64 grid was used with 7 bits per pixel.

The results of the processing are shown in Figs. 6.9–15 in the following manner: (a) is the original picture presented via an overprinting routine [6.28]. (b), (c), and (d) are CALCOMP plots showing the boundaries obtained by thresholding, the polygonal approximations obtained by a split-and-merge algorithm [6.14] and the results of the application of the main decomposition algorithm. (e) show the resulting adjacency graphs with the following notation: C stands for a convex set, S for a spiral and T for a T-type set. The latter are polygons with two concave arcs which have been "excused" from further decomposition. Figure 6.16 shows a typical computer printout describing the results of the decomposition. It corresponds to the example shown in Fig. 6.15.

Figures 6.17 and 6.18 show the application of these algorithms to the handwritten numerals used in Section 6.3. They are all CALCOMP plots showing the polygonal approximation with the vertices sequentially numbered. Figure 6.17

```
THE RESULT FOR OBJECT    1

  THE ORIGINAL CORNER LOCATIONS(IEVEL 0)

      1. ( 25, 31)  2. ( 25, 23)  3. ( 16, 20)  4. ( 26, 18)
      5. ( 30,  9)  6. ( 32, 17)  7. ( 47,  8)  8. ( 39, 17)
      9. ( 50, 15) 10. ( 54, 18) 11. ( 36, 21) 12. ( 36, 25)
     13. ( 30, 22) 14. ( 28, 25) 15. ( 31, 30) 16. ( 47, 22)
     17. ( 41, 30) 18. ( 55, 30) 19. ( 55, 26) 20. ( 61, 31)
     21. ( 56, 35) 22. ( 39, 35) 23. ( 48, 46) 24. ( 47, 55)
     25. ( 39, 42) 26. ( 30, 35) 27. ( 24, 42) 28. ( 23, 35)
     29. ( 12, 34)

LEVEL   1

  A CONVEX SUBPART BETWEEN    ( 23, 35) ( 25, 31)
  A CONVEX SUBPART BETWEEN    ( 25, 21) ( 26, 18)
  A CONVEX SUBPART BETWEEN    ( 26, 18) ( 32, 17)
  A CONVEX SUBPART BETWEEN    ( 32, 17) ( 39, 17)
  A CONVEX SUBPART BETWEEN    ( 39, 17) ( 36, 21)
  A CONVEX SUBPART BETWEEN    ( 36, 21) ( 30, 22)
  A SPIRAL SUBPART BETWEEN    ( 41, 30) ( 39, 35)
  A CONVEX SUBPART BETWEEN    ( 39, 35) ( 39, 42)
  A CONVEX SUBPART BETWEEN    ( 30, 35) ( 23, 35)
  THE REMAINING CORNERS IN THE CENTRAL SUBPOLYGON
      1. ( 25, 31)  2. ( 25, 23)  3. ( 26, 18)  4. ( 32, 17)
      5. ( 39, 17)  6. ( 36, 21)  7. ( 30, 22)  8. ( 28, 25)
      9. ( 31, 30) 10. ( 47, 22) 11. ( 41, 30) 12. ( 39, 35)
     13. ( 39, 42) 14. ( 30, 35) 15. ( 23, 35)

LEVEL   2

  A CONVEX SUBPART BETWEEN    ( 30, 35) ( 25, 31)
  A SPIRAL SUBPART BETWEEN    ( 25, 31) ( 31, 30)
  A SPIRAL SUBPART BETWEEN    ( 31, 30) ( 39, 35)
  A CONVEX SUBPART BETWEEN    ( 39, 35) ( 30, 35)
  THE REMAINING CORNERS IN THE CENTRAL SUBPOLYGON
      1. ( 25, 31)  2. ( 31, 30)  3. ( 39, 35)  4. ( 30, 35)

LEVEL   3

  A CONVEX SUBPART BETWEEN    ( 25, 31) ( 30, 35)
```

Fig. 6.16. Computer printout describing the analysis of Fig. 6.15

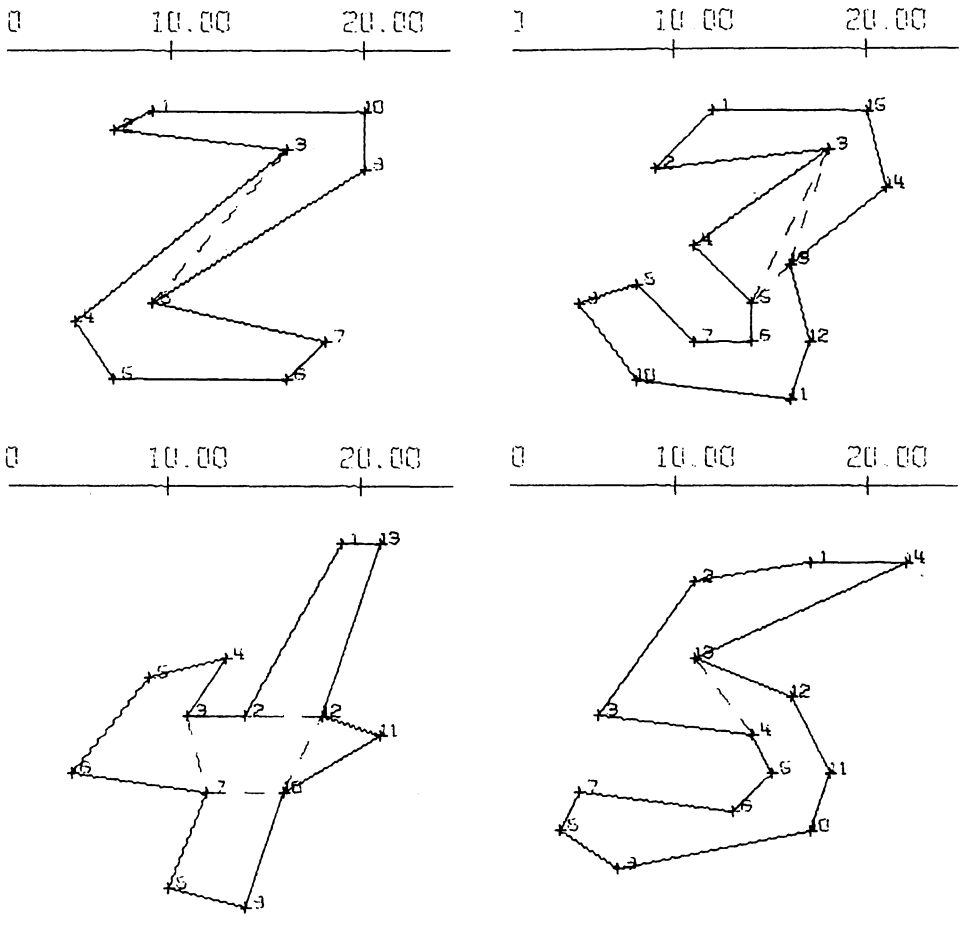


Fig. 6.17. Examples of decomposition of certain simply connected numerals

shows four simply connected polygons with the dividing lines indicated by broken lines. Figure 6.18 shows two examples of multiply connected polygons corresponding to the numerals 6 and 8. The heavy continuous lines indicate the dividing lines obtained by Approach 1. It is worthwhile noticing the resulting renumbering of the vertices. Due to a minor "bug" in the plotting program the numbers for the "split" vertices are plotted one on top of the other and some sophisticated human pattern recognition is needed to tell them apart. (Of course the internal representation is the correct one). These examples have been the result of work by L. P. CHANG who is currently working on algorithms for numeral recognition on the basis of such decompositions.

It is obvious that the decomposition algorithms are able to analyze quite complex shapes although their results do not always correspond to human intuition. For example the numeral 2 (in Fig. 6.17) is decomposed into two "boomerang" type components rather then three strokes. This can be rectified during spiral decomposition into convex sets when the possibility of merging such sets is

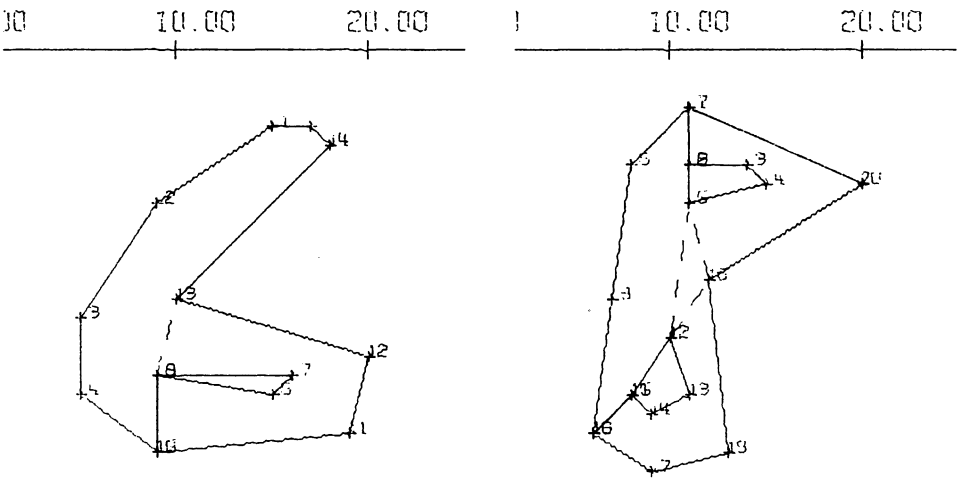


Fig. 6.18. Examples of decomposition of certain multiply connected numerals

examined. Such work is in progress. It is also possible to see the relation between these features and those derived on the basis of empirical considerations for all types of data considered-chromosomes [6.18, 19, 29–32], Chinese characters (Chapt. 8, this volume [6.33–38]) and handwritten numerals [6.39–43]. See also the literature on the general subject of feature selection for syntactic pattern recognition [6.13, 44].

By this time the reader may have been persuaded about the merits of this methodology for shape description but he still may question its computational requirements. It turns out that these are not as large as one may think at first. Our programs were written without a major effort in optimizing their speed, and they used the scratch disk extensively. This was necessitated by the need to organize various parts of the program as different steps under the IBM Job Control Language so that software written be different people could be easily interfaced. Thus most of our computer time was spent on input-output operations. In spite of this problem the time required for the total processing of the 64×64 pictures was always under 5 s. The thresholding required about 0.60 s while the decomposition required always about 0.40 s. Most of the time was consumed by the boundary tracing and particularly the subsequent polygonal approximation. Depending on the picture this varied from 1.5 to 3.7 s.

A number of remarks are pertinent in this context. It is well known that significant speedups may be obtained by using assembly language rather than FORTRAN programs. Since the major part of the computation involves the polygonal approximation, additional speedups could be obtained by using special-purpose hardware for that step. Thus a total speedup factor of 50 is feasible reducing the time required for processing to 0.10 s [6.45].

Probably the best way to obtain an idea about the speed of the decomposition is to notice that it requires about $^2/_3$ of the time needed for thresholding. This is not too surprising because the decomposition step deals with a small set of data; about 15 to 30 vertices.

6.8 Discussion of the Methodology

The results of the application of this method have a number of desirable features.

1) They are translation and rotation invariant and insensitive to registration as shown by Figs. 6.10 and 6.11. Rotation invariance can be controlled through the description of the juxtaposition relations in the final graph. The insensitivity to registration is important for many practical applications including optical page readers, mail sorters, cell counters, etc.

2) To a large extent they are size invariant. Problems may occur only when some of the objects in a picture are so small as to be of the same order of magnitude as what is considered noise for others. This will manifest itself only during the polygonal approximations.

3) They produce usually "anthropomorphic" descriptions and therefore they can be quite useful for feature extraction.

4) They produce data structures which are particularly appropriate for syntactic or structural pattern recognition which is natural since the methodology of decomposition is itself structural.

The only disadvantage of the methodology is that the programs implementing it tend to be quite complex. Such complexity does not imply slow processing but it may impose certain difficulties during research and development. It is our opinion that this is an unavoidable problem with any reasonable shape description scheme. After all we are trying to imitate a very complex mechanism, the human visual and perceptual processes.

Acknowledgements. The work described in this chapter has been supported by a grant from the Engineering Division of the National Science Foundation to T.P. (ENG 72-04133). We would like to thank L. P. CHANG for allowing us to use his examples from the handwritten numeral analysis program. We also thank the Institute of Electrical and Electronic Engineers for allowing us to reprint Figs. 6.2, 6.4 and 6.7–6.16 which appeared first in a paper published in the IEEE Trans. C-24, 636–650 (1975).

References

6.1 H. BLUM: A Transformation for Extracting New Descriptions of Shape, Symp. Models for the Perception of Speech and Visual Form (MIT Press 1964)
6.2 A. ROSENFELD, J. L. PFALTZ: J. Assoc. Computing Machinery **13**, 471–494 (1966)
6.3 T. PAVLIDIS: Pattern Recognition **1**, 165–178 (1968)
6.4 A. ROSENFELD: Picture Processing by Computer (Academic Press 1969)
6.5 M. B. CLOWES: Machine Intelligence **4**, 361–383 (1969)
6.6 U. MONTANARI: J. Assoc. Computing Machinery **16**, 534–549 (1969)
6.7 U. GRENANDER: in Advances in Computer **10** (Academic Press 1970), pp. 175–216
6.8 D. J. LANDRIDGE: in Frontiers of Pattern Recognition, ed. by S. WANTANABE (Academic Press 1972), pp. 347–365
6.9 T. PAVLIDIS: in Frontiers of Pattern Recognition, ed. by S. WATANABE (Academic Press 1972), pp. 421–451
6.10 H. BLUM: J. Theor. Biol. **XX**, 205–287 (1973)
6.11 L. S. DAVIES: Tech. Rept. TR-376, University of Maryland, Computer Science Department (1975)

6.12 O. PHILBRICK: Rept. 288, Air Force Cambridge Research Laboratories (1966)
6.13 K. S. FU: Syntactic Methods in Pattern Recognition (Academic Press 1974)
6.14 T. PAVLIDIS, S. L. HOROWITZ: IEEE Trans. C-**23**, 860–870 (1974)
6.15 H. Y. FENG, T. PAVLIDIS: IEEE Trns. CAS-**22**, 427–439 (1975)
6.16 T. PAVLIDIS: in Proc. Conf. Computer Graphics, Pattern Recognition and Data Structure, Los Angeles, CA (1975), pp. 215–219
6.17 H. FREEMAN: IEEE Trans. EC-**10**, 260–268 (1961)
6.18 R. S. LEDLEY: Science **146**, 216–223 (1964)
6.19 H. C. LEE, K. S. FU: IEEE Trans. C-**21**, 660–666 (1972)
6.20 W. S. ROSENBAUM, J. J. HILLIARD: IBM J. Res. Dev. **19**, 398–421 (1975)
6.21 T. PAVLIDIS, F. ALI: IEEE Trans. SMC-**5**, 610–614 (1975)
6.22 H. Y. F. FENG: PhD thesis, Princeton University, Dept. of Electrical Engineering (1974)
6.23 H. Y. FENG, T. PAVLIDIS: IEEE Trans. C-**24**, 636–650 (1975)
6.24 G. EWALD, G. C. SHEPHARD: Math. Zeitschr. **91**, 1–19 (1966)
6.25 D. E. MCCLURE: in Proc. 8th Princeton Conf. Information Sciences and Systems (1974), pp. 244–247
6.26 D. E. MCCLURE, R. A. VITALE: J. Math. Anal. Appl. **51**, 326–358 (1975)
6.27 T. PAVLIDIS: in Graphic Languages, ed. by F. NAKE, A. ROSENFELD (North-Holland Publishing Co. 1972), pp. 210–224
6.28 P. HENDERSON, S. TANIMOTO: Computer Graphics Image Processing **3**, 327–335 (1974)
6.29 G. GALLUS: in Applicazioni Bio-Mediche del Calcolo Elettronico (1968), pp. 95–108
6.30 D. RUTOVITZ: Machine Intelligence **5**, 435–462 (1970)
6.31 A. A. KLINGER, A. KOCHMAN, N. ALEXANDRIDIS: IEEE Trans. C-**20**, 1014–1022 (1971)
6.32 M. AIELLO, C. LAMI, U. MONTANARI: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 205–219
6.33 L. D. HARMON: Proc. IEEE **60**, 1165–1176 (1972)
6.34 S. K. CHANG: IEEE Trans. SMC-**3**, 257–265 (1973)
6.35 D. L. CASKEY, C. L. COATES: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 41–49
6.36 Y. NAKANO, K. NAKATA, Y. UCHIKURA, A. NAKAJIMA: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 172–178
6.37 S. YAMAMOTO, A. NAKAJIMA, K. NAKATA: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 187–196
6.38 M. YOSHIDA, M. EDEN: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 197–204
6.39 M. EDEN: IRE Trans. IT-**8**, 160–166 (1962)
6.40 D. R. ANDREWS, A. J. ATREBIN, K. C. HU: IBM J. Res. Dev. **12**, 364–371 (1968)
6.41 B. BLESSER, R. SHILLMAN, T. KUKLINSKI, C. COX, M. EDEN, J. VENTURA: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 33–40
6.42 C. COX, B. BLESSER, M. EDEN: in Proc. 2nd Intern. Joint Conf. Pattern Recognition, Copenhagen, Denmark (1974), pp. 228–232
6.43 S. MORI, T. MORI, K. YAMAMOTO, H. YAMADA, T. SAITO: in Proc. 2nd Intern. Joint Conf. Pattern Recognition, Copenhagen, Denmark (1974), pp. 233–236
6.44 K. HANAKATA: in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973), pp. 416–422
6.45 B. ARDEN: Personal communication

7. Two-Dimensional Mathematical Notation

R. H. ANDERSON

With 7 Figures

This chapter discusses the use of coordinate grammars for the recognition of two-dimensional (2 D) mathematics notation. The discussion is based on material in [7.1–2], and includes complete grammars for the recognition of commonly used arithmetic and matrix math notation. The format of the coordinate grammar rules has been completely revised for this chapter from earlier versions in an effort to increase the readability of the grammar. A top-down parsing scheme is used to partition a two-dimensional character configuration into subproblems. Syntax rules contain all necessary partitioning instructions. The syntax-directed recognition system described here has been successfully demonstrated in several prototype systems, and could be used to provide real-time interpretation of 2 D math expressions handprinted on a data tablet as input to an interactive computing system.

7.1 Mathematics Notation

Two-dimensional mathematics notation is an excellent domain for the application of syntactic pattern recognition techniques. The patterns are composed of primitive elements—symbols—which are easily isolated. The patterns have a spatial structure which determines their meaning. In addition, a computer program which recognizes 2 D math notation hand-printed on a data entry tablet can be used in conjunction with a character recognition program such as LEDEEN's (Ref. [7.3], Appendix VIII) to form a useful, natural interface to computer programs. Examples of interactive programs using 2 D math notation as input are contained in [7.4–7]. A useful overview of algorithms for the recognition and generation of 2 D math notation is given by MARTIN [7.8].

7.2 Coordinate Grammars

Coordinate grammars are phrase-structure grammars in which each production maps a set of symbols, located at given coordinates, into a new set of symbols whose coordinates are computed by a set of functions associated with the given production. They were first proposed by ANDERSON [7.1–2], and are discussed by MILGRAM et al. [7.9]. The following formal definition of a coordinate grammar is taken from Ref. [7.9], pp. 188–9.

A graphical rewriting grammar may be formally defined as a 6-tuple

$G = (T, N, D, n, P, g)$, where

- T is a finite set of terminal symbols
- N is a finite set of nonterminal symbols, $T \cap N = 0$
- D is a domain of "coordinates" (e.g., the set of integers)
- n is a positive integer, the number of coordinates used (i.e., the dimension of the space in which symbols are to be placed; the position of a symbol is determined by an n-tuple of coordinates)
- P is a finite set of "productions", each of which is a quadruple $(\Lambda, \Sigma, \pi, \Phi)$, where
  - Λ is a j-tuple of symbols, for some $j \geqq 1$
  - Σ is a k-tuple of symbols, for some $k \geqq 1$
  - π is a predicate with k arguments, each of which is an n-tuple of coordinates
  - Φ is a j-tuple of functions, each having k arguments; the arguments and function values are n-tuples of coordinates[1].

$g \in N$ is a special symbol, called the "goal" or initial symbol. A set S_{i+1} of symbols and associated n-tuples of coordinates is said to directly reduce into another such set S_i if there exists a production $(\Lambda, \Sigma, \pi, \Phi)$, for which Σ is a subset of S_{i+1}; its coordinates satisfy π; the coordinates of the symbols of Λ are obtained from those in Σ by applying the functions in Φ; and $S_{i+1} - \Sigma \cup \Lambda = S_i$. Similarly, S'' is said to reduce into S' if there exists $S'' = S_n$, $S_{n-1}, \ldots, S_1, S_0 = S'$ such that S_i directly reduces into S_{i-1}, $1 \leqq i \leqq n$. Finally, S is said to be a sentence of G if it reduces to $\{g\}$ (with some associated coordinates). The set of all sentences whose symbols are all terminals is called the terminal language of G.

7.3 A Syntax-Directed Recognition Algorithm

The recognition algorithm discussed here is top-down; it starts with the ultimate syntactic goal and the entire set of input characters and attempts to partition the problem into subgoals (and corresponding subsets of characters) until either every subgoal is reached, or else all possibilities have failed. A syntax rule, then, provides instructions for the partitioning[2] of a character set into subsets, and assigns a syntactic goal to each of these subsets.

A top-down parsing algorithm was chosen because it provides a natural way of hypothesizing the global properties of a configuration at an early stage of the recognition procedure. This feature is important because an operator symbol such as an integral sign affects the interpretation of the characters in its neighborhood ("dx" is a different syntactic unit in the configurations "$\int \sin x dx$" and "$cx + dx$"). A bottom-up algorithm could be devised which would provide an

[1] It is also assumed that productions always leave terminal symbols fixed.

[2] *Partition* is being used with the set-theoretic meaning of dividing a set into mutually exhaustive and mutually exclusive subsets.

equivalent recognition capability; it is not clear that it would be a more efficient recognizer.

The algorithm is also syntax-directed, meaning that all choices of subgoals and subsets are governed by syntax rules which are read as parametric data; the algorithm is independent of the content of these rules.

A top-down syntax-directed recognition scheme has several disadvantages which must be considered. It is slower than a machine-language program tailored to a specific job, since the syntax rules must be continually interpreted, and also since many incorrect parsings might be attempted before the correct one (if any) is found. A second disadvantage concerns error detection; a purely top-down syntax-directed algorithm often cannot pinpoint the error in an unsyntactic configuration. Both of these disadvantages may be overcome to a large degree by modifications to the parsing algorithm or syntax rules. A discussion of the implementation of the algorithm and its efficiency will be presented at a later point in this chapter. It is felt, however, that the disadvantages would be minimal in a highly interactive system in which the interpretation of each handwritten statement would be shown to the user upon completion of its syntactic analysis. In this case, the number of characters being analyzed at any time remains small, and any errors can be isolated and corrected as they appear.

7.3.1 Characters

We assume that a character-recognition program provides the following five items of information about each character recognized: its value (in ASCII or some other code) and its extent, indicated by the four values: *xmin*, *xmax*, *ymin*, *ymax*. The scale of the coordinate system used to determine the positional information may be arbitrary; all syntactic relationships are based on the positions of characters relative to each other.

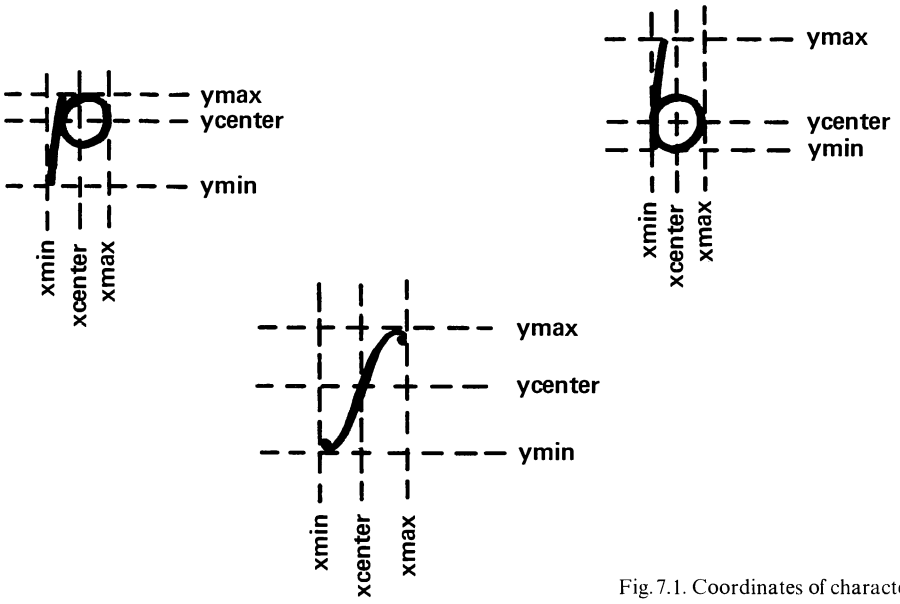


Fig. 7.1. Coordinates of characters

Table 7.1. Allowable input characters and their syntactic categories

Input character	*Syntactic category*
a, b, ..., z	letter
integers	unsignedint
real numbers	unsignedno
$+, -, \cdot, ., =, \propto, (,), [,], \Sigma, \pi, \int, \sqrt{\ }, \frac{\ }{\ }, \vert, \backslash$	(syntactic category represented by the symbol itself)

For efficiency in the syntactic analysis for the particular syntaxes discussed here, characters are preprocessed in the following manner.

By means of a table lookup or similar procedure, each character is given a syntactic category and two additional positional coordinates: *xcenter* and *ycenter*. These coordinates reflect the typographical center of the character, as illustrated in Fig. 7.1. The *xcenter* for a character is always the average of *xmin* and *xmax; ycenter* is calculated from *ymin* and *ymax* by a function which depends on the particular character. These center coordinates could best be supplied by the character-recognition program, since relevant information about the internal configuration of each handwritten character is available to it.

Table 7.1 gives the input characters, or strings of characters, needed for arithmetic and matrix syntaxes and the syntactic category which each of them is given by the preprocessing. It is assumed that each of these characters may be drawn in any size and at any location on the input surface. The terminal alphabet of the syntax, then, consists of the syntactic categories listed in the right-hand column of Table 7.1. In addition, certain rules test for the explicit letters *a*, *c*, *d*, *e*, *i*, *n*, *o*, *s*, *t* as components of trigonometric and function names.

7.3.2 Syntactic Units

Just as each character's position is described by the six spatial coordinates *xmin*, *xcenter*, *xmax*, *ymin*, *ycenter*, and *ymax*, higher syntactic units composed of these characters also will be assigned coordinates. The number and content of the coordinates assigned to a syntactic unit may in general vary and be used to transmit complex information during the syntactic analysis (for example, by having a list structure as a value). For the syntaxes under discussion each syntactic unit also has the six coordinates: *xmin*, *xcenter*, *xmax*, *ymin*, *ycenter*, *ymax*. Just as the *ycenter* of a character need not lie halfway between its *ymin* and *ymax*, the center of the arithmetic expression, for syntactic purposes, need not be halfway between its extrema; the center of an expression is usually determined by the position of the principal operator in that expression. Figure 7.2 illustrates this point. We also use an *m* coordinate to store the semantic meaning attached to a syntactic unit as a character string. Two other coordinates are used in the matrix syntax in Appendix 7.B: *rowsize* and *colsize;* these retain the current array size determined by syntactic processing of the configuration.

A syntax rule for two-dimensional analysis will specify replacements of syntactic categories contingent upon their correct relative replacement, i.e., contingent upon an examination of their relative coordinates. The next subsection describes how a coordinate grammar rule for 2 D analysis might be constructed.

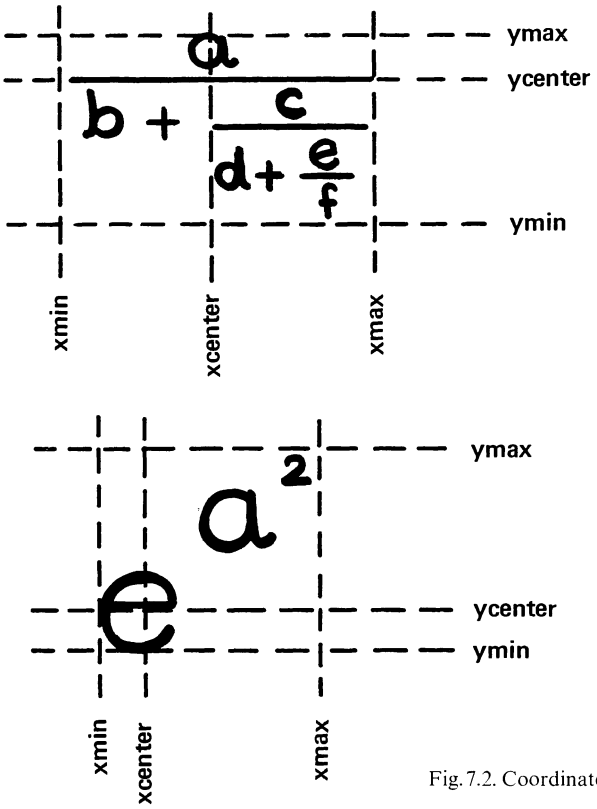


Fig. 7.2. Coordinates of syntactic units

7.3.3 Coordinate Grammar Rules for Two-Dimensional Character Configurations

Consider the following example of a context-free replacement rule for operating on a character string:

expression → expression "+" term .

In English, this might be interpreted as

Given the goal "expression" and a string of characters; try to partition the string into three substrings, where the first is an expression, and it is followed by the second, which is a "+" sign, and it is followed by the third, which is a term; if each of these subgoals is reached, report success; otherwise, report failure.

The analogous situation is more complicated in two dimensions. Consider the syntax rule we need to handle the following character configuration

$$\frac{a^2+b}{c}.$$

In English, we might say

Given the goal "division-type term" and a set of characters, where the position of each is described by a set of coordinates, try to partition the character set into three subsets S 1, S 2, and S 3 such that the following conditions hold:

1) S 1 is an expression,
2) S 2 contains the single character "horizontal line",
3) S 3 is an expression,
4) S 1 is above S 2, and bounded in the x-direction by the extent of S 2, and
5) S 3 is below S 2, and bounded in the x-direction by the extent of S 2.

If these tests are successful, assign a set of coordinates to the overall configuration, each of these being a function of the coordinates of S 1, S 2, and S 3; report these coordinates along with "success"; if not successful, report failure.

Several important differences between the syntax rules for linear and two-dimensional character configurations should be noted.

1) the linear rule reports only "success" or "failure"; the 2 D rule returns coordinate information in the event of success. These coordinates are necessary in the determination, at a higher level, of spatial relationships between syntactic units (just as conditions 4) and 5) in this rule tested relationships between "smaller" syntactic units).

2) in the linear case, the only relationship used between syntactic units is adjacency; this relationship is never explicitly tested, because this one-dimensional information is contained in the ordering of the input characters. In the 2 D case, there are many possible relationships between syntactic units; e.g., one may be above, to the right of, or within the other. These relationships are part of the syntactic structure of the character configuration and must be tested explicitly.

In summary, a coordinate grammar rule for a 2 D character configuration should contain the following information:

1) the syntactic category on the "left-hand side" of the rule,
2) directions for partitioning a character set into subsets,
3) syntactic goals associated with each of these subsets,
4) relations to be tested among syntactic subunits, if they are successfully parsed, and
5) the coordinate set of the left-hand side, as a function of the coordinates of the right-hand side subunits.

Rules Containing Terminal Symbols

We will now develop a notation in which all of the above information may be encoded. As an example, consider again the replacement rule applicable to the term

$$\frac{a^2+b}{c}.$$

Figure 7.3 gives a graphical representation of the desired replacement rule. That is, we wish to partition the "divterm" space into three subspaces, S 1, S 2, S 3, such

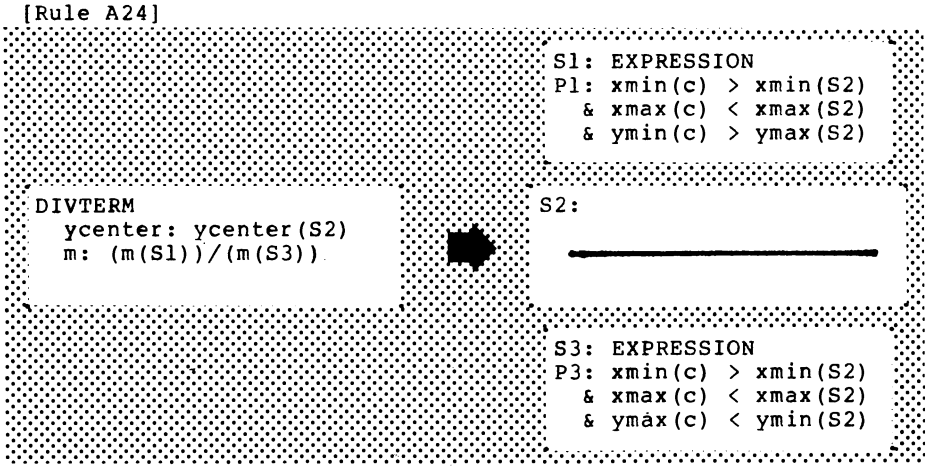


Fig. 7.3. Graphical form of replacement rule

that the second contains the single character "horizline". For this type of rule, in which a terminal character is one of the right-hand side components, the partitioning strategy will be as follows. If the set of characters under consideration does not contain the desired character (in this case, a horizontal line), then the rule does not apply; if there is exactly one instance of the desired character in the set, then the partitioning of the rest of the characters in the set will be based upon their position relative to that instance of the terminal character. In the example depicted by Fig. 7.3, a character, c, is placed in subset S 1 if and only if it meets the partitioning condition P 1: *xmin*(c) > *xmin*(horizline) and *xmax*(c) < *xmax*(horizline) and *ymin*(c) > *ymax*(horizline). A similar predicate tests the character for placement in subset S 3. If a character were found for which no predicate is true, the rule would be inapplicable to that character configuration.

If there are several instances of the desired terminal character in the set of characters under consideration, then these instances should be ordered and used successively as the basis for partitioning the other characters, until either an instance is found for which the partitioning is successful and all of the conditions in the syntax rule are satisfied by the resultant subsets, or else none is successful and the rule is therefore inapplicable.

It should be noted that the above partitioning strategy, used for rules with a terminal character on the right-hand side, places a restriction on the other (nonterminal) right-hand side syntactic units in the rule. It must be possible to delineate these units by mutually exclusive conditions which define each area by its spatial relationship to the terminal character(s) in the rule. This restriction considerably simplifies the partitioning algorithm, and does not in practice seriously limit the descriptive power of the syntax.

The following notation is used in Fig. 7.3: S 1, S 2, ..., Sn represent a partitioning of the characters in the set, S, under consideration. Each set Si has the

following attributes in addition to its syntactic category (shown capitalized):

$$\left.\begin{array}{l} xmin \\ xcenter \\ xmax \\ ymin \\ ycenter \\ ymax \end{array}\right\} \text{coordinates}$$

m meaning attached to Si, represented as a character string.

All of the above attributes of a set, Si, are assumed known if the definition for that set consists of a single terminal symbol.

The right-hand side (RHS) of a grammar rule consists of (white) areas representing syntactic categories S 1 ... Sn, with the syntactic category shown. (For terminal symbols, the category is represented by the symbol itself). For nonterminal categories, there is a corresponding partitioning condition Pi to be met by characters in that set. If there is a relation to be tested among successfully parsed Si subunits, it is shown in square brackets at the top of the rule's RHS. The left-hand side (LHS) consists of a single syntactic category, with a list of attribute assignments to be made if the rule is successfully applied. For brevity, the following assignments are assumed by default if the corresponding attribute name is not shown on the rule's LHS:

xmin (S):	min (*xmin*(S1), ..., *xmin* (Sn))
xmax (S):	max (*xmax*(S1), ..., *xmax*(Sn))
xcenter (S):	0.5 * (*xmin*(S) + *xmax*(S))
ymin (S):	min (*ymin*(S1), ..., *ymin*(Sn))
ymax (S):	max (*ymax*(S1), ..., *ymax* (Sn))
ycenter (S):	0.5 * (*ymin*(S) + *ymax*(S))
m (S):	*m* (S1) .

Other Types of Replacement Rules

Up to this point, the only type of replacement rule which has been discussed is one which has one or more terminal symbols on its right-hand side. These symbols were used as a basis for the partitioning of the other characters.

A second type of replacement rule is one which has exactly one nonterminal category on its right-hand side. A rule of this type is used to change the syntactic category assigned to a character set. Figure 7.4 gives an example of such a rule. Since the character set is not partitioned, neither partitioning instructions nor relations between syntactic units are needed.

The only other possible type of replacement rule is one which has several subcategories on its right-hand side, none of which is terminal. For simplicity in the partitioning algorithm, rules of this type will be restricted to the following "normal form": only two subcategories are allowed, and it must be possible to

Fig. 7.4. Replacement rule with right-hand side consisting of one nonterminal category

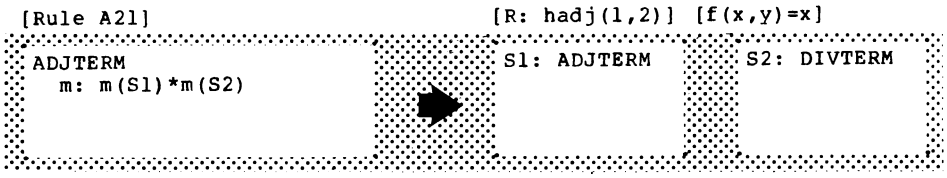


Fig. 7.5. Replacement rule with right-hand side consisting of two nonterminal categories

draw a straight line on the plane in which the characters are drawn, such that the line partitions the characters into the two subsets corresponding to the two syntactic subcategories.

For this type of rule, the partitioning strategy is the following. Let "f(x, y) = constant" be the equation of a straight line which would correctly partition the characters into the syntactic categories S1 and S2. The n characters are then ordered (using, for example, their center coordinates) by the function f(x, y), producing the ordered set $(c_1 c_2 \ldots c_n)$; the $n-1$ partitions

$$\begin{array}{ll} S1=\{c_1 \ldots c_{n-1}\} & S2=\{c_n\} \\ S1=\{c_1 \ldots c_{n-2}\} & S2=\{c_{n-1} c_n\} \\ \vdots & \vdots \\ S1=\{c_1\} & S2=\{c_2 \ldots c_n\} \end{array}$$

are then successively attempted until either one is found which meets all other criteria in the syntax rule (namely, the relation R) or else none are successful and the rule is therefore inapplicable[3].

An example of this type of rule is given in Fig. 7.5. The rule in this example recognizes implied multiplication between two adjacent syntactic units. The partitioning line is defined by $f(x, y) \equiv x$ (indicated in square brackets at the top of the rule's RHS) reflecting the fact that the two syntactic units are separated by the line "x = constant". The relation R checks that the magnitude of the difference between the *ycenter* coordinates of the two units is less than the parameter *htol* and that the separation between the two syntactic units is less than the parameter *hmax*. By manipulating the values of these parameters, the system user should be

[3] Although the stated partitioning strategy is sufficient, in practice it is much more efficient to calculate the spatial extent of each character in the direction perpendicular to the line "f(x, y) = constant". All partitions are then ignored between characters whose extents are overlapping.

able to "tune" the recognition algorithm to fit his printing, so that incorrect analyses are minimized. When not explicitly set by the user, parameters like *htol* and *hmax* would be given default values which are some function of the average size of the input characters.

7.4 Scope of Recognition Capability

Appendix 7.A contains a coordinate grammar for recognizing two-dimensional arithmetic expressions. Each rule is presented as a graphical "picture" as an aid in visualizing the relative spatial placement of the syntactic units.

In Appendix 7.A, it should be noted that the rule for the syntactic unit "alpha" (for "alphabetic character") uses only the center coordinates of "letter". This effectively contracts the spatial extent of each "letter" into a point. This is done so that a minor overlapping of adjacent handwritten characters will not cause a rejection of the configuration.

The design of the syntax occasionally relies on several features of the implementation of the parsing algorithm.

1) Given a set, S, and a syntactic goal G, each rule whose left-hand side has the syntactic category G is tried in the order given, until either a "success" is reported, or else all such rules have failed.

2) For rules with two nonterminal subcategories on the right-hand side, the partitioning algorithm places the minimum possible number of characters in the second subcategory listed (i.e., in category S 2).

3) For rules with a terminal character on their right-hand side, the partitioning conditions associated with sets S 1, ..., Sn of the rule are evaluated in order for

configuration	corresponding semantic string generated by syntax
$3 \sum_{i=1}^{n} \sin^2 x_i$	3* sum from i=1 to n of sin to power 2 of (x (i))
$\dfrac{a}{b + \dfrac{c}{d + \dfrac{e}{f}}}$	(a)/(b+ (c) / (d+ (e) / (f)))
sin x cos y	sin (x)*cos (y)
sin xy	sin (x*y)
$\int_{i-1}^{i+1} \dfrac{\sqrt{x^2+1}}{\sqrt{x^2-1}}\, dx$	definite integral from i - 1 to i + 1 of ((root 2 of x** (2) + 1)) /((root 2 of x**(2) - 1)) dx

Fig. 7.6. Some recognizable arithmetic expressions and their interpretation

each input character in the set being parsed by the rule; a character is placed in the set Si corresponding to the first true partitioning condition encountered.

The syntax could probably be designed in such a manner that it would be independent of the above features, but this would entail more rules and possibly additional syntactic categories.

Figure 7.6 contains some handprinted mathematical expressions which this grammar has been able to recognize (assuming reasonable values are assigned to such parameters as *htol* and *hmax*).

The same type of syntax rules may also be used to recognize many handprinted descriptions of matrices and vectors. Appendix 7.B contains a coordinate grammar for matrix notation. Figure 7.7 shows some of the recognition capabilities of this grammar.

configuration	interpretation	array example (illustrated for dimension = 5 when size is indeterminate)
$\begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$	n x n diagonal matrix	$\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}$
$\begin{bmatrix} a & b & c & d \\ & & & c \\ & & & b \\ 0 & & & a \end{bmatrix}$	4 x 4 uppertriangular matrix	$\begin{matrix} a & b & c & d \\ 0 & a & b & c \\ 0 & 0 & a & b \\ 0 & 0 & 0 & a \end{matrix}$
$\begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{bmatrix}$	3 x 3 explicit matrix	$\begin{matrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{matrix}$
$\begin{bmatrix} x & \text{———} & x \end{bmatrix}$	1 x n linked[a] rowvector	x x x x x
$\begin{bmatrix} 2 & 3 & & 0 \\ 1 & & & 3 \\ 0 & & 1 & 2 \end{bmatrix}$	n x n tridiagonal matrix	$\begin{matrix} 2 & 3 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 2 \end{matrix}$

[a] A vector is called "linked" when it contains a line connecting two elements. A linked vector is treated as a separate syntactic unit because its size is indeterminate.

Fig. 7.7. Some recognizable matrix notation and its interpretation

7.5 Implementation and Efficiency

The recognition algorithm discussed in this paper was first implemented in 1968 using the LISP 1.5 programming language on the CTSS time-sharing system of Project MAC at MIT. That implementation used simulated graphic input: a typed list of characters and their coordinates. Experimentation with this program has shown that a "bare" top-down syntactic analysis is quite inefficient, especially in rejecting nonsyntactic character configurations. A major source of this inefficiency is the partitioning strategy used for rules with two nonterminal syntactic units on their right-hand side; up to $n-1$ partitions may be generated by a set of n characters, and each of these partitions might require considerable processing.

However, by taking advantage of certain features of mathematical notation (e.g., its basically linear structure), and by the use of techniques employed in precedence analysis for programming languages, many "dead-end" analyses may be avoided. Two of the more important techniques used to gain efficiency are the following:

1) Although the order in which characters are given to the parsing algorithm is not important, it is presumed that a mathematical expression is written in a generally left-to-right manner. Also, for consistency, the rules in the syntax have been made left recursive whenever possible. Therefore, if there are multiple instances of a terminal character which appears on the right-hand side of a syntactic rule, these instances are used in the reverse of the order in which they were received. This strategy permits expressions like

$$a+b+c+d+e,$$

if written from left to right, to be parsed by the left-recursive rule

expression → expression "+" term

without any backup or false analysis. Left recursion is also desirable for the partitioning strategy for rules with two nonterminal right-hand side categories, in which the minimum number of characters are placed in the second (rightmost) category.

2) Two sets are formed for each nonterminal category in the syntax: the set L of all possible characters which can occur leftmost in a valid character configuration for the category, and the set R of all possible rightmost characters. These sets may be calculated quickly from the syntax itself, and need be recalculated only when a change is made to the syntax. The majority of rules which have two nonterminal categories on their right-hand side in the syntax for mathematical expressions has these categories separated by the line "x = constant". For these rules, the partitioning strategy entails ordering the character set by the x-coordinate. Using the L and R sets for the categories on a rule's right-hand side, it is easy to test possible partitions of the ordered character set for valid leftmost or rightmost characters. Invalid partitions may be discarded immediately. These tests are not employed for rules in which the two categories are partitioned by some line other than "x = constant".

It should be noted that the recognition algorithm stops after the first successful parse. The syntax rules should be ordered so that the first interpretation encountered is the desired one. For example, they should test for the entity "*sin*" before adjacent letters are interpreted to mean implied multiplication of variables.

The recognition algorithm has also been implemented in PL/I on an IBM 360/40 computer with RAND Tablet and IBM 2250 display at The RAND Corporation, Santa Monica, California. GRONER'S [7.10] character recognition program is used to provide hand-drawn character input. This implementation is described by BLACKWELL et al. [7.5]. As each character is drawn and recognized, it is displayed on a CRT in a canonical form in the same size and position in which it was drawn. When the desired mathematical expression has been drawn in this manner, the user signals that he has finished (by a button push or hand-drawn special character), and the parsing algorithm analyzes the configuration. Upon successful recognition, a box is drawn around the mathematical expression which has been recognized, and the meaning is displayed beneath the box.

The PL/I version of the recognition algorithm is not optimized for speed. Mathematical expressions involving about ten characters are recognized in several seconds of execution time. This time includes the handling of CRT interrupts; much could be done to increase the efficiency of this implementation.

More efficient recognizers of 2 D arithmetic mathematical notation can be designed which capitalize on speciál topological features of that notation and which sacrifice some of the flexibility of a purely syntax-directed approach. A description of such a specialized recognizer is contained in Ref. [7.11], Appendix D.

7.6 Summary

This chapter has discussed the use of a coordinate grammar for handling two-dimensional character configurations. Two applications were illustrated: a syntax for recognizing arithmetic expressions, and one for recognizing matrix descriptions. The recognition scheme is syntax directed for flexibility; by modifying the replacement rules which define the interactive language, a user may incorporate additions or alterations to the language which were unforeseen by the system designer.

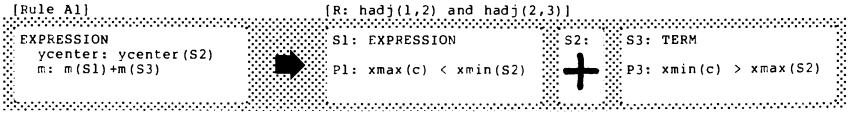
The problems of implementation and efficiency have been discussed briefly, with several methods indicated for considerably improving the efficiency of the parsing algorithm. Using these methods, a sufficient efficiency for interactive man-machine communication is attainable.

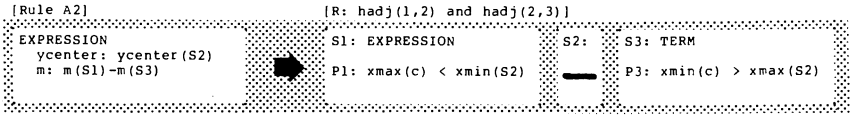
Appendix 7.A

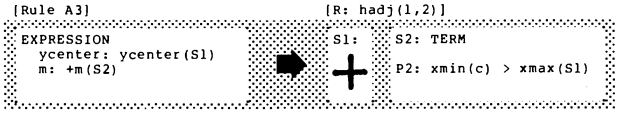
The rules in Appendix 7.A use the following global parameters:

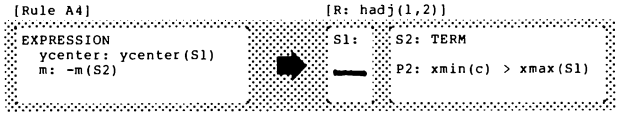
htol Maximum vertical deviation for two syntactic units on the same typographical line

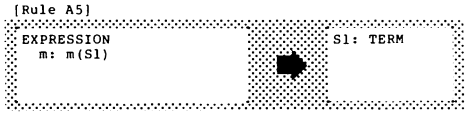
hmax Maximum horizontal separation between two adjacent syntactic units

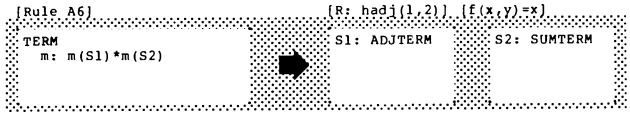

[Rule A1]
[R: hadj(1,2) and hadj(2,3)]
EXPRESSION
ycenter: ycenter(S2)
m: m(S1)+m(S3)
S1: EXPRESSION
P1: xmax(c) < xmin(S2)
S2:
S3: TERM
P3: xmin(c) > xmax(S2)

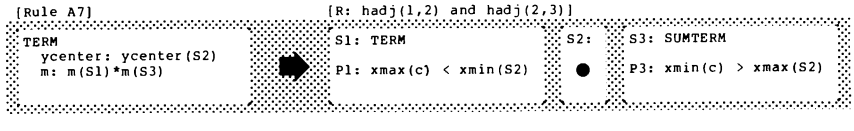

[Rule A2]
[R: hadj(1,2) and hadj(2,3)]
EXPRESSION
ycenter: ycenter(S2)
m: m(S1)-m(S3)
S1: EXPRESSION
P1: xmax(c) < xmin(S2)
S2:
S3: TERM
P3: xmin(c) > xmax(S2)


[Rule A3]
[R: hadj(1,2)]
EXPRESSION
ycenter: ycenter(S1)
m: +m(S2)
S1:
S2: TERM
P2: xmin(c) > xmax(S1)

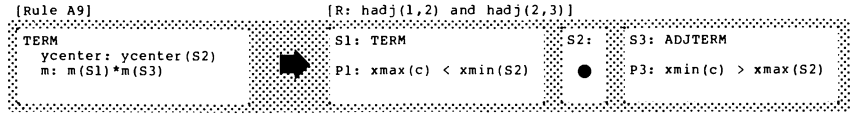

[Rule A4]
[R: hadj(1,2)]
EXPRESSION
ycenter: ycenter(S1)
m: -m(S2)
S1:
S2: TERM
P2: xmin(c) > xmax(S1)


[Rule A5]
EXPRESSION
m: m(S1)
S1: TERM


[Rule A6]
[R: hadj(1,2)] [f(x,y)=x]
TERM
m: m(S1)*m(S2)
S1: ADJTERM
S2: SUMTERM


[Rule A7]
[R: hadj(1,2) and hadj(2,3)]
TERM
ycenter: ycenter(S2)
m: m(S1)*m(S3)
S1: TERM
P1: xmax(c) < xmin(S2)
S2:
S3: SUMTERM
P3: xmin(c) > xmax(S2)

[Rule A8]
TERM
m: m(S1)
S1: SUMTERM


[Rule A9]
[R: hadj(1,2) and hadj(2,3)]
TERM
ycenter: ycenter(S2)
m: m(S1)*m(S3)
S1: TERM
P1: xmax(c) < xmin(S2)
S2:
S3: ADJTERM
P3: xmin(c) > xmax(S2)

[Rule A10]

```
TERM
  m: m(S1)
```

```
S1: ADJTERM
```

[Rule A11]

[R: abs(ycenter(S1)-ycenter(S2)) < htol
and (xmin(S2)-xmin(S1)) < hmax and hadj(2,3) and hadj(3,4)]

```
SUMTERM
  ycenter: ycenter(S1)
  m: definite integral
     m(S2) dm(S4)
```

```
S1:
```

$\int$

```
S2: DEFINTEGTERM

P2: xmin(c) > xmin(S1)
  & xmax(c) < xmin(S3)
```

```
S3:
```

d

```
S4: VARIABLE

P4: xmin(c) > xmax(S3)
```

[Rule A12]

[R: hadj(1,2) and hadj(2,3) and hadj(3,4)]

```
SUMTERM
  ycenter: ycenter(S1)
  m: indefinite integral
     of m(S2) dm(S4)
```

```
S1:
```

$\int$

```
S2: TERM

P2: xmin(c) > xmax(S1)
```

```
S3:
```

d

```
S4: VARIABLE

P4: xmin(c) > xmax(S3)
```

[Rule A13]

[R: hadj(1,4)]

```
SUMTERM
  ycenter: ycenter(S1)
  m: sum from m(S2) to
     m(S3) of m(S4)
```

```
S3: LIMIT
P3: xmin(c) > xmin(S1)
  & xmax(c) < xmax(S1)
  & ymin(c) > ymax(S1)
```

```
S1:
```

$\sum$

```
S2: SUMINITIAL
P2: xmin(c) > xmin(S1)
  & xmax(c) < xmax(S1)
  & ymax(c) < ymin(S1)
```

```
S4: TERM

P4: xmin(c) > xmax(S1)
```

[Rule A14]

[R: hadj(1,3)]

```
SUMTERM
  ycenter: ycenter(S1)
  m: sum over m(S2) of
     m(S3)
```

```
S1:
```

$\sum$

```
S2: VARIABLE
P2: xmin(c) > xmin(S1)
  & xmax(c) < xmax(S1)
  & ymax(c) < ymin(S1)
```

```
S3: TERM

P3: xmin(c) > xmax(S1)
```

[Rule A15]

[R: hadj(1,4)]

```
SUMTERM
  ycenter: ycenter(S1)
  m: product from m(S2)
     to m(S3) of m(S4)
```

```
S3: LIMIT
P3: xmin(c) > xmin(S1)
  & xmax(c) < xmax(S1)
  & ymin(c) > ymax(S1)
```

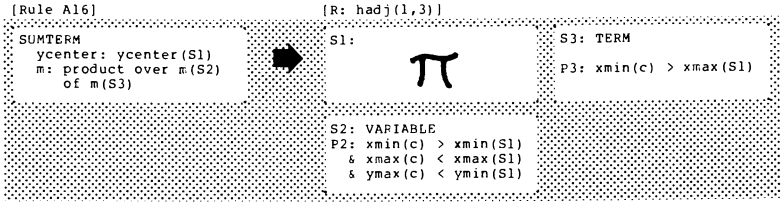
```
S1:
```

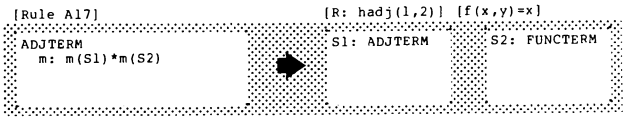
π

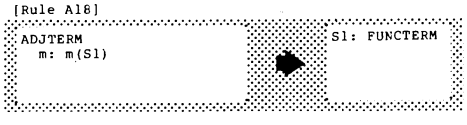
```
S2: SUMINITIAL
P2: xmin(c) > xmin(S1)
  & xmax(c) < xmax(S1)
  & ymax(c) < ymin(S1)
```

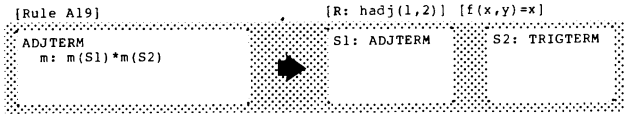
```
S4: TERM

P4: xmin(c) > xmax(S1)
```

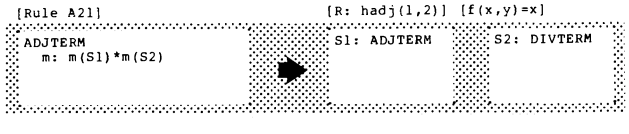

[Rule A16]
[R: hadj(1,3)]
SUMTERM
ycenter: ycenter(S1)
m: product over m(S2)
of m(S3)
S1:
π
S2: VARIABLE
P2: xmin(c) > xmin(S1)
& xmax(c) < xmax(S1)
& ymax(c) < ymin(S1)
S3: TERM
P3: xmin(c) > xmax(S1)


[Rule A17]
[R: hadj(1,2)] [f(x,y)=x]
ADJTERM
m: m(S1)*m(S2)
S1: ADJTERM
S2: FUNCTERM

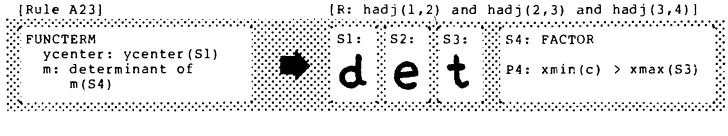

[Rule A18]
ADJTERM
m: m(S1)
S1: FUNCTERM


[Rule A19]
[R: hadj(1,2)] [f(x,y)=x]
ADJTERM
m: m(S1)*m(S2)
S1: ADJTERM
S2: TRIGTERM

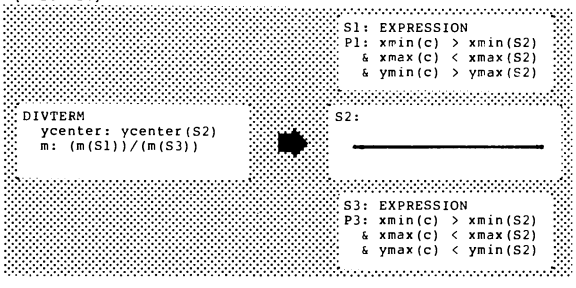
[Rule A20]
ADJTERM
m: m(S1)
S1: TRIGTERM


[Rule A21]
[R: hadj(1,2)] [f(x,y)=x]
ADJTERM
m: m(S1)*m(S2)
S1: ADJTERM
S2: DIVTERM

[Rule A22]
ADJTERM
m: m(S1)
S1: DIVTERM

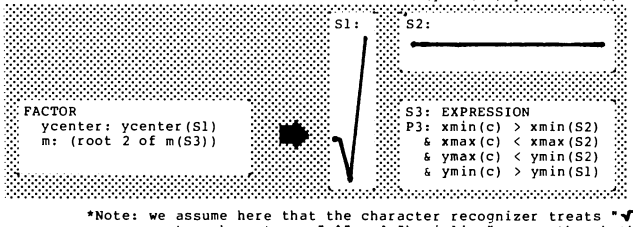

[Rule A23]
[R: hadj(1,2) and hadj(2,3) and hadj(3,4)]
FUNCTERM
ycenter: ycenter(S1)
m: determinant of
m(S4)
S1:
d
S2:
e
S3:
t
S4: FACTOR
P4: xmin(c) > xmax(S3)

[Rule A24]



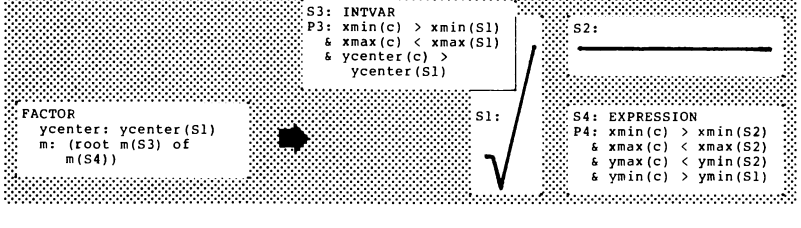
[Rule A25]

[Rule A26]* [R: (xmin(S2)-xmax(S1)) < hmax and abs(ymax(S1)-ycenter(S2)) < htol]

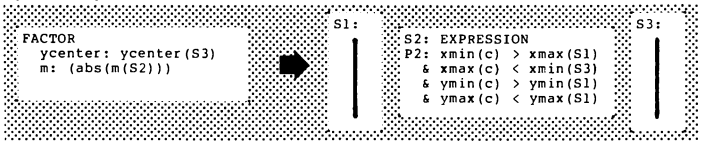


*Note: we assume here that the character recognizer treats "√‾" as two characters: "√" and "horizline", even though the combined symbol might have been drawn with one stroke.

[Rule A27] [R: (xmin(S2)-xmax(S1)) < hmax and abs(ymax(S1)-ycenter(S2)) < htol]



[Rule A28]



[Rule A29]

```
FACTOR
  ycenter: ycenter(S3)
  m: (m(S2))
```

```
S1:  (
S2: EXPRESSION
P2: xmin(c) > xmax(S1)
  & xmax(c) < xmin(S3)
  & ymin(c) > ymin(S1)
  & ymax(c) < ymax(S1)
S3:  )
```

[Rule A30]

[R: (xmin(S2)-xmax(S1)) < hmax
and (ycenter(S2)-ycenter(S1)) > htol]
[f(x,y)=x]

```
FACTOR
  ycenter: ycenter(S1)
  m: m(S1)**(m(S2))
```

```
S1: FACTOR
S2: EXPRESSION
```

[Rule A31]

[R: hadj(1,2)] [f(x,y)=x]

```
FACTOR
  ycenter: ycenter(S1)
  m: m(S1)m(S2)
```

```
S1: FACTOR
S2: MODIFIER
```

[Rule A32]

```
FACTOR
  m: m(S1)
```

```
S1: VARIABLE
```

[Rule A33]

```
FACTOR
  m: m(S1)
```

```
S1: UNSIGNEDNO
```

[Rule A34]

```
FACTOR
  m: m(S1)
```

```
S1: UNSIGNEDINT
```

[Rule A35]

[R: hadj(1,2)] [f(x,y)=x]

```
TRIGTERM
  ycenter: ycenter(S2)
  m: m(S1)(m(S2))
```

```
S1: TRIGOP
S2: ADJTERM
```

[Rule A36]

[R: hadj(1,2)] [f(x,y)=-x]*

```
DEFINTEGTERM
  ycenter: ycenter(S1)
  m: m(S2)m(S1)
```

```
S2: LIMITS
S1: TERM
```

* Note: the negative sign on this function definition has the effect of placing the minimum number of characters, such that the rule is satisfied, in the leftmost category (S2) rather than in the rightmost one as is usually the case.

[Rule A37] [f(x,y)=y]

```
LIMITS
  m: from m(S1) to
     m(S2) of
```

```
S2: LIMIT
S1: LIMIT
```

[Rule A38]

```
LIMIT
  m: m(S1)
```

```
S1: EXPRESSION
```

[Rule A39] [R: hadj(1,2)]

```
LIMIT
  m: m(S1)m(S2)
```

```
S1: +
S2: ∞
```

[Rule A40] [R: hadj(1,2)]

```
LIMIT
  m: m(S1)m(S2)
```

```
S1: —
S2: ∞
```

[Rule A41]

```
LIMIT
  m: m(S1)
```

```
S1: ∞
```

[Rule A42] [R: hadj(1,2) and hadj(2,3)]

```
SUMINITIAL
  ycenter: ycenter(S2)
  m: m(S1)=m(S3)
```

```
S1: VARIABLE
P1: xmax(c) < xmin(S2)
S2: =
S3: EXPRESSION
P3: xmin(c) > xmax(S2)
```

[Rule A43]

```
SUMINITIAL
  m: m(S1)=1
```

```
S1: VARIABLE
```

[Rule A44] [R: (xmin(S2)-xmax(S1)) < hmax
and (ycenter(S2)-ycenter(S1)) > htol]
[f(x,y)=x]

```
TRIGOP
  ycenter: ycenter(S1)
  m: m(S1) to power
     m(S2) of
```

```
S1: TRIGNAME
S2: EXPRESSION
```

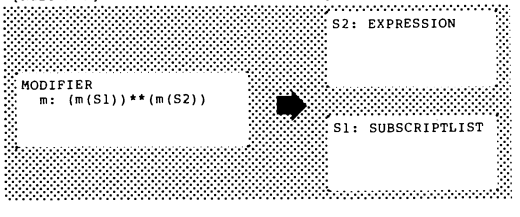
[Rule A45]

[Rule A46] [R: hadj(1,2) and hadj(2,3)]

[Rule A47] [R: hadj(1,2) and hadj(2,3)]

[Rule A48] [R: hadj(1,2) and hadj(2,3)]

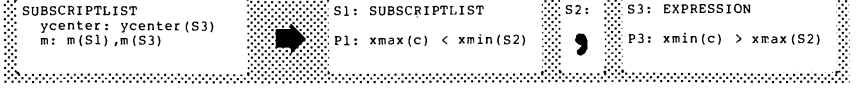
[Rule A49] [R: abs(xmin(S1)-xmin(S2)) < vtol] [f(x,y)=y]



[Rule A50]

[Rule A51]

[Rule A52] [R: hadj(1,2) and hadj(2,3)]



```
[Rule A53]
SUBSCRIPTLIST                    S1: EXPRESSION
  m: m(S1)
```

```
[Rule A54]                       [R: (xmin(S2)-xmax(S1)) < hmax
                                     and (ycenter(S1)-ycenter(S2)) > htol]
                                 [f(x,y)=x]
VARIABLE                         S1: ALPHA
  ycenter: ycenter(S1)                                S2: SUBSCRIPTLIST
  m: m(S1)(m(S2))
```

```
[Rule A55]
VARIABLE                         S1: ALPHA
  m: m(S1)
```

```
[Rule A56]
ALPHA                            S1: LETTER
  xmin: xcenter(S1)
  xmax: xcenter(S1)
  ymin: ycenter(S1)
  ymax: ycenter(S1)
  m: m(S1)
```

In addition, the following Boolean predicate is used:

$$hadj(\mathrm{i},\mathrm{j}) \equiv 0 < (\mathrm{c}_{\mathrm{j}1} - \mathrm{c}_{\mathrm{i}3}) < hmax \quad \text{and} \quad |\mathrm{c}_{\mathrm{i}5} - \mathrm{c}_{\mathrm{j}5}| < htol\,.$$

That is, the predicate "$hadj(\mathrm{i},\mathrm{j})$" is true for syntactic units Si and Sj if they are on the same typographical line, and Si is to the left of Sj, and they are sufficiently close together to be considered adjacent.

The following function "average" is also used in the rule descriptions:

$$avrg(\mathrm{x},\mathrm{y}) \equiv 0.5*(\mathrm{x}+\mathrm{y})\,.$$

Appendix 7.B

This appendix contains a coordinate grammar for the recognition of matrix notation. The following parameters are used in addition to those defined for use

in Appendix 7.A:

mhmax maximum horizontal separation between adjacent matrix elements

mvmax maximum vertical separation between two vertically adjacent matrix elements

vmax maximum vertical separation between adjacent characters belonging to the same matrix element

The following Boolean predicates are used:

$$sameh(y_1, y_2) \equiv |y_1 - y_2| < htol$$

$$samev(x_1, x_2) \equiv |x_1 - x_2| < vtol$$

$$hm(x_1, y_1, x_2, y_2) \equiv sameh(y_1, y_2) \quad \text{and}$$

$$hmax < (x_2 - x_1) < mhmax$$

"*hm*" is true for points (x_1, y_1) and (x_2, y_2) if they are spaced properly to belong to adjacent matrix elements in the same row, and (x_1, y_1) is to the left of (x_2, y_2).

$$vm(x_1, y_1, x_2, y_2) \equiv samev(x_1, x_2) \text{ and}$$

$$vmax < (y_2 - y_1) < mvmax$$

"*vm*" is true for points (x_1, y_1) and (x_2, y_2) if they are spaced properly to belong to adjacent matrix elements in the same column, and (x_1, y_1) is below (x_2, y_2).

$$dm(x_1, y_1, x_2, y_2) \equiv hm(x_1, y_1, x_2, y_1) \text{ and}$$

$$vm(x_2, y_2, x_2, y_1)$$

"*dm*" is true for points (x_1, y_1) and (x_2, y_2) if they are spaced properly to belong to diagonally adjacent matrix elements, and (x_1, y_1) is above and to the left of (x_2, y_2).

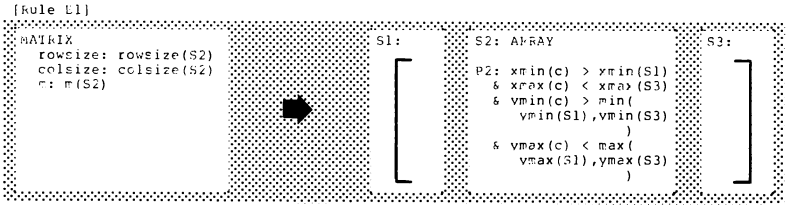
In addition to the coordinates used for the grammar in Appendix 7.A, the following two coordinates are used in this grammar:

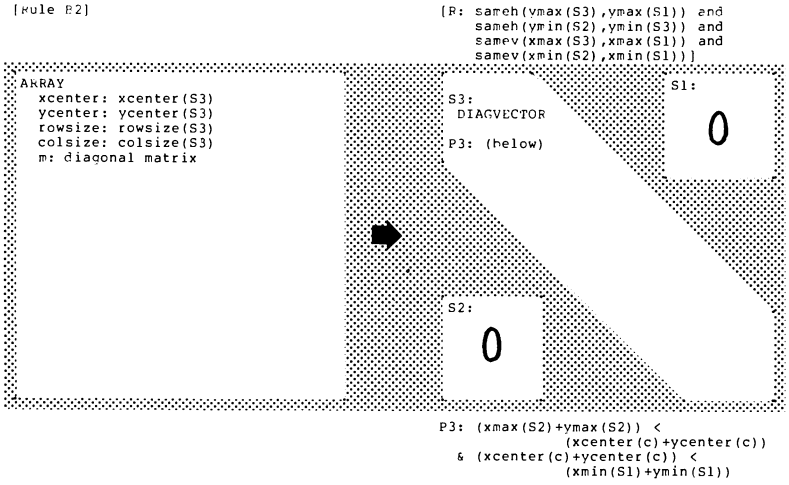
rowsize an integer representing the number of rows in the array, or "indeterminate"

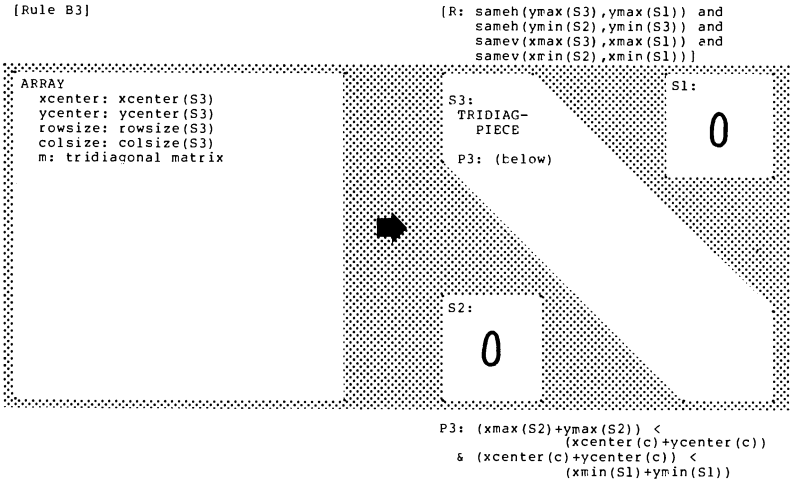
colsize an integer representing the number of columns in the array, or "indeterminate"

We have omitted an assignment for the semantic attribute, *m*(S), in the LHS of most rules; the particular semantics used depends heavily on the purpose for which this syntax-directed recognizer is used. The semantic component could, for example, build a list structure representing the matrix, and could retain a record of such attributes of the matrix as "uppertriangular", "diagonal", etc.

The grammar rules for recognition of matrix mathematical notation follow.

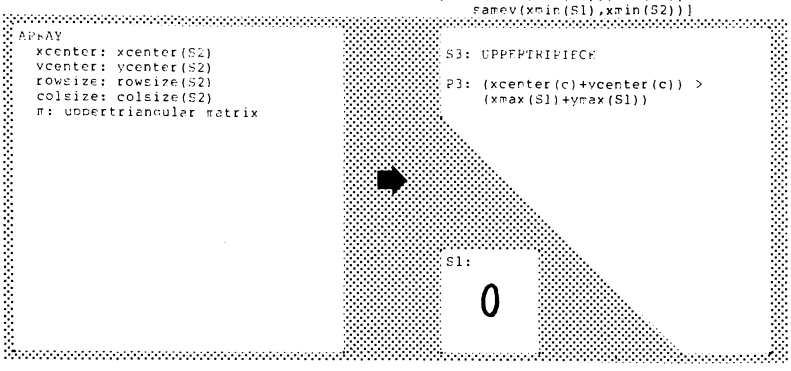

[Rule B1]
MATRIX
rowsize: rowsize(S2)
colsize: colsize(S2)
m: m(S2)
S1:
S2: ARRAY
P2: xmin(c) > xmin(S1)
& xmax(c) < xmax(S3)
& vmin(c) > min(
vmin(S1),vmin(S3)
)
& vmax(c) < max(
vmax(S1),vmax(S3)
)
S3:


[Rule B2]
[R: sameh(vmax(S3),ymax(S1)) and
sameh(ymin(S2),ymin(S3)) and
samev(xmax(S3),xmax(S1)) and
samev(xmin(S2),xmin(S1))]
ARRAY
xcenter: xcenter(S3)
ycenter: ycenter(S3)
rowsize: rowsize(S3)
colsize: colsize(S3)
m: diagonal matrix
S3:
DIAGVECTOR
P3: (below)
S1:
0
S2:
0
P3: (xmax(S2)+ymax(S2)) <
(xcenter(c)+ycenter(c))
& (xcenter(c)+ycenter(c)) <
(xmin(S1)+ymin(S1))


[Rule B3]
[R: sameh(ymax(S3),ymax(S1)) and
sameh(ymin(S2),ymin(S3)) and
samev(xmax(S3),xmax(S1)) and
samev(xmin(S2),xmin(S1))]
ARRAY
xcenter: xcenter(S3)
ycenter: ycenter(S3)
rowsize: rowsize(S3)
colsize: colsize(S3)
m: tridiagonal matrix
S3:
TRIDIAG-
PIECE
P3: (below)
S1:
0
S2:
0
P3: (xmax(S2)+ymax(S2)) <
(xcenter(c)+ycenter(c))
& (xcenter(c)+ycenter(c)) <
(xmin(S1)+ymin(S1))

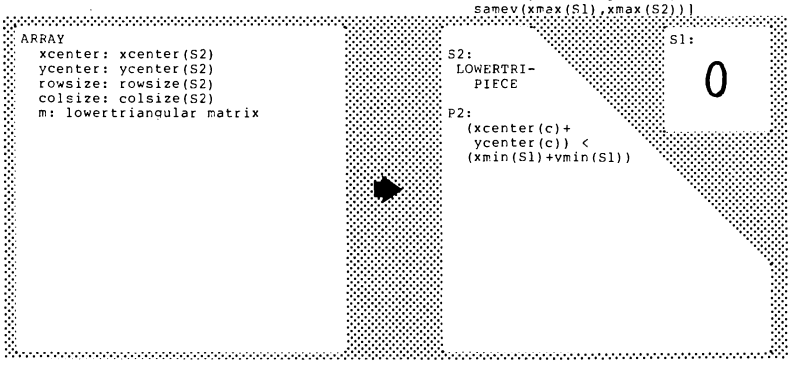
[Rule B4]

[R: sameh(ymin(S1),ymin(S2)) and
samev(xmin(S1),xmin(S2))]

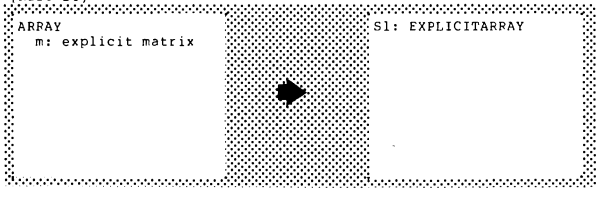


[Rule B5]

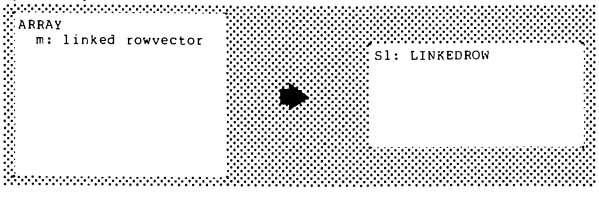
[R: sameh(ymax(S1),ymax(S2)) and
samev(xmax(S1),xmax(S2))]



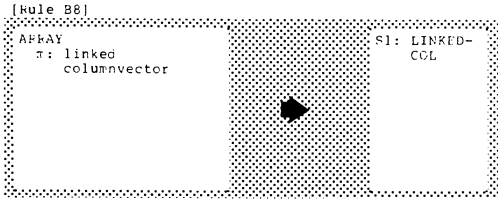
[Rule B6]



[Rule B7]

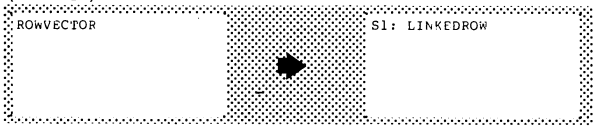


[Rule B8]

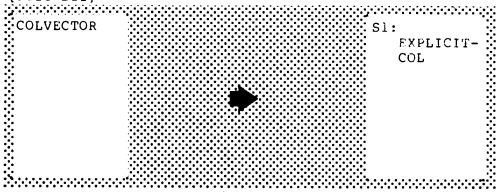


[Rule B9]

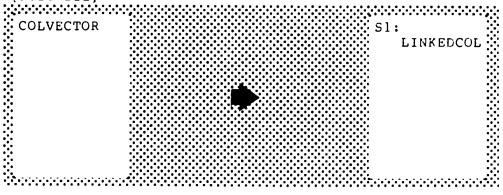
[Rule B10]



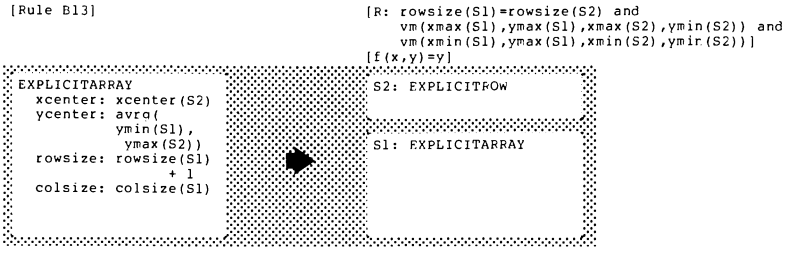
[Rule B11]



[Rule B12]



[Rule B13]



[Rule B14]

```
EXPLICITARRAY
  rowsize: 1
```

```
S1: EXPLICITROW
```

[Rule B15]

[P: hm(xmax(S1),ycenter(S1),xmin(S2),ycenter(S2))]
[f(x,y)=x]

```
EXPLICITROW
  xcenter: avrg(
           xmin(S1),
            xmax(S2))
  rowsize: 1
  colsize: colsize(S1)
                +1
```

```
S1: EXPLICITROW
```

```
S2: ELEMENT
```

[Rule B16]

```
EXPLICITROW
  rowsize: 1
  colsize: 1
```

```
S1: ELEMENT
```

[Rule B17]

[R: vm(xcenter(S1),xmax(S1),
xcenter(S2),ymin(S2))]
[f(x,y)=y]

```
EXPLICITCOL
  ycenter: avrg(
           ymin(S1),
            ymax(S2))
  rowsize: rowsize(S1)
                +1
  colsize: 1
```

```
S2: ELEMENT
```

```
S1:
   EXPLICIT-
   COL
```

[Rule B18]

```
EXPLICITCOL
  rowsize: 1
  colsize: 1
```

```
S1: ELEMENT
```

[Rule B19]

[R: hm(xmax(S1),ycenter(S1),xmin(S2),ycenter(S2)) and
hm(xmax(S2),ycenter(S2),xmin(S3),ycenter(S3))]

```
LINKEDROW
  xcenter: avrg(
           xmin(S1),
            xmax(S3))
  rowsize: 1
  colsize: indeter-
             minate
```

```
S1: ROWVECTOR

P1: xmax(c) < xmin(S2)
```

```
S2:
```

```
S3: ELEMENT

P3: xmin(c)
      >
   xmax(S2)
```

[Rule B20]

```
LINKEDROW
  xcenter: avrg(
           xmin(S1),
            xmax(S2))
  rowsize: 1
  colsize: indeter-
             minate
```

[R: hm(xmax(S1),vcenter(S1),xmin(S2),vcenter(S2))]
[f(x,y)=x]

```
S1: LINKEDROW
```

```
S2: ELEMENT
```

[Rule B21]

```
LINKEDCOL
  ycenter: avrg(
           ymin(S1),
            ymax(S3))
  rowsize: indeter-
             minate
  colsize: 1
```

[R: vm(xcenter(S1),xmax(S1),
xcenter(S2),ymin(S2)) and
vm(xcenter(S2),ymax(S2),
xcenter(S3),ymin(S3))]

```
S3: ELEMENT
P3: ymin(c)
      >
    ymax(S2)
```

```
S2:
```

```
S1: COL-
    VECTOR
P1: ymax(c)
      <
    ymin(S2)
```

[Rule B22]

```
LINKEDCOL
  ycenter: avrg(
           ymin(S1),
            ymax(S2))
  rowsize: indeter-
             minate
  colsize: 1
```

[R: vm(xcenter(S1),ymax(S1),
xcenter(S2),ymin(S2))]
[f(x,y)=y]

```
S2: ELEMENT
```

```
S1:
    LINKEDCOL
```

[Rule B23]

```
TRIDIAGPIECE
  xcenter: xcenter(S1)
  ycenter: ycenter(S1)
  rowsize: rowsize(S1)
  colsize: colsize(S1)
```

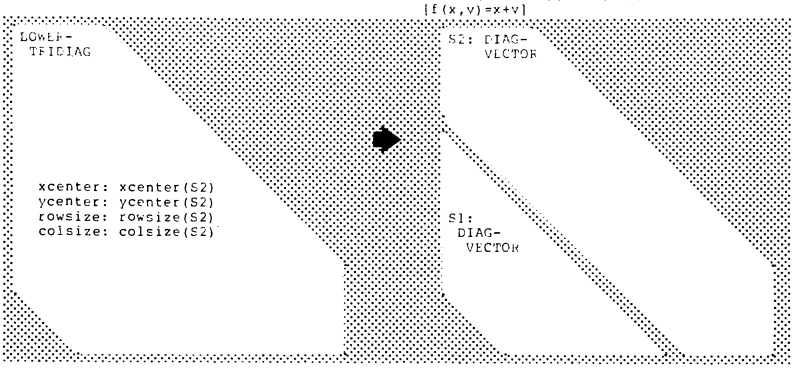
[R: hm(xmin(S1),ymax(S1),xmin(S2),ymax(S2)) and
vm(xmax(S1),ymin(S1),xmax(S2),ymin(S2))]
[f(x,y)=x+y]

```
S1:
    LOWER-
    TRIDIAG
```

```
S2: DIAG-
        VECTOR
```

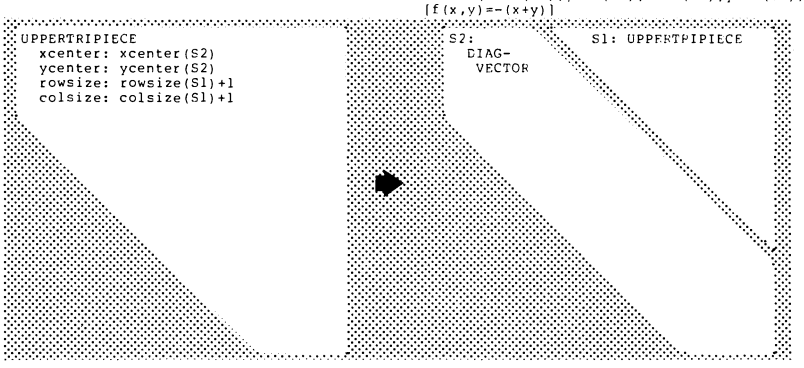
[Rule B24]

[R: hm(xmax(S1),ymin(S1),xmax(S2),ymin(S2)) and
vm(xmin(S1),ymax(S1),xmin(S2),ymax(S2))]
[f(x,y)=x+y]

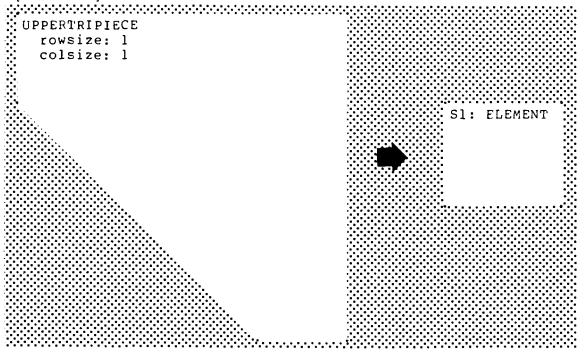


[Rule B25]

[R: hm(xmin(S2),ymax(S2),xmin(S1),ymax(S1)) and
vm(xmax(S2),ymin(S2),xmax(S1),ymin(S1))]
[f(x,y)=-(x+y)]

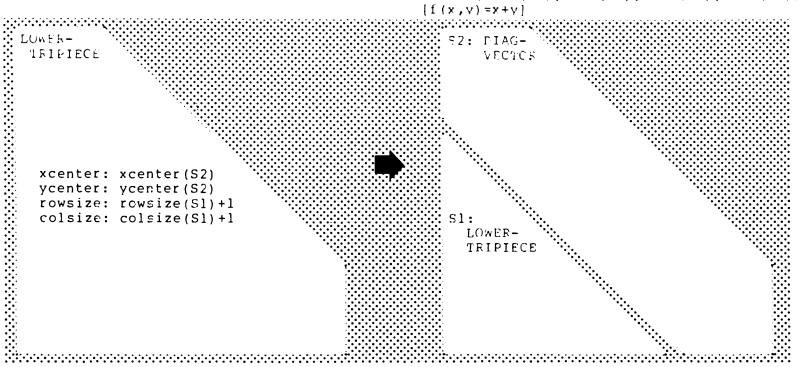


[Rule B26]

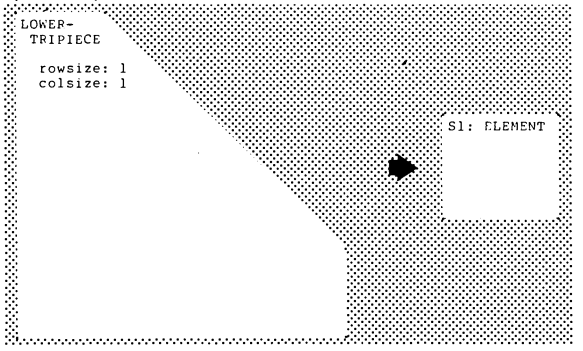


[Rule B27]

[R: hm(xmax(S1),ymin(S1),xmax(S2),ymin(S2)) and
vm(xmin(S1),ymax(S1),xmin(S2),ymax(S2))]
[f(x,y)=x+y]

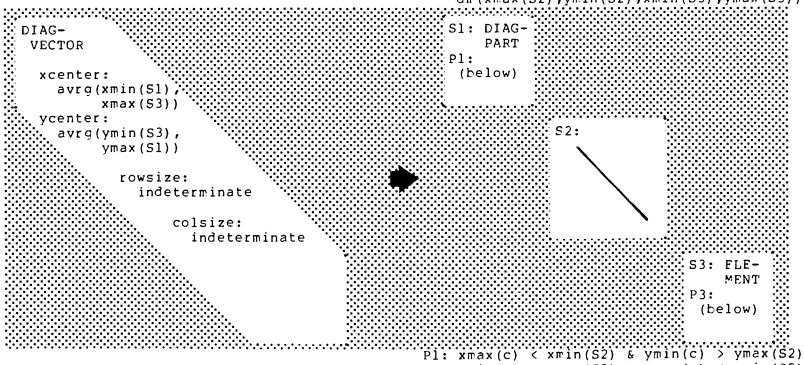


[Rule B28]



[Rule B29]

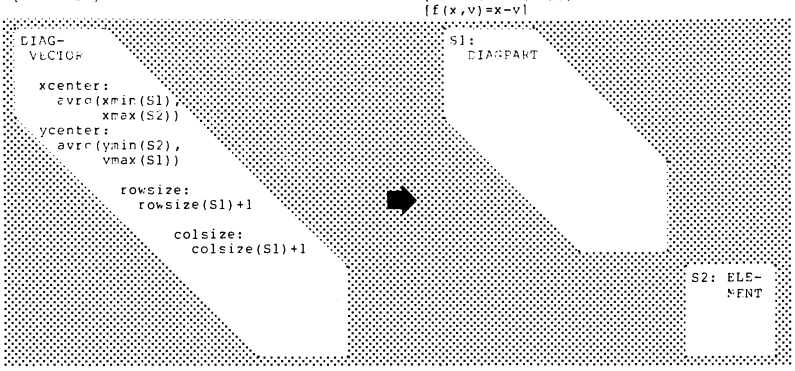
[R: dm(xmax(S1),ymin(S1),xmin(S2),ymax(S2)) and
dm(xmax(S2),ymin(S2),xmin(S3),ymax(S3))]



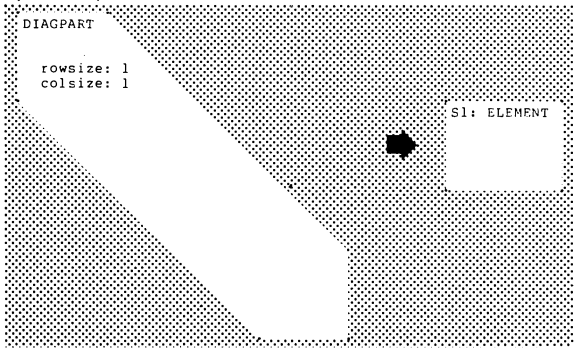
P1: xmax(c) < xmin(S2) & ymin(c) > ymax(S2)
P3: xmin(c) > xmax(S2) & ymax(c) < ymin(S2)

[Rule B30]

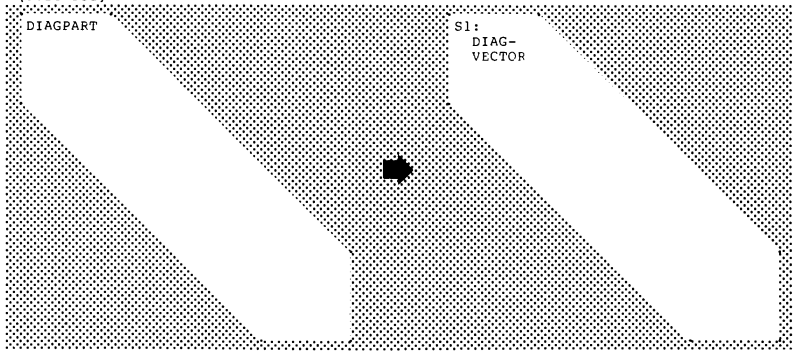
[k: dr(xmax(S1),ymin(S1),xmin(S2),ymax(S2))]
[f(x,y)=x-y]



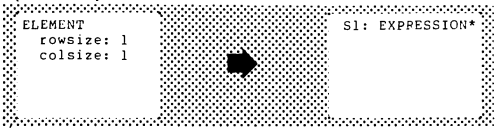
[Rule B31]



[Rule B32]



[Rule B33]



*Note: the syntax for "expression" is given in Appendix 7.A

References

7.1 R. H. ANDERSON: Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics, in *Interactive Systems for Experimental Applied Mathematics*, ed. by M. KLERER and J. REINFELDS (Academic Press, New York 1968)

7.2 R. H. ANDERSON: Ph D Thesis, Department of Engineering and Applied Mathematics, Harvard University, Cambridge, Mass. (1968)

7.3 W. N. NEWMAN, R. F. SPROULL: *Principles of Interactive Computer Graphics* (McGraw-Hill, New York 1973)

7.4 H. R. LEWIS: in *Proc. 1968 ACM National Conference* (Association for Computing Machinery, New York 1968), pp. 717–724

7.5 F. W. BLACKWELL, R. H. ANDERSON: in *Proc. 1969 ACM National Conference* (Association for Computing Machinery, New York 1969), pp. 551–557, 349–350

7.6 T. G. WILLIAMS: in *Proc. Fall Joint Computer Conference*, Vol. 27, Part 1 (AFIPS Press, Montvale, New Jersey 1966), pp. 499–506

7.7 M. KLERER, J. MAY: in *Proc. Fall Joint Computer Conference*, Vol. 27, Part 1 (AFIPS Press, Montvale, New Jersey 1965), pp. 63—75

7.8 W. A. MARTIN: Computer Input/Output of Mathematical Expressions, in *Proc. Second Symposium on Symbolic and Algebraic Manipulation.* (Association for Computing Machinery, New York 1971)

7.9 D. L. MILGRAM, A. ROSENFELD: in *Graphic Languages*, ed. by F. NAKE, A. ROSENFELD (North-Holland Publ. Co., Amsterdam 1972), pp. 187–191

7.10 G. F. GRONER: in *Proc. Fall Joint Computer Conference*, Vol. 29 (AFIPS Press, Montvale, New Jersey, 1966), pp. 591–601

7.11 K. S. FU: *Syntactic Methods in Pattern Recognition*, (Academic Press, New York 1974)

8. Fingerprint Classification

B. MOAYER and K. S. FU

With 17 Figures

8.1 Historical Background

Development of advanced digital computers has attracted the interest of government, industry, and many researchers in picture processing. Picture classification is concerned with classification of unknown patterns into one of the several categories. The problem is to find a suitable technique for the pattern of interest. Often one is interested in unique identification of the unknown, but in the field of pattern recognition one is more concerned with determining into which of a number of categories the unknown pattern best fits. A general view indicates that the selection of categories depends on the subjective opinion of the researcher; and suitability of the classification technique can be determined only by the degree of confidence. Today one of the most interesting unknown patterns is fingerprint impressions which is the subject of this chapter.

8.1.1 Fingerprint Pattern

The early history of fingerprints is not exactly known, however, the credit for scientific contributions in fingerprinting should be given to many people, especially to Dr. HENRY FAULDS (1843–1930), HARRIS WILDER (1864–1928), and HEINRICH POLL (1877–1939) [8.1]. The first major scientific contribution to fingerprint pattern recognition was made by FRANCIS GALTON (1822–1916) who did establish the fact that no two fingerprints were alike and classified the patterns into three major classes for filing purposes. EDWARD HENRY (1850–1931) [8.1] made a more advanced classification of the fingerprints, which is most widely used by the law enforcement people and is known as "Henry's Classification". The real significance of fingerprint impressions is mainly due to the following criteria:

1) Unchangeability: The configuration and details of the fingerprint patterns are permanent and never change through life until the skin disintegrates after death.

2) Uniqueness: The degree of variation of the ridges is so high that no two fingerprints have the same characteristics.

3) Classification: The variations of fingerprint patterns are within a limit which allows a systematic classification of the patterns.

Presentation of Henry's classification requires the knowledge of some technical terms which are presented next and are according to the FBI fingerprint manual [8.2].

1) Pattern Area. The area of the fingerprints containing the cores, deltas, and the ridges is called a pattern area. While the pattern area of loops and whorls can be easily defined, it is very difficult to define the area of arch patterns. Henry's classification is mainly based on the features which are located inside the pattern area.

2) Type Lines. The two innermost ridges confining the pattern area are called type lines. Type lines must start in parallel, diverge, and surround or tend to surround the pattern area. The immediate outside ridge is taken as the continuation of type lines, in case of discontinuity of the type lines.

3) Bifurcation. Division of a ridge into two or more ridges is called bifurcation.

4) Divergence. The spreading apart of two parallel or nearly parallel ridges is called divergence.

5) Focal Points. The two elements located in the pattern area, namely core and delta, are called focal points.

6) Delta. The point on or in front of the ridge closest to the divergence center of the type lines is defined as delta point. Delta could be any element such as a dot, a short ridge, a bifurcation, the end of a ridge, the intersection of two ridges. A set of rules is considered in selecting the delta point in the case of ambiguity.

7) Core. The core, which is approximately the center of loop fingerprints, is located within or on the innermost recurve. Whorls and arches do not have any cores. A set of rules is used for defining the core in the case of ambiguity.

8) Ridge Count. The number of ridges intervening between the delta and the core is called ridge count.

The fingerprint classification used by FBI, which is based on Henry's classification, consists of the following eight classes.

1) Plain Arch. Plain arch is the pattern in which the ridges enter on one side, rise in the middle, and flow or tend to flow out from the other side as shown in Fig. 8.1.

2) Tented Arch. Tented arch is the pattern in which the riges have the tendency to enter from one side and flow out from the other side, with the exception that the ridges form either an angle or an upthrust at the center. The patterns, with only two out of three characteristics of whorl type, are classified as tented arch. Figure 8.2 shows a typical tented arch fingerprint.

3) Radial Loop. Radial loop is the pattern in which one or more of the ridges enter on either side of the impression, recurve, touch or pass an imaginary line drawn from the delta to the core, and terminate on or toward the same side of the impression from which such a ridge or ridges entered. Radial loops, as shown in Fig. 8.3, must have sufficient recurve, a delta, and a ridge count across a looping ridge; the proper location of the delta and core is extremely important in the classification of the patterns. The ridges of radial loop flow toward the radius bone (thumb).

4) Ulna Loop. Ulna loop is exactly like the radial loop except the direction of the ridges which is toward the ulna bone (the little finger).

5) Plain Whorl. Plain whorl is the pattern in which at least two deltas and one recurving ridge such as a spiral or any variation of a circle must exist

Fig. 8.1. Arch pattern

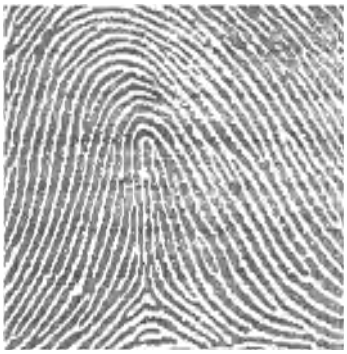


Fig. 8.2. Tented arch pattern

Fig. 8.3. Plain loop

Fig. 8.4. Whorl pattern

Fig. 8.5. Central pocket loop pattern

as shown in Fig. 8.4. In any plain whorl, an imaginary line drawn between the two deltas must touch or cross at least one of the recurving ridges within the pattern area. However, the presence of any appendage or spike may destroy the sufficient recurve.

6) Central Pocket Loop. In the central pocket loop, as shown in Fig. 8.5, which has two deltas and at least a ridge making a complete circuit as in the

Fig. 8.6. Double loop pattern

Fig. 8.7. Accidental pattern

plain loop, the imaginary line drawn between the two deltas must not touch any of the recurve ridges within the pattern area.

7) Double Loop. There are two deltas and two separate loops with separate and distinct shoulders in each double or twinned loop as shown in Fig. 8.6.

8) Accidental Whorl. The accidental, as shown in Fig. 8.7, is a pattern consisting of a combination of two or more different types of configurations, with the exception of the plain arch; or a pattern which does not belong to any of the other classes.

The FBI has subclassified each of the above classes; however, the system is still quite inefficient because its classes are extremely unbalanced. For instance, the ten most populated classes contain 55 percent of the entire file while some classes have hardly any member [8.3].

There are a number of fingerprint patterns which cannot be classified into any of the classes unambiguously. Sometimes, multiplicity of selection of focal points or spoilage of the sufficient recurve by the presence of a spike or an appendage may lead to erroneous classification. It appears that the complexity of the definition of the focal points leaves room for interpretation which makes Henry's classification system undesirable for computerized fingerprint recognition.

8.1.2 Automatic Fingerprint Identification

The need for an automatic fingerprint identifier has attracted many people and companies to conduct research in this area. A few of these techniques will be mentioned here.

TRAURING [8.4] has proposed basically the registration of the relative location of ridge and valley ending of some portions of the pattern. KINGSTON [8.5] suggested a semiautomated system because of the tedious classification scheme and the amount of time required for processing each pattern. WEGSTEIN and RAFFERTY [8.6–8] have considered several descriptors such as ridge endings, bifurcations, enclosures, and some others for describing the fingerprint impressions. The approach is based on matching a constellation or group of minutiae formed around a particular minutia. A discrimination or score matrix containing the percentage of misses and false matches is used for

evaluating the performance of the matching algorithm. SHELMAN [8.9] also selected the ridge endings and slope as the descriptors; however, his prime goal was to classify the patterns into several equally large groups in order to minimize the search time of the recognizer. At Calspan [8.10–12], in addition to concentration on the fingerprint reader, extractor of minutiae location and ridge direction, a tremendous effort has been devoted to the design of an efficient reader and an on-line data display for the fingerprint patterns. The Calspan system processes the patterns in five steps; namely, digitizing the pattern, removing the gaps and blots and producing the binary fingerprint image, preprocessing, minutiae detection, and post-editing the pattern. MARAM [8.13] and HORVATH [8.14] have attempted to use optical techniques for identification of fingerprints. Maram attempted to match the Fourier transform and the cross correlation of the patterns. The technique provides all the details of the pattern; however, the numerous matching filters required and the slow process are the real drawbacks. The holographic technique also is a slow process and requires numerous filters. However, the optical techniques generally have the advantage of translational and rotational invariance. A semiautomated system has been developed by the Electo Dynamic Division to encode latent prints. The system is currently under study by NYSIIS [8.3]. GRASSELLI [8.15] was the first to advocate the linguistic approach in fingerprint identification. He suggested the subdivision of the pattern into "sample" squares with each containing the direction of the predominant slope of the ridge passing through it. TOU and HANKLY [8.16, 17] did not formulate any form of language; however, their topological encoding scheme of fingerprint patterns has implicit context-free language structure. With this technique, it appears that a great deal of information has been lost as a result of data compression based on the redundancy assumptions among the ridges which may lead to ambiguous description of the patterns. Finally LEVI and SIROVICH [8.18] have focused on a technique which detects the global structure of the pattern with a set of paths using a linguistic-oriented algorithm. There are other techniques which have not been mentioned here only because of space limitations.

8.2 Syntactic Approach

The prime goal of the research in this field has been toward matching of the fingerprints, i.e., determining whether two fingerprints are exactly the same or not. However, considering the number of patterns and the degree of variation which is sometimes so small that even the experts have difficulty in performing the matching, it seems that the final matching must be accomplished manually or interactively. Therefore, the aim here is toward classifying the patterns into many categories for minimizing the search time. The syntactic approach seems to be the most suitable technique for classification because of its inherent hierarchical structure of the fingerprint patterns whose syntax structure consists of a set of ridges, a set of features, and the relationship among them within the pattern area.

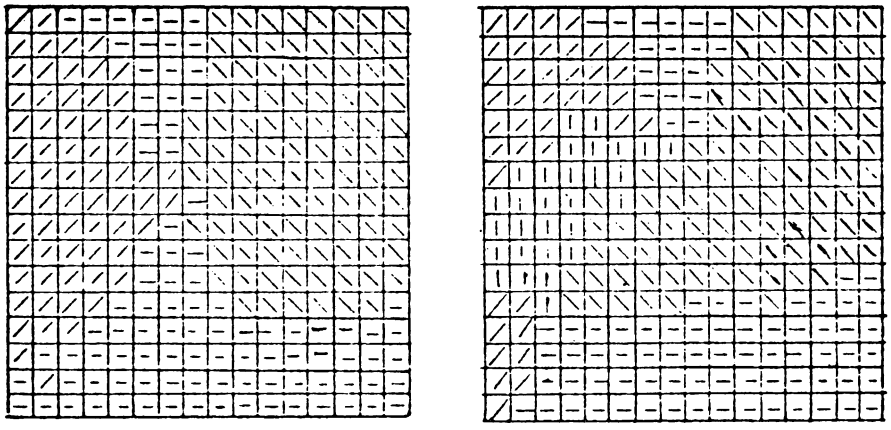
In general, a digital pattern recognition system consists of four entities—a digitizer, a preprocessor, a feature extractor, and a classifier.

8.2.1 Digitizer

The fingerprint image is entered as a photographic transparency and scanned by a computer-controlled flying spot scanner. Digitized pictures usually involve an enormous amount of data which requires large memory capacity. But there is also a high degree of redundancy in the pictorial data. To avoid the problem of memory requirement, every fingerprint impression is digitized into 192×192 data points, i.e., the essential information of the fingerprint, namely the pattern area, is considered.

8.2.2 Preprocessing

All digitized fingerprints have a lot of noise such as the extra thickness of the ridges which is due to the ink when the fingerprint is taken. A preprocessor algorithm [8.19] has been used to eliminate the noise and map the multi-grey level patterns into binary grey level patterns. To eliminate the redundant information and noise further, each pattern is represented by a sampling matrix as suggested by GRASSELLI [8.15]. That is, each pattern is divided into 256 sampling squares or windows, each containing 12×12 binary bits. Since the slope of the ridges in each window is almost constant, the ridges of each window may be represented by a directional code. A set of four directional codes is sufficient to generate a close approximation of the fingerprint impressions and, meanwhile, limit the number of possible combinations of directional codes for feature extraction. The four directional codes 0, 1, 2, and 3 are four straight lines making angles of -45, 0, 45, and 90 degrees with respect to the xy-coordinates, respectively. Hence, each sampling window contains one of the directional codes representing the dominant slope of the ridges running through the sampling square. The directional code of each sampling square is determined by the preprocessor consisting of skeletonization, tracing, and code detection algorithms which are explained in full detail in [8.19, 20]. Figure 8.8 shows typical sampling matrices of an arch and a whorl fingerprint pattern.



a b

Fig. 8.8. The sampling matrices of arch (a) and whorl (b)

8.2.3 Feature Extractor

The proper selection of features in the syntactic approach is also very important. The terminal primitives of any language directly affect the length of the terminal strings and the complexity of the generating grammars and,

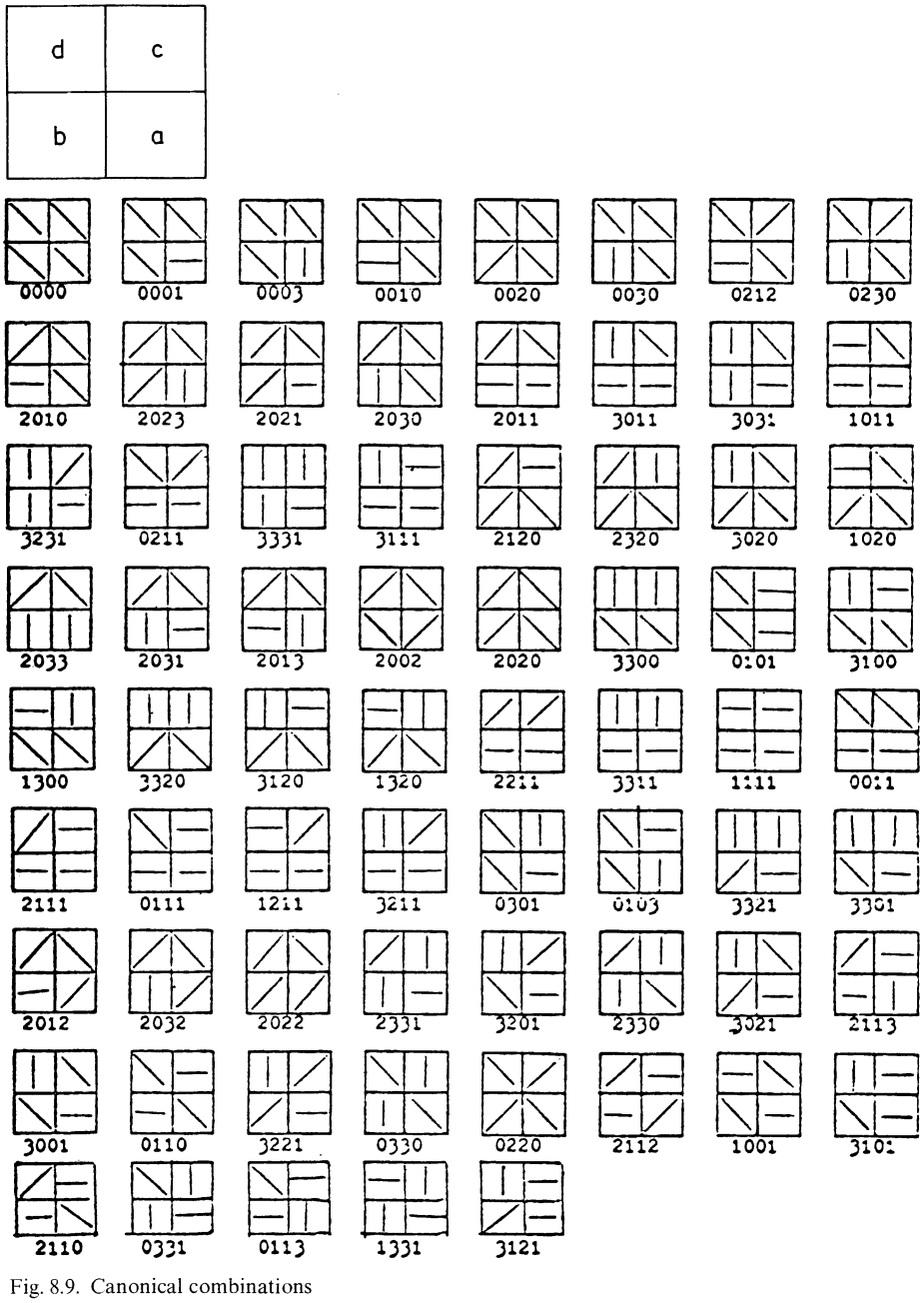


Fig. 8.9. Canonical combinations

consequently, the efficiency of the parsing algorithm. The feature extraction process is a mapping of the syntax of the pattern into a one-dimensional terminal string. Since the mapping function is highly dependent on the structure of the pattern, one can hardly talk about the optimality of the feature extraction.

Here the input to the feature extractor is the sampling matrix of 16×16 windows with the directional codes. The mapping assigns a feature for every four neighboring windows. Every four windows form an ordered set $\langle a, b, c, d \rangle$ which corresponds to the directional codes i_a, i_b, i_c, i_d belonging to the domain $D = \{0, 1, 2, 3\}$. Then every ordered set may be represented by an integer called containment code C of the set [8.21].

$$C = (i_a i_b i_c i_d) = 4^3 i_a + 4^2 i_b + 4^1 i_c + 4^0 i_d$$

where any containment code C can be represented by a 12-bit binary number. Because each member of the ordered set can assume one of four directional codes, there are 256 containment codes; however, only 69 of the combinations are canonical and the rest are merely the canonical forms rotated by 90, 180,

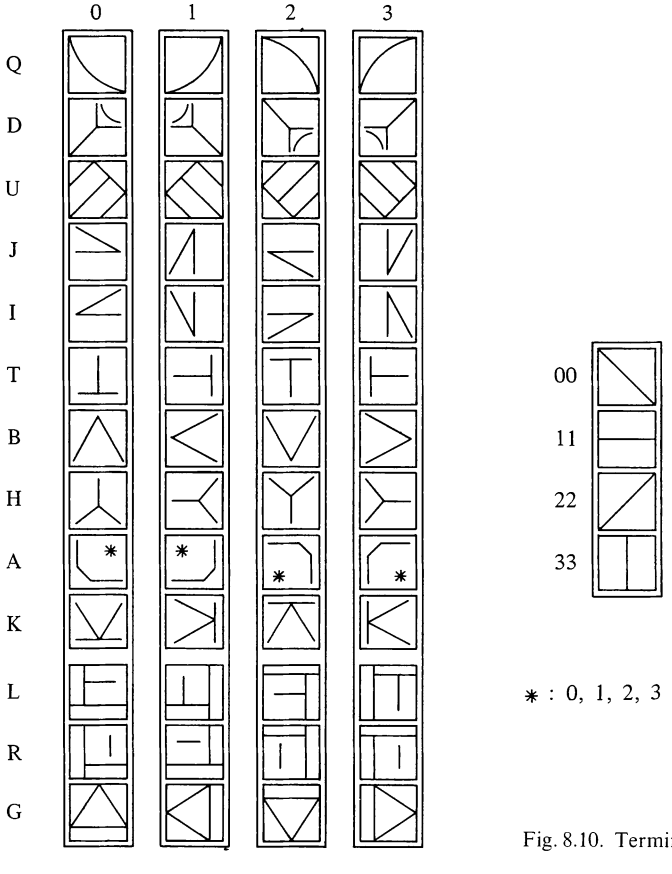


Fig. 8.10. Terminal primitives

and 270 degrees. The order of containment code and the 69 canonical combinations along with their containment codes are shown in Fig. 8.9. It is important to realize that the probability of occurrence of these 256 combinations is not the same at all. While the probability of occurrence of combinations such as $\langle 0,0,0,0 \rangle$ and $\langle 2,2,2,2 \rangle$ is high, the probability of occurrence of some others is almost zero. It is easy to see that a set of 256 features is too complex, and many of the combinations could be easily approximated by the others. Therefore, a set of 17 features is chosen to represent the 69 canonical combinations; i.e., a feature extractor has been developed to reduce the number of features while preserving the approximate syntax configuration of the fingerprints. The 56 primitives corresponding to the 17 features when the orientation of the 69 canonical combinations is taken into account are shown in Fig. 8.10.

8.2.4 Fingerprint Classifier

The complexity of filing fingerprints occurs due to the unbalanced classes of the Henry's classification system in which about 65 percent of the pattern are loops, 30 percent are whorls, and 3 percent are arch and others. Similarly the subclasses of the extended Henry system are also unbalanced which makes the system inadequate for an automatic classification system. It appears that Mair's family tree as shown in Fig. 8.11, which demonstrates the transitional topological structure of the fingerprint impressions, is more suitable for classification.

First-Level Classifier

The current classification scheme, which is based on the Henry system, has many drawbacks for an automatic classifier. In addition to the limited number fo classes, the distinction between the classes is very fine and ambiguous in certain cases. The Mair's family tree reveals how the syntax configuration of the 39 classes is so closely related. The advantage of Mair's family tree over the Henry's classification is the larger number of classes. For instance, patterns 1 through 15 are classified as whorl in Henry's classification. The classification of the 39 classes is not so trivial either, and a multi-level classifier seems more appropriate. The first-level classifier accepts a string of terminals as input, and either classifies the string into one of the seven classes or rejects the pattern as a fingerprint impression entirely.

Class I. This class contains the patterns whose global configuration consists of concave ridges on the top and delta configurations in each lower corner, as shown in Fig. 8.12.

Class II. This class includes patterns with concave ridges on the top, a delta in the lower right corner and a ridge with positive slope in the lower left corner, as shown in Fig. 8.12.

Class III. Here the global structure is similar to class II except that the delta configuration is located in the left corner and the slope of the ridges on the right side are negative, as shown in Fig. 8.12.

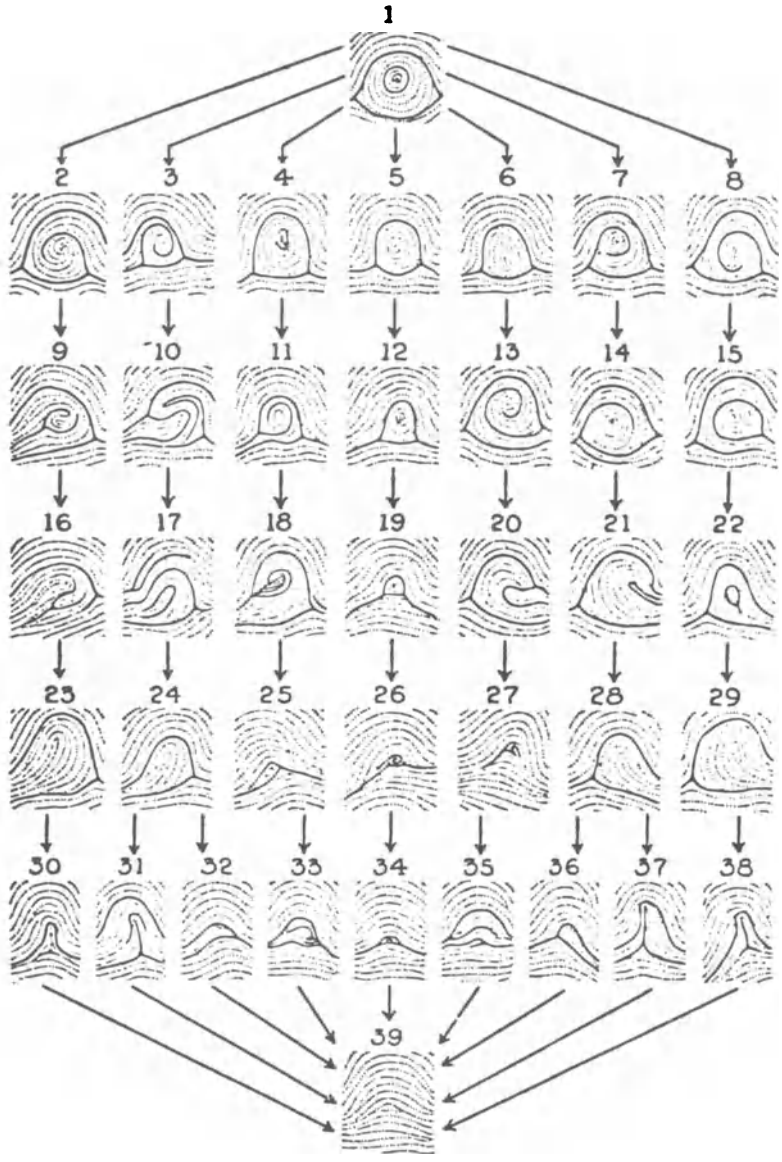


Fig. 8.11. A "family tree" of fingerprint types (modified from Mair's)

Class IV. The global structure of this class consists of concave ridges on top and an upside-down "*T*" in the lower part, as shown in Fig. 8.12.

Class V. Class *V* corresponds to plain arches whose syntax is simply a collection of concave ridges, as shown in Fig. 8.12.

Classes VI and VII. The global structure of these two classes, which are not explicitly included in the family tree, is a combination of two loops, as shown in Fig. 8.12.

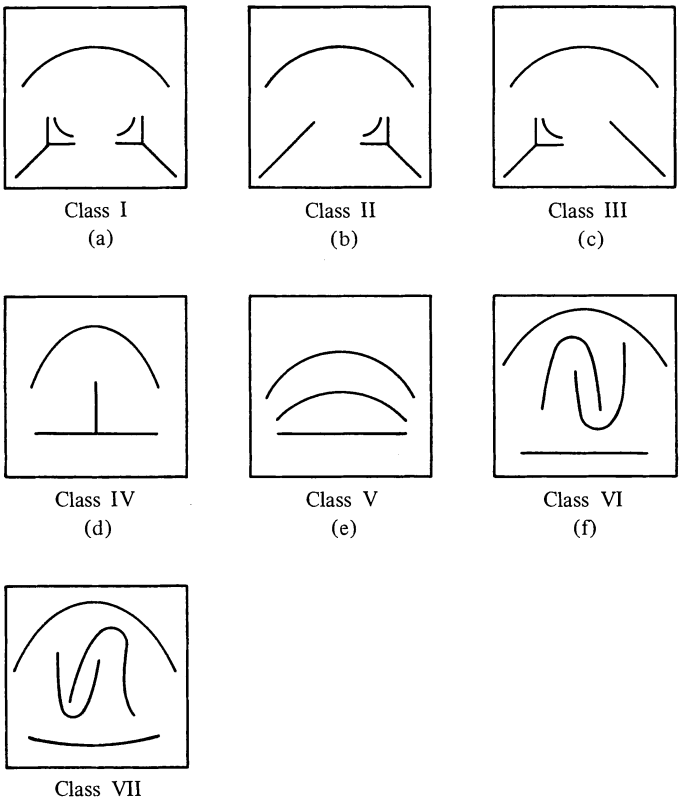


Fig. 8.12. Global configuration of the classes

All other patterns are rejected as fingerprint impressions. The classification scheme, which classifies the patterns into seven classes, is the first-level classifier. The first-level classifier classifies patterns 1 through 15 and possibly 18, 19, 20 into class I; patterns 16, 17, 23, 24, and 25 into class II; patterns 21, 22, 28, and 29 into class III; patterns 27, 30, 31, 36, 37, and 38 into class IV; and patterns 26, 32, 33, 34, 35, and 29 into class *V*. The classes could be named using the Henry's classification terminology such as whorl, tented arch, etc; however, the use of these terminologies may be very confusing because the definition of the classes in the two systems is not the same.

The goal of the classifier is to make the syntax analysis of the terminal strings as efficient as possible, so it is desirable to develop an algorithm to parse the string in a finite number of steps. This may be accomplished by a set of regular grammars; however, the regular grammars will have too many production rules because the production rules will have the form

$$N_i \rightarrow t_j N_{i+1} \quad \text{for} \quad i = 1, 64 \text{ and } j = 1, 56$$
$$N_{65} \rightarrow \lambda$$

where N_i's are nonterminals associated with each window of the sampling matrix, λ is the null terminal, and t_j's are the terminal primitives shown in Fig. 8.10. Although the syntax analysis of the regular grammars is deterministic, the number of production rules, which is on the order of 1460 for each grammar, eliminates the possibility of using this type of grammar. To decrease the number of production rules, a higher level grammar with self-embedding capability, namely a set of context-free grammars, is considered. To achieve a high reduction in the number of production rules and to keep the parsing reasonably efficient, a special context-free grammar is developed which has only two types of production rules. The first type, which may be self-embedding, associates the grammar with the decomposition of the global structure of the pattern. The second type of the production rules is used solely to decrease the number of production rules. If S, the starting symbol, corresponds to the main frame of the pattern, then replacement of S by the right-hand side of the first type production rules denotes the division of the pattern into four sampling squares. If the fingerprint pattern is considered as zero level, then the sampling matrix of four windows is called the first-level sampling matrix. Now, if each variable of the sentential string of length four is replaced by the right-hand side of the production rules of the first type, a sentenial string of length 16 is generated. The new sentential string implies the division of the first-level sampling matrix into a second-level sampling matrix with 16 windows. Applying the process of replacement to the second level will generate the third-level sampling matrix, which consists of 64 windows. The sentential string associated with the third-level sampling matrix has a length of 64. The third-level sampling matrix has the same rank as the sampling matrix of the features. Any further application of these types of production rules, which causes the length of the sentential string to exceed the terminal string, is not permissible. Obviously the application of the second type of production rules is possible. Therefore, the first level is the division of the pattern into four sampling squares; then the process of subdivision continues through the parsing levels. Hence, the pattern is subdivided into 4, 4^2, 4^3, and 4^4 in the levels 1, 2, 3, and 4, respectively. The sampling matrices of levels 4 and 3 are the input and output of the feature extractor, respectively. The process of subdivision of the pattern in each level is illustrated in Fig. 8.13.

The grammar of class I is a 4-tuple context-free grammar; i.e., $CI = (V_N, V_T, P, S)$ where $S \in V_N$ is the starting symbol, V_T is the set of features or primitives shown in Fig. 2.3, V_N is the set of nonterminal variables $(S, X_1, X_2, X_3, \ldots, N_1, N_2, N_3 \ldots)$, and P is the set of production rules presented in Table 8.1.

The 290 production rules of the grammar CI do not really decrease the efficiency of the recognizer when the total system is developed. The fact is that the grammars of the other classes are very similar to the grammar of the class I. As a result many production rules are duplicates of each other and will be eliminated in construction of the final grammar G, which is the union of all grammars. Also the parsing algorithm, which will be explained later, is not a brute-force syntax analyzer; that is, it will not make an exhaustive search. The parsing algorithm, with the knowledge of the basic structure of the derivation tree, is really a goal-oriented parser.

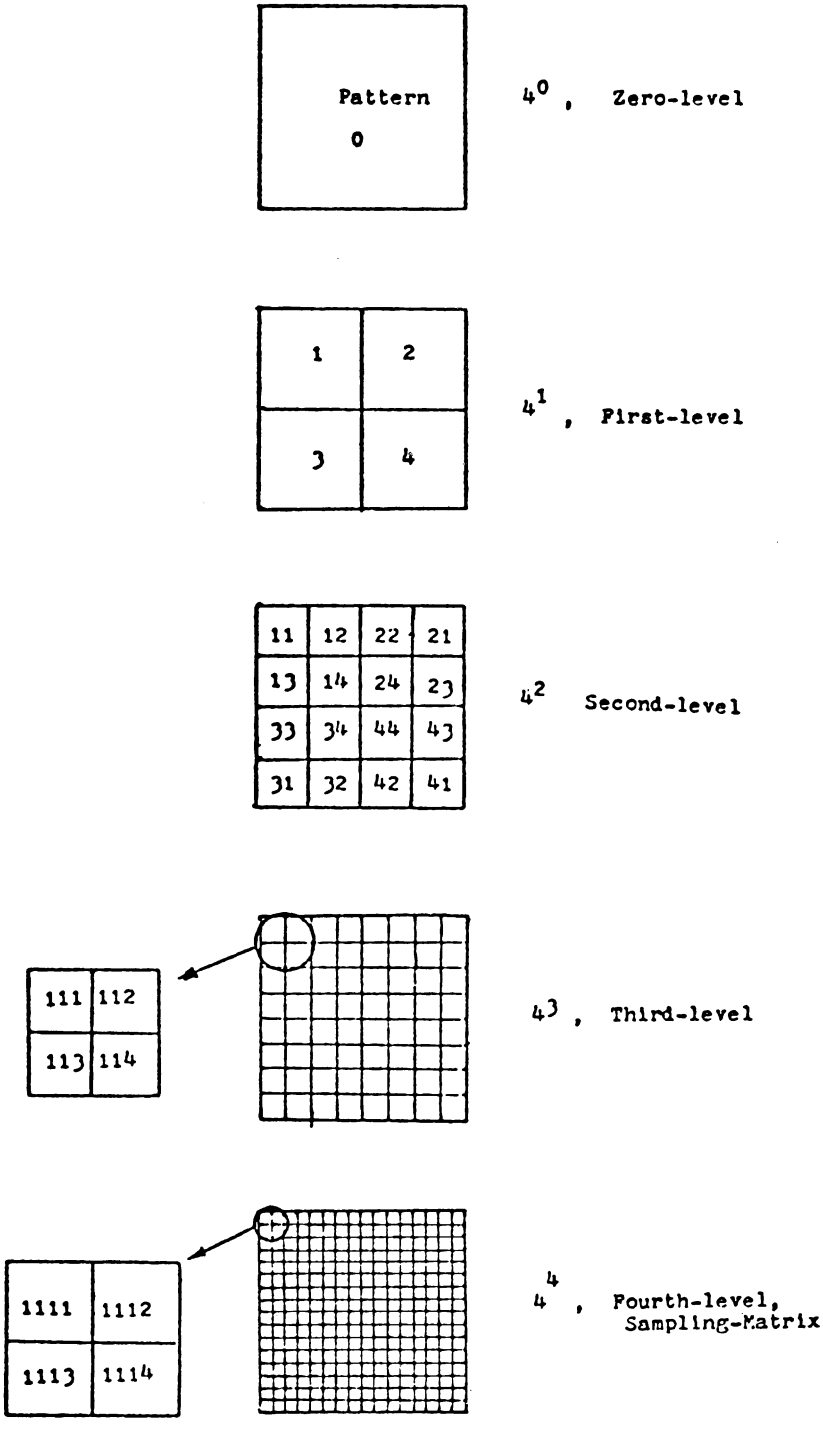


Fig. 8.13. Multilevel sampling matrices

Table 8.1. Grammar CI

$$CI = (V_{N1}\ V_{T1}\ P_1, S_1)$$

where

$$V_{N1} = \{S_1, X_1, X_2, \ldots, X_{435}, X_{445}, \ldots, N_1, N_{21}, \ldots N_{35}\}$$

$$V_{T1} = \{\text{Set of primitives shown in Figure 8.10}\}$$

And P_1:

First Type

1.	S1	→X1	X2	X3	X4	2.	X1	→X11	X12	X13	X14
3.	X11	→X111	X111	X111	X111	4.	X12	→X121	X121	X121	X121
5.	X13	→X131	X131	X131	X131	6.	X14	→X141	X141	X141	X141
7.	X2	→X21	X22	X23	X24	8.	X21	→X211	X211	X211	X211
9.	X22	→X221	X221	X221	X221	10.	X23	→X231	X231	X231	X231
11.	X24	→X241	X241	X241	X241	12.	X3	→X30	X325	X335	X345
13.	X325	→X321	X321	X321	X321	14.	X345	→X431	X341	X341	X341
15.	X3	→X316	X30	X336	X346	16.	X316	→X312	X312	X312	X312
17.	X336	→X332	X332	X332	X332	18.	X346	→X342	X342	X342	X342
19.	X3	→X317	X327	X30	X347	20.	X317	→X313	X313	X313	X313
21.	X327	→X313	X313	X313	X313	22.	X347	→X343	X343	X343	X343
23.	X3	→X318	X328	X338	X30	24.	X318	→X314	X314	X314	X314
25.	X328	→X324	X324	X324	X324	26.	X335	→X331	X331	X331	X331
27.	X338	→X334	X334	X334	X334	28.	X4	→X40	X425	X435	X445
29.	X425	→X421	X421	X421	X421	30.	X435	→X431	X431	X431	X431
31.	X445	→X441	X441	X441	X441	32.	X4	→X416	X40	X436	X446
33.	X416	→X412	X412	X412	X412	34.	X436	→X432	X432	X432	X432
35.	X446	→X442	X442	X442	X442	36.	X4	→X417	X427	X40	X447
37.	X417	→X413	X413	X413	X413	38.	X427	→X423	X423	X423	X423
39.	X447	→X443	X443	X443	X443	40.	X4	→X418	X428	X438	X40
41.	X418	→X414	X414	X414	X414	42.	X428	→X424	X424	X424	X424
43.	X438	→X434	X434	X434	X434	44.	X30	→P1	P1	N30	N3
45.	X30	→N2	P1	N30	N3	46.	X30	→N0	N4	P1	P1
47.	X30	→N1	N1	P1	P1	48.	X30	→N1	N1	N30	P1
49.	X30	→N1	N4	P1	N0	50.	X30	→N1	N1	N2	N0
51.	X30	→P1	N0	N2	N3	52.	X30	→N1	N0	N5	N3
53.	X30	→N2	N1	N2	N3	54.	X30	→N1	N0	N2	P1
55.	X30	→P1	N0	N3	P1	56.	X30	→N31	P1	N3	P1
57.	X30	→P1	N0	N3	N0	58.	X30	→P1	P1	N30	P1
59.	X30	→P1	N30	N30	P1	60.	X30	→N1	P1	N2	P1
61.	X30	→N1	P1	P1	N0	62.	X30	→P1	P1	P1	N0
63.	X30	→P1	N0	P1	P1	64.	X40	→N1	N1	P0	N0
65.	X40	→N4	N1	N0	N3	66.	X40	→N1	N5	P0	P0
67.	X40	→N4	P0	N3	N2	68.	X40	→P0	P0	N3	N30
69.	X40	→N1	N2	N3	N3	70.	X40	→N0	P0	N3	N30
71.	X40	→P0	N3	N3	N3	72.	X40	→N0	N1	P0	N1
73.	X40	→N0	N1	N0	P0	74.	X40	→N0	P0	N0	N2
75.	X40	→N2	N2	P0	N2	76.	X40	→P0	N1	P0	N31
77.	X40	→N1	P0	N3	P0	78.	X40	→P0	N2	P0	N3
79.	X40	→N1	P0	P0	N3	80.	X40	→P0	P0	N3	P0
81.	X40	←N0	N3	P0	P0	82.	X40	→P0	N1	P0	P0
83.	X40	→P0	N2	N3	P0	84.	X40	→N1	N1	P0	P0

Table 8.1. Grammar CI (continued)

Second Type

85.	X111→N12	86.	X111→N8	191.	X434→N20	192.	P1 →I3
87.	X121→N7	88.	X121→N12	193.	P1 →Q1	194.	P1 →R1
89.	X131→N15	90.	X131→X111	195.	P1 →D1	196.	P1 →H2
91.	X141→N7	92.	X141→N11	197.	P1 →R0	198.	P1 →N33
93.	X141→N13	94.	X141→N15	199.	P0 →A2	200.	P0 →J3
95.	X141→N28	96.	X141→H2	201.	P0 →A1	202.	P0 →N33
97.	X141→A3	98.	X141→H1	203.	N0 →00	204.	N1 →11
99.	X141→12	100.	X141→N26	205.	N2 →22	206.	N3 →33
101.	X141→A2	102.	X141→N34	207.	N4 →00	208.	N4 →11
103.	X141→N35	104.	X211→N6	209.	N5 →11	210.	N5 →22
105.	X211→N14	106.	X221→N7	211.	N6 →N4	212.	N6 →33
107.	X221→N14	108.	X231→X211	213.	N7 →N6	214.	N7 →22
109.	X231→N16	110.	X241→X141	215.	N8 →N5	216.	N8 →33
111.	X321→N7	112.	X321→N19	217.	N9 →N4	218.	N9 →22
113.	X321→N23	114.	X321→N22	219.	N10 →00	220.	N10 →22
115.	X331→N7	116.	X331→Q3	221.	N10 →33	222.	N11 →N18
117.	X331→N14	118.	X331→N16	223.	N11 →N20	224.	N12 →A3
119.	X331→N24	120.	X341→N7	225.	N12 →Q3	226.	N12 →I2
121.	X341→N11	122.	X341→N17	227.	N12 →J2	228.	N13 →B0
123.	X341→N23	124.	X341→T0	229.	N13 →H0	230.	N13 →G0
125.	X341→B0	126.	X341→A0	231.	N14 →Q2	232.	N14 →J2
127.	X341→A3	128.	X341→N34	233.	N14 →A2	234.	N14 →J0
129.	X312→N9	130.	X312→Q0	235.	N15 →J1	236.	N15 →J3
131.	X332→N27	132.	X332→N32	237.	N15 →I0	238.	N16 →I1
133.	X332→Q1	134.	X342→N29	239.	N16 →I3	240.	N17 →J1
135.	X342→N10	136.	X313→N9	241.	N17 →I3	242.	N18 →Q1
137.	X313→N18	138.	X313→N14	243.	N18 →Q3	244.	N19 →N18
139.	X343→N7	140.	X343→L1	245.	N19 →Q0	246.	N20 →Q0
141.	X343→T1	142.	X343→Q0	247.	N20 →Q2	248.	N21 →I0
143.	X343→J0	144.	X343→I3	249.	N21 →Q2	250.	N22 →J0
145.	X343→Q3	146.	X343→N25	251.	N22 →J2	252.	N22 →I0
147.	X314→N7	148.	X314→N20	253.	N23 →R1	254.	N23 →R3
149.	X314→J0	150.	X324→N9	255.	N24 →L1	256.	N24 →N26
151.	X324→N21	152.	X334→N18	257.	N24 →J1	258.	N25 →L0
153.	X334→N7	154.	X334→N20	259.	N25 →Q0	260.	N25 →T1
155.	X334→N28	156.	X334→N22	261.	N26 →R0	262.	N26 →R2
157.	X421→N7	158.	X421→N19	263.	N27 →J0	264.	N27 →R0
159.	X421→N23	160.	X421→I0	265.	N28 →N23	266.	N28 →I3
161.	X431→N7	162.	X431→N14	267.	N29 →R3	268.	N29 →Q1
163.	X431→N24	164.	X431→N16	269.	N30 →22	270.	N30 →33
165.	X431→Q1	166.	X431→Q0	271.	N31 →11	272.	N31 →33
167.	X441→N7	168.	X441→N11	273.	N32 →00	274.	N32 →22
169.	X441→D1	170.	X441→B0	275.	N33 →N11	276.	N33 →N22
171.	X441→J1	172.	X441→N28	277.	N33 →N24	278.	N33 →N34
173.	X441→N34	174.	X412→X312	279.	N33 →N35	280.	N33 →D1
175.	X432→X332	176.	X442→X342	281.	N33 →H3	282.	N33 →A3
177.	X413→N4	178.	X423→Q0	283.	N33 →N13	284.	N34 →L0
179.	X423→I1	180.	X443→Q2	285.	N34 →L1	286.	N34 →L2
181.	X443→N8	182.	X443→N25	287.	N34 →L3	288.	N35 →I0
183.	X414→N7	184.	X414→Q0	289.	N35 →T2	290.	N35 →T3
185.	X414→J0	186.	X424→N7				
187.	X424→N21	188.	X434→N6				
189.	X434→N20	190.	X434→N28				

It is very clear from Fig. 8.12 that the only difference between the global structure of the fingerprint patterns in class I and class III is in the lower right corner. Therefore the production rules corresponding to the other three squares will be the same in both grammars. It is this kind of similarity of the patterns which allows a significant reduction in the total number of production rules. The grammar CIII for the class III can be considered as a modified version of the grammar CI. In other words, the two grammars share the rewriting rules corresponding to X_1, X_2, and X_3 associated with the sampling squares 1, 2, and 3 in the first level. The grammar CIII for the class III is given in Table 8.2.

Similarly, the difference between the global structure of the classes I and II is only in the lower left corner. The global structure of the classes IV and V is also similar to the class I except for the lower part of the pattern. Therefore it is very easy to see that by relaxing the exact representation of global structure of the patterns, the number of production rules can be decreased significantly, which contributes to the efficiency of the system.

Table 8.2. Grammar CIII

$$\mathrm{CIII} = (V_{N3}, V_{T3}, P_3, S_3)$$

Where:

$$V_{N3} = V_{N1} \cup \{Y_4, Y_{41}, Y_{42}, Y_{43}, Y_{44}\}$$
$$V_{T3} = V_{TI}$$

And P_3:

1.	S3	→X1	X2	X3	Y4
2.	Y4	→Y41	Y42	Y43	Y44
3.	Y42	→Y421	Y421	Y421	Y421
4.	Y43	→Y431	Y431	Y431	Y431
5.	Y44	→Y441	Y441	Y441	Y441
6.	Y411	→N4			
7.	Y411	→Q0			
8.	Y421	→N4			
9.	Y421	→N20			
10.	Y421	→J0			
11.	Y431	→Y421			
12.	Y431	→33			
13.	Y441	→Y431			

Productions corresponding to X_1, X_2, and X_3 are presented in Table 8.1.

The global configuration of classes VI and VII is different from the previous classes. In this case, the global information of the upper sections is also important for classification. Fully detailed information about the grammars may be found in [8.19].

Parsing or syntax analysis is the process of analyzing the terminal strings in terms of a grammar. A top-down parser synthesizes the pattern by applying the production rules from the root of the tree and proceeding downward, while the bottom-up parser tries to synthesize the pattern by applying the production rules in reverse order; i.e., it starts from the terminal string and proceeds upward by replacing the terminal and nonterminal elements by nonterminals until the sentential string reduces to the root of the tree. There are several top-down and bottom-up parsing algorithms for context-free grammars which seem to be applicable for syntactic analysis of fingerprint patterns [8.22, 23].

Here syntax analysis means that each terminal string is parsed under control of the seven grammars. Obviously, this is an inefficient technique because each parser attempts to generate or recognize the whole terminal string of length 64. One of the impediments of pattern recognition is the total cost of the process which is simply the time requirement of digitizing, preprocessing, feature extraction, and parsing which might involve some backtracking. To minimize this cost, the concept of sequential decision procedure [8.24] is used in the system. Sequential decision procedure means to extract features sequentially and stop the decision process whenever the pattern is classified into a class or rejected as fingerprint. Obviously, in this case, the order of taking features is very important. The sequential processing in syntax analysis implies parsing of the sentential string after adding each new terminal variable. The sequential procedure is particularly suitable for grammars which have inherently independent subderivation trees. For instance, the grammars of classes I through VII have four subtrees which are independent of each other and are associated with the windows of the first-level sampling matrix. The sequential parsing is accomplished by parsing the subtree associated with X_1 or window 1. If the subtree belongs to the grammar, the second window will be processed; otherwise, parsing under the current grammar will be terminated. The sequential processing includes the preprocessing, feature extraction, and parsing of each subtree. Therefore, it may happen that a pattern will be rejected as a fingerprint impression by syntax analysis of the first window space only. A pattern is accepted as a member of a class when all four subtrees are sequentially accepted by the grammar of the class.

A top-down parsing algorithm is developed because the parsing is sequentially controlled by the grammar, and the parsing of each subtree is done in two or three levels. Therefore, the goal of the parser is to prove that the substring belongs to the class of that grammar. In case of a failure, the process is repeated under the control of another grammar until all seven grammars are exhausted. Because the number of derivation levels of each subtree is very small, the backtracking, which usually requires

Table 8.3. Summary of classification

	True classification								
	Classes	I	II	III	IV	V	VI	VII	Reject
Machine classification	I	21	1	2					
	II	1	10						
	III			12					
	IV				11				
	V					4			
	VI						9		
	VII							7	
	Reject	3	3	1			3	4	1

a lot of time and memory, is not needed at all. Therefore, any time the parsing of a subtree fails, another alternative subtree will be considered until all possible subtrees are exhausted [8.20].

The classification of the 92 fingerprint patterns was simulated on CDC-6500 Computer at Purdue University. The classification result is summarized in Table 8.3.

The patterns which were rejected as fingerprint impressions were either improperly scanned or too large for the sampling matrix and one nonfingerprint impression was included. The simulation of each pattern took about 25 s for digitized binary detection; about 15 s for scanning, tracing, and feature extraction; and about 0.152 s for parsing.

Second-Level Classifier

The first-level classifier with only seven classes is not an adequate system because the purpose of an automatic classifier is to generate as many classes as possible. Therefore a second-level classifier is developed in order to further subclassify each class of the first-level classifier. However, since the global structure of each class is very similar to each other, any subdivision will be based on small differences in the curvature of the ridges. In fact, the development of a set of context-free grammars with disjoint subclasses seems impossible. Also, it is most desirable to use a scheme which uses the set of primitives of the first-level classifier. Thus, the second-level classifier is designed to use the same primitives as the first-level classifier, while it has the capability of solving ambiguous decision, i.e., the recognition of a string by more than one grammar. This extra dimension can be achieved by assigning probability to production rules along with a decision rule as suggested by FU [8.25, 26] and LEE [8.27]. The stochastic syntax-directed approach means that, given a finite set S composed of stochastic grammars $S=\{G_1, G_2, \ldots, G_n\}$ where the language $L(G_i)$ generated by grammar G_i is a collection of all possible structural patterns of class S_{G_i} and G_i defines a probability distribution, overall strings of $L(G_i)$, i.e., $p(x|G_i)$, is the probability associated to the generation of string x by grammar G_i. The stochastic syntax analysis scheme may classify an input pattern into one of n classes.

However, in case of ambiguity, the Bayesian maximum-likelihood decision rule [8.28] can be used to resolve the ambiguity, i.e., the string will be classified into the class S_{G_i} when $p(G_i|x)$ is a maximum and

$$P(G_i|x) = P(G_i)P(x|G_i)/P(x) \tag{8.1}$$

and $P(G_i)$ is the a priori probability of class S_{G_i}. Therefore, with the stochastic syntax model a measure of acceptance is assigned to each string which can be used to eliminate any ambiguity.

The second-level classifier is also based on Mair's family tree which is not quite complete but it does provide guidance for subclassification of the patterns. For instance, patterns 1 through 15, 18, and 19 of the family tree are assigned to class I under the first-level classifier. It is necessary to assign a different D_i for each of the above classes in order to subclassify the class I further. The problem is how to determine D_i, the production probability of each grammar. The statistical distribution of the patterns usually can be found by studying a large sample set; however here, the sample of 92 fingerprints is too small to obtain any reliable statistical distribution. Hence, among the procedures presented in [8.27], the supervised weighted approach, which is the most suitable technique for the patterns with small sample size, may be used. Since context-free languages [8.29] obey the statistical inference of Mark'ov processes, then the production probability D can be inferred where

$$D = \{\hat{p}_{ij} | A_i \to \eta_j \in P \text{ and } A_i \in V_N, \eta_j \in (V_N \cup V_T)^*\} \tag{8.2}$$

$$P_{ij} = n_{ij} / \sum_j n_{ij} \tag{8.3}$$

where $\hat{P}_{ij}$ is the maximum likelihood estimate for P_{ij} [8.29] and n_{ij} follows a multi-normal distribution with P_{ij}; and

$$n_{ij} = \sum_{x \in S_x} n_x \mathrm{N}_{ij}(x) \tag{8.4}$$

and $\mathrm{N}_{ij}(x) = \# A_i \to \eta_j$ used in x. S_x is the given set of sample strings, n_{ij} is the number of times production $A_i \to \eta_j$ is used in the generation of all strings in the estimates. n_x is the number of times string x appears in the genration of S_x. Σ is the sum over all the right parts η_j for a given A_i as left part. Therefore

$$P_{ij} = \sum_{x \in S_x} n_x \mathrm{N}_{ij}(x) / \sum_j \sum_{x \in S_x} n_x \mathrm{N}_{ij}(x) \tag{8.5}$$

and in the supervised weighted approach the inferred probability $\hat{P}_{ij}$ defined in (8.1) approaches P_{ij}, the actual probability [8.27].

In the second-level classifier, because the difference among the patterns is mainly due to the curvature of the ridges, the sampling matrix of 16×16 containing the predominant slopes of the ridges, namely 0, 1, 2, and 3, is used as terminal primitives. Because of our page limit the production rules of the grammars, which are the same for all the classes with different D's, are not

Table 8.4. Summary of the second-level classifier

No.	Pattern number	First-level classifier		Second-level classifier	
		True	Machine	True	Machine
1.	1	I	I	3	5
2.	2	I	I	15	15
3.	8	I	I	7	7
4.	12	I	I	15	2
5.	20	I	Reject	7	5
6.	26	I	I	14	14
7.	27	I	Reject	1	14
8.	28	I	I	18	18
9.	30	I	I	7	7
10.	34	I	I	18	18
11.	35	I	I	15	15
12.	37	I	II	11	12
13.	40	I	I	11	11
14.	41	I	I	14	14
15.	46	I	I	13	13
16.	52	I	I	18	18
17.	53	I	I	11	11
18.	55	I	Reject	3	3
19.	60	I	I	13	13
20.	64	I	I	15	15
21.	72	I	I	9	9
22.	74	I	I	15	15
23.	79	I	I	13	13
24.	89	I	I	15	15
25.	92	I	I	9	9

presented in full detail; see [8.19]. The grammars are ambiguous in the sense that all the input strings are accepted by the grammars; however, the ambiguity may be solved using (2.4.1) of [8.19]. To determine maximum $P(G_i|x)$ is the same as searching for maximum $P(G_i)P(x|G_i)$ because $P(x)$ is the same for all grammars. Also, assuming $P(G_i)$ is the same for all subclasses due to the small sample size of the patterns, one really needs to look for maximum $P(x|G_i)$. Hence, any input strings x_i are parsed under control of all grammars and the string x belongs to the subclass i when $P(x|G_i)$ is maximum.

The result of the computer simulation, shown in Table 8.4, illustrates the subclasses of class I of the first-level classifier. The misclassification of patterns 20 and 27 is due to excessive noise of the patterns, and patterns 1, 12, and 37 are misclassified because subclasses 2, 3, and 11 are very similar to the subclasses 5, 15, and 12, respectively.

8.3 Tree Grammar Approach to Fingerprint Pattern Recognition

The first- and second-level grammars have the potential of generating about 40 classes which seems to be inadequate for identification and filing systems; however, the techniques may be valuable as an intermediate step in a fully

automatic system. The desired number of classes can be achieved only with a technique which utilizes the features of the fingerprint patterns such as abrupt ridge ending, short ridge, branches, and mergers. The relationship of the minutiae is so unique that although each fingerprint pattern has about 100 minutiae, as many as a dozen is considered sufficient to identify an impression [8.3].

The classification of the fingerprints with minutiae as the set of primitives is much more difficult. In this case, the general view is that rather than using string grammars with blank and directive terminal variables, a higher level language is more suitable for classification. Recently many people have argued that string grammars are not powerful enough to describe high-dimensional patterns. KIRSCH [8.30] and DACEY [8.31] have attempted to use a two-dimensional grammar for a class of 45 degree right-triangle patterns located in a subdivided square unit of a plane such as the sampling matrix, PFALTZ [8.32] proposed the use of web grammars which are essentially representations of sentences by directed graphs, FEDER [8.33] suggested the use of plex grammars which basically describe the connectivity of the picture components, PAVLIDIS [8.34] suggested the use of graph grammars, and FU and BHARGAVA [8.35] proposed the use of tree grammars for recognition of high-dimensional patterns. So far, there is no rigorous technique to evaluate the power of these high-dimensional grammars and since regular tree grammars are really generalization of string grammars, i.e., a string grammar is a tree grammar when the rank of all variables are zero or one, it therefore becomes natural to use tree grammars for extension of the classifier.

8.3.1 Tree Classifier

In this case, the feature extraction which is basically extraction of all minutiae of fingerprint patterns is more crucial and consequently the system uses a picture enhancement and postprocessing algorithms in order to eliminate extraction of erroneous minutiae.

Picture Enhancement

Since the moving average technique was found inadequate for extraction of the minutiae primitives and it has been shown [8.36] that if the spot scanner has a Gaussian distribution, then the following relationship exists between the true picture $g(x, y)$ and the digitized image $f(x, y)$:

$$g(x, y) = (1 - K^2 \nabla^2) f(x, y)$$

$$\nabla^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2}$$

where ∇^2 is the Laplacian operator, and K^2 is a positive constant. Therefore the picture enhancement is achieved by applying the Laplacian operator, which is basically an approximation of each point by a weighted sum of the grey levels of a certain region, to the grey level of each point of the pattern.

Preprocessing

Here the preprocessing technique is the same as in the case of string grammars; however, the sampling matrix is not exactly the same due to the higher accuracy needed for tree grammars. Since the representation of about 100 minutiae of each fingerprint by a string or a tree is an extremely difficult task, the patterns are decomposed into smaller regions which makes the representation of each region feasible. Although it is desirable to decompose the patterns according to their distinct and repetitive components, fingerprint impressions have no natural decomposable substructure. Therefore, it appears that any arbitrary decomposition of the patterns may be as good as any other one. For decomposition one must be careful that the components will not be too small, which would require more bookkeeping and may not have sufficient

```
X= 4    Y= 3    N=15
    12345678901234567890123456789012345678901234567890
   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1 X                                                  X   1
 2 X ***      ****   ****  *****    ***   *****  **** X   2
 3 X *       ****  *****   ****   ****   *****  ***** X   3
 4 X         ***   *****  ******  *****  ****   X   4
 5 X     **** *  ******  ******  *****   ****  ****   X   5
 6 X    *****   ******  ******  ****   ******  **** * X   6
 7 X  ******   *****   *************  ******  ****  * X   7
 8 X *****    *****    **** *******   ****   ***** ** X   8
 9 X  ****   *****   ******  *****   *****  ***** *** X   9
10 X   **   ******  ******  ****     ***** ****** *** X  10
11 X       *****   ******   ****  *  ***  *****   *** X  11
12 X     ******   *****   ***    **  **   *****  **** X  12
13 X    *****    *****    ***   **  ****  ****  ***** X  13
14 X  ******    ****    *****   **  ***  *****  ****  X  14
15 X ******    ****    ****     ******* ****** ***** X  15
16 X ** **   *****    ***        *****  ***** *****   X  16
17 X **     ***        ***       ***** ***** ******   X  17
18 X *    ****    *** **         ***  ************  * X  18
19 X     *****    *****         ***   ************ ** X  19
20 X  * **** *   ****     *    ****  ****** *****  ** X  20
21 X *******    ****     **    ***   ****  ****** *** X  21
22 X ******    *****    **    ****  *****  ***** **** X  22
23 X ***      *****    **    ****  *****  ************ X  23
24 X ***     ***     ***    ****  *****  *****  ****  X  24
25 X **     *****   ***    ****  *****  ****** *****  X  25
26 X       *****   ****   ****   *****  ****  ****    X  26
27 X     *****    ****   *****  *****  ***** ******   X  27
28 X    ****     ****    ***   *****  ****   ****   * X  28
29 X  *****     ****    ****  ************  ****    * X  29
30 X *****    *****   *****   *****  **** ******  *** X  30
31 X ****   ******   *****   *****  **** ******  **** X  31
32 X **    *****    *****   ****   ***** *****   ***  X  32
33 X *    *****    *****   *****  ***** ******  ****  X  33
34 X      ****    *****   *****  *****  *****  ****** X  34
35 X    *****   ******   ****** *****  ******  **** * X  35
36 X   *****    *****   *****   ****  ************ ** X  36
37 X  *****    ****    *****  *****  ******* **** *** X  37
38 X ****    *****    *****  *****  ******  **** *** X  38
39 X ***     ****    ****    *****  ******  ********* X  39
40 X *      ****   *****   ******  ******  ********** X  40
41 X      *****   *****   ******  ******  *****    ** X  41
42 X     ****    *****    *****  ******   *****   *** X  42
43 X   *****    *****   ****     *****   ******  **** X  43
44 X  *****    *****   *****   ******   *******  **** X  44
45 X  ****    ****    *****   ***** ** ******   ***** X  45
46 X ***    *****    *****   ******  ****** *  ****** X  46
47 X **    *****    *****    *****  *******   ******* X  47
48 X *    *****    ******   ****    ******   *******  X  48
49 X     *****    ******  ******  *******   *******   X  49
50 X                                                  X  50
   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Fig. 8.14. Window No. 15 of pattern No. 8

terminal primitives for a meaningful classification, and also the components should not be too large, i.e., containing too many terminal primitives. Hence a sampling matrix of 4×4 with each window containing 48×48 binary points was chosen for decomposition of the fingerprint patterns. The location of the sampling matrix may be very important for classification because any significant change in location of the sampling matrix may cause some of the minutiae to transfer from one window to another leading to misclassification of the pattern. This kind of event may be prevented by a set of rules for selection of the sampling matrix for each class or by defining buffer zones for separating the windows of the sampling matrix by which the terminal primitives on the dividing line will be shared by the neighboring windows. Figure 8.14 shows a sampling window for tree grammar approach.

```
    123456789012345678901234567890123456789012345678901234567890
   ************************************************************
 1 * +        +      +      +        +      +      +      +   *  1
 2 * 5        6      6      5        6      6      6      *   *  2
 3 * .      5      57      6      67      5      5      *   *  3
 4 *      67      6      6      5      5      6      *   *  4
 5 *    677     67      *77777777      6      5    + *   *  5
 6 *   6      6      6              67      6    5 *   *  6
 7 *  57      6      5              5      6      6 *   *  7
 8 *  4      6      B33335          6      6      6   *   *  8
 9 *   5     6      67     5      5      6      5   *   *  9
10 *   .    6      6      5      + 5      6      5   *   * 10
11 *      6      6      6      6  5     5      6   *   * 11
12 *    67      6      57     6  6     5      6   *   * 12
13 *   6      6      6     5  5     6      5   *   * 13
14 *  67      6      67     *7777    6      6   *   * 14
15 * 5      6      6      5      5      6      *   *  15
16 * 5      67      5      5      *777777      *   *  16
17 * 5      6      6      5      6            + *   *  17
18 * .    67      5777      6    5            6 *   *  18
19 *    6      6      +    6      6            6 *   *  19
20 *   6      6      6    5      B333335      5   *   *  20
21 * 577      6      6      6      6      *7777777   *   *  21
22 * 5      6      6      6      6      6           *   *  22
23 * 5      6      6      6      6      B33335      *   *  23
24 * 5      6      67     6      6      6      5    *   *  24
25 * .      6      6      6      5      6      6    *   *  25
26 *      6      6      6      6      5      6      *   *  26
27 *    67      6      5      *7777777      5      + *   *  27
28 *   6      6      6      6              6      5 *   *  28
29 *  67      6      6      B333335      6      6 *   *  29
30 * 5      67      67      6      5      67      67   *   *  30
31 * 5      67      6      6      6      5      5    *   *  31
32 * 5      6      6      6      6      6      6      *   *  32
33 * .    5      6      6      6      5      6      *   *  33
34 *      6      6      6      6      6      5      *   *  34
35 *    67      57      6      5      *77777777      *   *  35
36 *   6      6      6      6      6              *   *  36
37 *  6      6      6      57      B3333335      *   *  37
38 * 5      57      6      6      6      5      *   *  38
39 * 5      6      6      6      6      6      *   *  39
40 * .      6      67      67      6      B3333335   *   *  40
41 *    67      6      5      6      5      5    *   *  41
42 *   6      6      6      5      6      6    *   *  42
43 *  67      6      67      6      57      5    *   *  43
44 *  5      6      6      6*77777777      6    *   *  44
45 *  6      6      6      6              6    *   *  45
46 * 5      67      6      5              6    *   *  46
47 * 5     6      6      6              6    *   *  47
48 * .    5      5      5              5    *   *  48
49 *      .      .      .              .    *   *  49
50 *                                         *   *  50
   ************************************************************
    123456789012345678901234567890123456789012345678901234567890
```

Fig. 8.15. Chain coded window after postprocessing

Postprocessing

The task of postprocessing is to consider the output of the preprocessor and decide whether to connect a pair of ridges to each other or not, i.e., it essentially distinguishes whether the discontinuity of a ridge is due to erroneous tracing or gaps because of ink when the fingerprint is taken. While the human brain can easily interpret the gaps of the ridges and extract the true global configuration of the window, the right interpretation of the gaps is a very tedious task by computers. The postprocessing algorithm considers all connecting alternatives of the ridges and ensures that the direction of the connecting line is according to the global configuration of the window. The algorithm [8.19] is based on the criteria that in each fingerprint the ridges are generally continuous lines, run mostly in parallel, any new start of sudden termination is mostly located between two continuous ridges, and finally the degree of variation of curvature of the ridges is very gradual except around deltas and the center of loops and whorls. Figure 8.15 shows the output of the postprocessor corresponding to the window shown in Fig. 8.14.

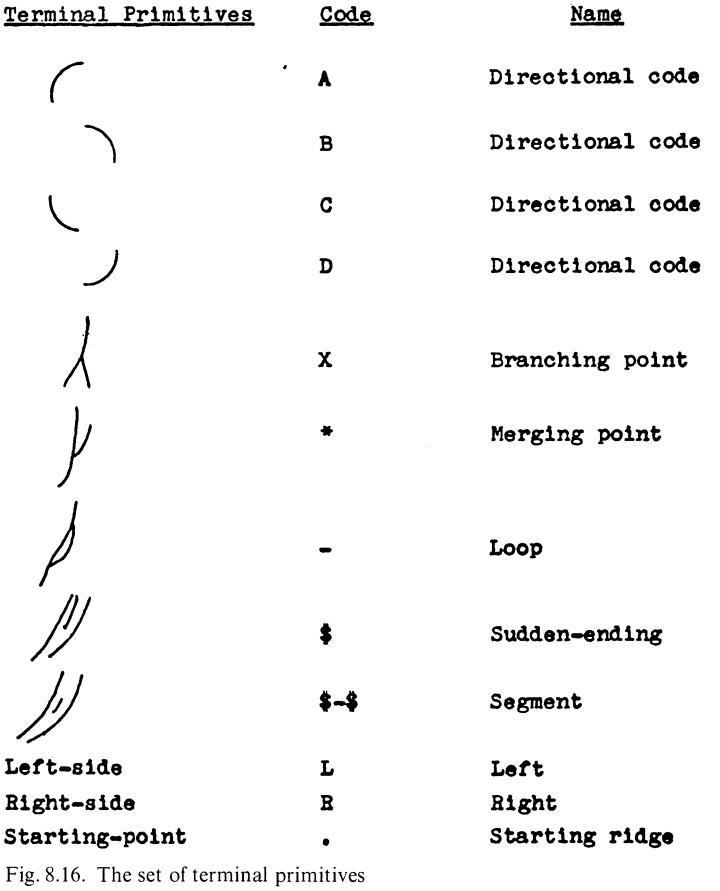


Fig. 8.16. The set of terminal primitives

Feature Extraction

The minutiae of fingerprint impressions define the patterns so uniquely that if 50 minutiae are considered in each pattern and if it is assumed that each minutia can be either a branch or termination, then 2^{50} or about 10^{15} unique fingerprints can be expected. Therefore, if all variations of minutiae and their location within the pattern area are taken into account, the number of unique fingerprint impressions can reasonably be considered infinite for all practical purposes. The primitive set shown in Fig. 8.16 has been selected based on the type of information associated with the ridge configuration within each window. The observation indicates that

1) continuous ridges which are entering from adjacent windows, running through the window, and leaving the window for other windows, have no minutia and thus no information for classification;

2) ridges with sudden ending, branching, merging, segmented ridge, or combination of these, have a large amount of information for classification;

3) positioning of the ridges with respect to each other within a window has some information for classification.

A set of simple rules has been used for extraction of the terminal primitives. For instance, a termination is considered as a true sudden ending if its coordinates are located well within the window area.

8.3.2 Tree Grammar

It is true that the goal of the classifier is to generate as many classes as possible. However, the generation of all the tree grammars is a very cumbersome task, unless an inference procedure is developed to infer new grammars or their recognizers whenever unclassified fingerprint patterns are found. The inference procedure and inferred tree grammars are explained after a typical tree grammar is presented.

Example. The grammar G_{t3} is a typical tree grammar which generates all windows with a set of continuous ridges along with one ridge having a branch minutia

$$G_{t3} = (V, r, \mathrm{P}, \mathrm{S})$$

where

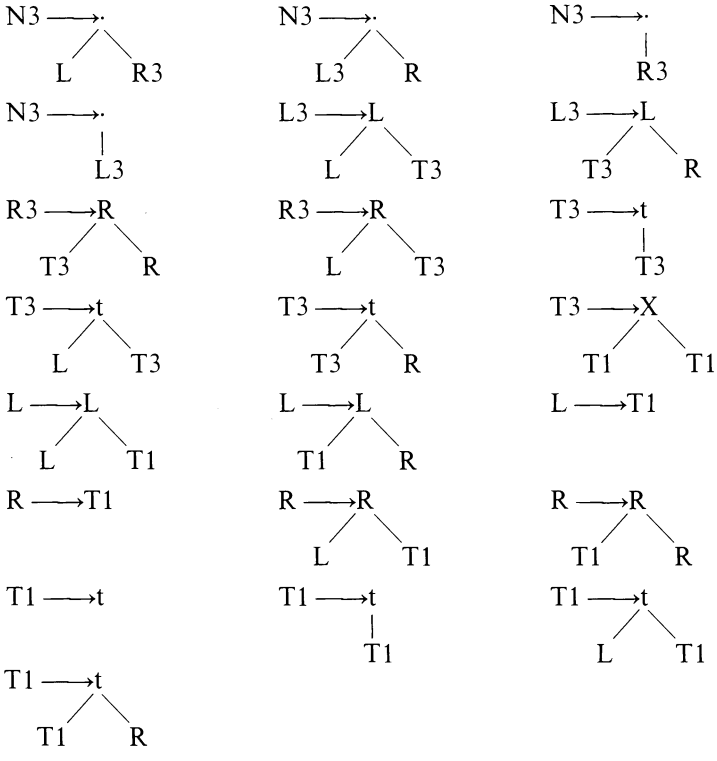
$$\mathrm{S} = \{\text{starting symbol}\} = \{\mathrm{N3}\}$$

$$V = \{\mathrm{N3}, \mathrm{T1}, \mathrm{R}, \mathrm{L}, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \ ,.,.,\mathrm{T\,3}, \mathrm{R\,3}, \mathrm{L\,3}, \mathrm{X}\}$$

$$r(.) = r(\mathrm{L}) = r(\mathrm{R}) = r(\mathrm{A}) = r(\mathrm{B}) = r(\mathrm{C}) = r(\mathrm{D}) = r(\mathrm{R3}) = r(\mathrm{L3}) = 1, 2$$

$$r(\mathrm{X}) = 2$$

and P:



where $t=\{A, B, C, D\}$. The tree automaton corresponding to this grammar accepting windows of continuous ridges with one ridge having a branch minutia is

$$M_{t3}=(Q, f_A, f_B, f_C, f_D, f_R, f_L, f_{\cdot}, F)$$

$$Q=\{q_{T1}, q_{T3}, q_{N3}, q_R\}$$

$$F=\{q_{N3}\}$$

$$\begin{aligned} f:\quad & f_X(q_{T1}, q_{T1})=q_{T3} & f_t(q_{T1}, q_{T3})&=q_{T3} \\ & f_L(q_{T1}, q_{T3})=q_{T3} & f_R(q_{T1}, q_{T3})&=q_{T3} \\ & f_L(q_{T1}, q_{T1})=q_{T1} & f_R(q_{T1}, q_{T1})&=q_{T1} \\ & f_t(q_{T1}, q_{T1})=q_{T1} & f_{\cdot}(q_{T1}, q_{T3})&=q_{N3} \\ & f_{\cdot}(q_{T3})=q_{N3} & f_t(q_{T3})&=q_{T3} \\ & f_t(q_{T1})=q_{T1} & f_t&=q_{T1} \end{aligned}$$

where $t=\{A, B, C, D\}$. All other transitions lead to q_R the rejecting state. Depending on the final state of the tree automaton, the window is classified as continuous ridges with one ridge having a branch minutia or is rejected as a member of this class.

8.3.3 Grammatical Inference

Learning capability is the most important part of a pattern recognition system whenever a practical application with a large number of samples is under investigation. Learning in the sense of linguistic pattern recognition means inferring new grammars, i.e., inferring new production rules of unknown grammar G, based on the observation of a finite set of strings S from $L(G)$ and possibly also a finite set of strings S' from the complement of $L(G)$ [8.37]. The generation of tree grammars for such a large number of classes of fingerprints is a very tedious task unless an inference procedure is developed.

Basically there are three approaches to grammatical inference procedure, i.e., 1) grammatical inference by enumeration, 2) grammatical inference by induction, and (3) grammatical inference by a set of heuristic rules. Also an inference procedure for tree grammars is presented by BHARGAVA and FU [8.38]. A typical block diagram of grammatical inference procedure is shown in Fig. 8.17. The purpose of the system is to infer a grammar for each set of input patterns, i.e., a grammar G_t is to be inferred which would be capable of generating at least the given tree sample patterns $S=\{X_1, X_2, X_3, \ldots, X_m\}$. The grammatical inference algorithm is made of three parts. The aim of the first part is to choose a set of substructures of the fingerprint patterns which have high occurrence. Since each window basically contains a set of ridges, it seems natural to take individual ridge configurations as substructures. Therefore a simple ridge, a ridge branching into two ridges, two ridges merging into one, or any other ridge configurations are considered as substructures of the fingerprint patterns. In the second part, the grammatical inference infers the production rules to recognize both the new ridge configuration such as a ridge with double branching and the production rules recognizing the collection of these substructures, such as production rules recognizing the occurrence of a ridge with double branching and a simple merging ridge along with a set of continuous simple ridges. Since regular tree grammars are closed under union operation, in the third part, the inferred grammar G_t for the window becomes the union of G_i's the inferred subgrammars of the individual ridge configuration. Under the union operation all redundant rewriting rules are eliminated and also the tree automaton recognizing the regular tree grammar G_t is derived from the production rules. Since the production rules of the subgrammars G_i's corresponding to various ridge configurations are the same in many grammars, then by forming the union of the G_i's a significant reduction in the number of production rules can be made. In fact, sometimes, the difference between two windows is due to the occurrence of one extra ridge which means the grammars of the two classes are exactly the same except for one or two production rules. Also, because no ordering criterion of the ridges within a window is observed for classification of the patterns, any newly inferred grammar is tested to see whether the new grammar is similar to the previous grammars with the same set of ridges but different order of the ridges in this case the needed production rules are added or eliminated in order to avoid any ambiguity. An inferred grammar, for instance, recognizing the class of windows with two branching ridges and a merging ridge can

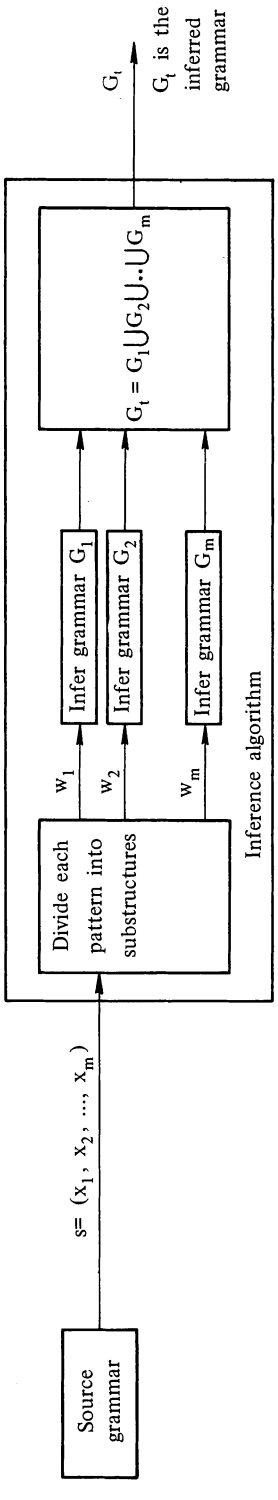


Fig. 8.17. A model of grammatical inference system

also be inferred under the union operation of the grammars generating windows with a branching ridge along with a merging ridge and generating windows with only one branching ridge. A reachability matrix M whose elements are

$$m_{ij} = \begin{cases} 1 & \text{if } T_i \overset{*}{\Rightarrow} T_j \\ 0 & \text{otherwise} \end{cases}$$

where T_i and T_j are nonterminals associated with classes i and j is used to make the above adjustment. To avoid any ambiguity after the inference of any new grammar G_k the following two tests are made.

1) If m_k is identical to any of the rows of the matrix M, say m_i, in which case the two grammars G_k and G_i are describing a different order of the similar ridges within a window. Hence the needed production rules are added to the grammar G_i in order to describe all combinations of the set of ridges.

2) If m_k is identical to the union of any two rows of the matrix M, say m_i and m_j, in which case the needed production rules will be added to the grammar G_k in order to complete the new grammar.

8.3.4 Computer Simulation

The programs for simulation of the 92 fingerprint samples were written in Fortran language on the CDC-6500 computer at Purdue University. The grammars of the first 10 classes were generated manually while the rest of the grammars were inferred through the simulation by the grammatical inference algorithm. Although the 92 fingerprint patterns may generate as many as 1472 classes for each window, many of the window configurations are similar to each other and a small group of tree languages may be sufficient to describe all the fingerprint impressions. The inference procedure was designed to generate the maximum number of classes, so a tree grammar was inferred for each new combination of the ridges although no member of the class may have yet been encountered. For instance, when the tree grammar of class C-23 presented in Table 8.5, which describes the class of windows with two branching ridges, one sudden ending, and a set of simple continuous ridges, was inferred, the grammar of class C-22 shown in Table 8.6 was also inferred. The grammar of class C-22, which recognizes window patterns having two branching ridges along with a set of simple continuous ridges, has recognized 23 windows by the time all the 92 fingerprint impressions were simulated. In part three of the inference algorithm the production rules

were added to the grammar of class C-23 when it was observed that the union of the rows m_{21} and m_3 of the matrix M was identical to row m_{23}. It is clear that the addition of the two production rules caused no differentiation

Table 8.5. Tree grammar class 23

$G_{t23} = (V, r, \mathrm{P}, \mathrm{S})$

where

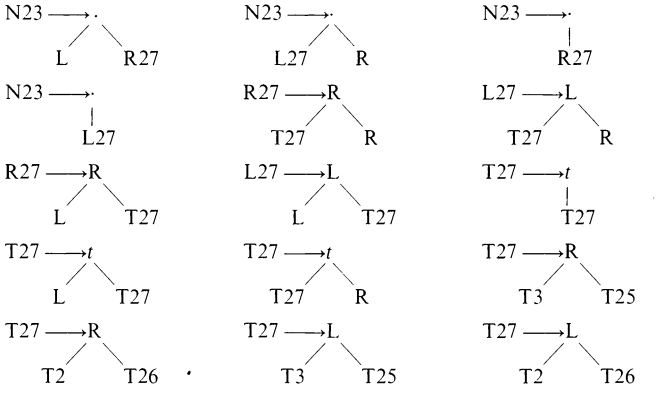
$\mathrm{S} = \text{starting symbol} = \{\mathrm{N23}\}$

$V = \{\mathrm{N23, T1, T3, T27, T2, T25, T26}, t, \mathrm{R, L, R27, L27}\}$

$r(.) = r(\mathrm{L}) = r(\mathrm{R}) = r(t) = \{1, 2\}$

$t = \{\mathrm{A, B, C, D}\}$

and P:



Production rules of T2, T3, T25, T26 can be found in [8.19].

Table 8.6. Tree grammar class 22

$G_{t22} = (V, r, \mathrm{P}, \mathrm{S})$

where

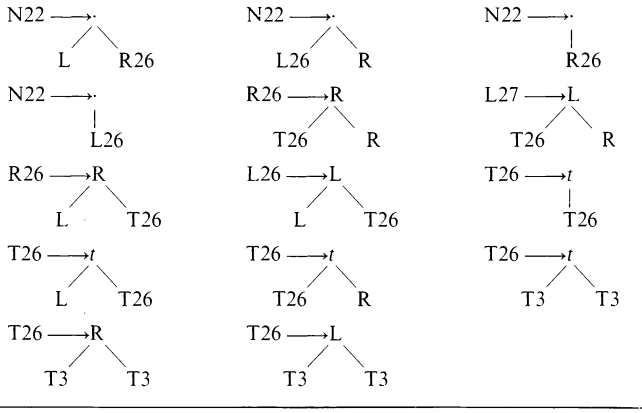
$\mathrm{S} = \text{Starting symbol} = \{\mathrm{N22}\}$

$V = \{\mathrm{N22, T1, T3, T26, R26, L26}, t, \mathrm{R, L}\}$

$r(.) = r(\mathrm{L}) = r(\mathrm{R}) = r(t) = \{1, 2\}$

$t = \{\mathrm{A, B, C, D}\}$

and P:



Production rules of T3 can be found in [8.19].

to be made in the order of the appearance of the ridges within the window, i.e., windows with branching-branching-ending ridge sequence, branching-ending-branching ridge sequence, and ending-branching-branching ridge sequence are classified into class C-22. However, in a practical application such differentiation may be made in order to further subdivide the classes if desired. The results of the first-level classifier and the tree grammar technique are presented in Table 8.7. In this study, for no specific reason, the patterns were not simulated in the order of their identification number. The simulation results indicate that 193 unbalanced classes were generated. For instance, classes C-1 and C-2 have 93 and 193 members, respectively, but some others have none or one member only. Class C-1, which represents a set of parallel and continuous ridges running through the windows, has no information while class C-2, which represents windows with sudden ending ridges which is very important for classification, is overpopulated. The advantage of a balance system is for minimizing the search time. However, no attempt was made to generate a balance system at this time because it was felt that the set of 92 fingerprint patterns was too small to adequately represent the true distribution of the fingerprint impressions. It is obvious that a balance system may be developed for example by dividing class C-2 into a set of classes describing windows with 1, 2, 3, or more ridges with sudden endings. Also, intermediate classes such as C-30 and C-32 with no member may be eliminated and classes with one member such as C-29 may be combined with similar classes. To limit the number of classes in this study the differentiation between the left and right position of the features and their type of curvature was ignored, while these primitives may be used to generate a more balanced system. The study showed that the inference rate of the new classes was high at the beginning of the study and it decreased as more patterns were considered. As a result, it may be expected that this rate approaches zero as more and more fingerprint impressions are analyzed. About 4.2% of the windows were classified as misconnected which indicated that the postprocessing algorithm made improper connection of the ridges. The erroneous results have mainly been found in the windows with the ridges having a high degree of curvature. It seems that this type of misconnection may be eliminated by developing different algorithms for different windows of possibly by considering different sizes of windows. Also, about 11.9% of the windows were found too noisy for feature extraction. Windows which were too dark or too light due to excessive or insufficient use of ink when the fingerprints were taken were classified as too noisy. Some of the fingerprint impressions with scars, which interrupted the normal flow of the ridges, were also found too noisy for classification. If all 16 windows of the sampling matrix have the same probability to be classified into any of the 193 classes, then a total of 193^{16} or approximately 2×10^{34} classes of fingerprints may be generated which is much greater than all the fingerprints already collected. However, the fact is that many of these classes have zero probability of occurrence such as a fingerprint whose 16 sampling windows were classified into C-20 which describes windows consisting of two loops and a merging ridge.

The overall results are very encouraging and the complete documentation of the programs along with the tree grammars of the 193 classes is presented in

Table 8.7. Summary of the tree system

Tree grammar

Pattern number	1st-level classifier		Window number 2nd-level classifier															
	True	Machine	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	I	I	C-2	C-23	C-7	C-1	C-6	*	C-9	C-55	C-63	†	†	C-2	C-11	C-43	†	C-2
2	I	I	C-14	C-13	C-2	C-2	C-107	C-60	C-1	C-12	C-12	C-16	*	C-11	C-3	C-21	C-55	C-1
3	V	V	C-2	C-6	C-1	C-12	C-3	C-20	C-24	C-12	C-1	C-16	C-177	C-1	C-6	C-3	C-1	C-1
4	IV	IV	C-21	*	*	C-2	C-3	C-13	C-12	C-1	C-24	*	C-2	C-2	C-187	C-65	†	†
5	V	V	C-54	†	†	†	C-22	†	C-94	C-28	C-1	C-160	C-6	C-2	C-28	C-2	C-1	C-1
6	VII	VII	C-110	C-21	C-27	C-2	C-2	C-49	C-2	C-1	C-10	C-107	C-1	C-167	C-2	C-2	C-12	C-1
7	III	III	C-2	C-2	C-2	C-44	C-2	C-11	C-17	C-13	C-12	C-6	C-6	C-1	C-27	C-2	C-24	C-1
8	I	I	C-28	C-28	C-2	C-2	C-44	C-41	C-43	C-65	C-2	C-139	C-2	C-10	C-10	C-2	C-101	C-3
9	VI	VI	C-44	C-27	C-44	C-58	C-5	C-109	C-48	C-2	C-167	C-5	C-13	C-12	C-2	C-69	C-2	C-27
10	II	II	C-67	C-28	C-2	C-2	C-68	C-17	C-43	C-2	C-1	C-6	C-6	C-2	C-1	C-22	†	C-6
11	III	III	C-2	C-21	C-1	C-25	C-23	C-2	C-10	†	C-25	C-2	C-13	C-26	C-24	C-29	C-12	C-2
12	I	I	C-37	*	C-2	C-2	C-12	C-25	C-2	C-6	C-3	C-107	C-2	C-2	C-34	C-35	C-12	C-2
13	IV	IV	C-25	C-1	C-11	C-1	C-42	*	*	C-23	C-40	C-13	C-1	C-2	C-13	C-1	C-38	†
14	II	II	C-2	C-2	C-2	C-27	C-27	C-2	C-10	C-47	C-51	C-2	C-12	C-50	C-24	C-5	C-12	C-12
15	IV	IV	C-22	C-3	C-3	C-54	C-21	*	*	C-53	C-13	C-65	C-57	C-10	C-21	C-25	C-55	C-12
16	IV	IV	C-23	†	C-9	C-1	C-38	†	C-123	C-61	C-12	†	†	C-9	C-57	C-1	†	†
17	VII	VII	C-25	C-5	C-1	C-5	C-4	C-25	C-17	C-64	C-10	C-27	C-11	C-2	C-36	C-9	C-84	C-2
18	V	V	C-9	C-6	C-7	C-68	C-2	C-70	C-6	C-83	C-21	C-52	C-57	†	C-22	C-72	†	C-3
19	IV	IV	C-1	C-68	C-21	C-3	C-46	*	C-22	C-44	C-69	C-24	C-9	C-9	C-4	C-36	C-9	†
20	I	R	C-6	C-1	C-6	C-2	C-9	C-86	†	C-1	C-22	C-22	C-101	†	C-1	C-13	C-13	C-24
21	V	V	C-3	C-79	C-2	C-5	C-13	C-73	C-81	C-25	C-12	C-28	C-78	C-6	C-12	C-12	C-2	†
22	III	III	C-23	*	C-26	C-6	C-87	*	C-89	*	C-86	C-88	C-2	C-2	C-24	C-65	C-21	C-65
23	III	III	C-92	C-23	C-2	C-2	C-12	C-91	C-66	†	C-53	*	C-6	C-6	C-12	C-7	C-21	C-1
24	II	R	*	†	†	†	*	C-71	†	†	C-12	C-21	C-12	†	†	C-21	†	†
25	VI	VI	C-38	C-2	C-6	C-119	C-2	C-12	C-122	C-3	C-10	C-2	C-2	C-123	C-56	C-1	C-1	C-24
26	I	I	*	C-97	C-2	C-3	C-21	C-24	C-71	C-10	C-54	C-74	*	C-11	C-120	C-6	C-12	C-2
27	I	R	*	†	†	C-6	†	†	†	†	†	C-71	C-89	†	C-21	C-42	*	†

28	I	I	C-23	C-12	C-6	C-2	C-12	C-2	C-12	C-21	C-106	C-177	C-66	C-10	C-35	*	C-24	C-12
29	II	II	C-2	C-2	C-2	C-1	C-124	C-21	C-3	C-1	C-2	C-100	C-126	C-5	C-2	C-2	C-2	C-17
30	I	I	C-12	C-2	C-6	C-3	C-1	*	C-118	C-9	C-22	C-12	C-24	C-3	C-6	C-25	C-3	C-6
31	III	III	C-2	C-25	C-12	C-1	C-1	C-2	C-12	C-3	*	C-24	C-123	C-101	C-3	C-100	C-2	C-6
32	VII	VII	C-2	C-1	C-94	C-2	C-5	C-1	C-25	C-12	C-99	C-104	C-2	C-102	C-2	C-13	C-2	C-9
33	III	III	C-21	C-2	C-23	C-12	C-77	C-2	C-4	C-25	C-23	C-105	C-123	C-12	C-21	C-54	C-2	C-21
34	I	I	C-78	C-82	C-82	C-2	C-13	C-12	†	C-105	C-2	C-105	C-25	C-105	C-13	C-23	C-101	C-12
35	I	I	C-3	C-92	C-2	C-2	C-3	C-110	C-111	C-41	C-3	C-9	C-107	C-24	C-52	C-108	C-2	C-9
36	VI	VI	C-8	C-35	C-12	C-2	C-51	C-3	C-15	*	C-21	C-10	C-112	C-44	C-113	C-109	C-1	C-1
37	I	II	C-57	C-12	C-12	C-6	C-66	*	C-31	C-106	C-1	C-31	C-27	C-12	C-13	C-27	C-27	C-12
38	IV	IV	C-88	C-1	C-38	*	C-88	C-24	C-3	C-1	C-2	C-114	*	C-23	C-51	C-1	C-9	C-3
39	II	II	C-27	C-28	C-12	C-1	C-13	C-112	C-13	C-13	C-10	C-10	C-129	C-61	C-107	C-158	C-11	C-12
40	I	I	C-115	C-35	C-12	C-2	C-9	C-14	C-2	C-2	C-21	C-25	C-2	C-3	C-37	*	C-2	C-2
41	I	I	C-101	C-24	C-6	C-22	C-1	C-71	C-93	C-21	C-1	C-71	C-22	C-2	C-127	C-1	C-2	C-11
42	II	II	C-73	C-69	C-1	C-1	C-21	C-130	C-21	C-1	C-3	*	C-132	C-2	C-24	C-73	*	C-11
43	II	II	C-21	C-3	C-3	C-1	C-2	C-47	C-6	C-24	C-93	C-3	C-2	†	C-12	C-122	C-1	C-3
44	III	III	C-10	C-7	C-2	C-35	C-23	C-12	C-2	C-134	C-167	C-2	*	C-24	C-133	C-2	C-21	C-2
45	III	III	C-77	C-3	C-28	C-1	C-23	C-8	C-136	C-2	C-66	C-2	C-6	C-1	C-3	C-50	C-1	C-22
46	I	I	C-142	C-13	C-12	C-2	C-60	C-137	C-139	C-7	C-3	C-24	C-71	C-12	C-27	C-66	C-2	C-3
47	VI	VI	C-1	C-76	C-12	C-26	C-2	C-98	C-143	C-1	C-2	C-27	C-3	C-102	C-73	C-2	C-20	C-71
48	II	II	C-142	C-2	C-1	C-2	C-13	C-21	C-6	C-2	C-12	C-21	C-141	C-2	C-10	C-68	C-140	C-6
49	II	R	C-2	C-2	C-108	*	C-2	C-2	C-3	C-5	C-6	C-2	†	†	C-27	*	*	C-1
50	VII	VII	C-12	C-13	C-2	C-63	C-10	C-106	C-24	C-11	C-8	C-6	C-144	†	C-54	C-146	*	C-9
51	IV	IV	C-21	C-2	C-6	C-56	C-17	C-143	C-61	C-66	C-54	C-1	C-1	C-1	C-27	C-1	C-2	†
52	I	I	C-69	C-12	C-12	C-11	C-93	C-12	C-62	C-149	C-71	C-9	C-148	C-12	C-24	*	C-89	C-6
53	I	I	C-11	C-150	C-107	C-1	C-1	C-153	C-39	C-102	C-2	*	C-151	C-3	C-107	*	C-63	†
54	VII	R	C-12	†	†	†	C-12	†	†	C-2	C-12	†	C-12	†	†	†	†	†
55	I	R	C-24	C-46	†	†	C-52	†	†	†	C-63	C-12	†	†	C-6	†	†	C-2
56	R	R	†	†	†	†	†	†	†	†	†	†	†	†	†	†	†	†
57	VI	VI	C-2	*	C-13	†	C-2	C-2	C-121	C-101	C-35	C-26	C-24	*	C-146	C-12	C-6	†
58	II	II	C-12	C-124	C-12	C-62	C-17	C-107	C-23	C-154	C-75	C-24	C-155	C-23	C-107	C-71	C-157	C-2
59	VI	R	C-1	†	C-2	C-1	C-25	C-12	C-6	†	C-2	*	*	C-21	C-25	†	C-35	C-12
60	III	I	C-12	C-155	C-10	C-10	C-124	C-21	C-159	C-10	C-2	C-2	C-141	C-126	C-158	C-162	C-13	C-13
61	VII	R	†	†	†	†	†	†	†	†	†	†	†	†	C-54	†	C-11	†
62	II	II	C-163	C-23	C-112	C-14	C-12	C-164	C-12	C-24	C-89	C-155	†	C-24	C-13	C-56	*	C-9
63	VI	VI	C-102	C-13	C-13	C-2	C-54	C-2	*	C-2	C-22	C-35	C-21	C-6	C-78	†	†	C-21

Table 8.7 (continued)

Tree grammar																		
Pattern number	1st-level classifier		Window number 2nd-level classifier															
	True	Machine	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
64	I	I	*	C-66	C-1	C-1	C-167	C-166	C-147	C-5	C-21	C-159	C-170	C-93	C-66	C-2	C-6	C-1
65	VII	VII	†	†	†	C-1	C-24	C-24	†	C-171	C-172	C-21	C-4	C-21	C-173	†	C-26	C-12
66	IV	IV	C-109	C-69	C-175	C-174	C-21	C-23	C-175	C-145	C-9	C-38	C-153	C-12	C-6	C-6	C-56	C-3
67	IV	IV	†	†	C-1	†	C-2	†	C-61	†	C-2	†	C-2	C-1	†	†	†	C-1
68	II	II	C-12	C-45	C-12	C-11	C-145	C-62	C-123	C-101	C-12	C-2	C-12	C-3	C-2	C-2	C-21	C-2
69	III	III	C-176	C-21	C-2	C-6	C-21	C-156	C-1	C-10	C-51	C-24	C-13	C-2	C-142	C-13	C-6	C-1
70	III	III	C-165	C-54	C-1	C-21	C-1	C-123	C-3	C-6	*	C-24	C-9	C-21	C-12	C-3	C-1	C-2
71	VI	VI	C-24	*	C-177	C-2	C-179	*	C-22	C-6	C-66	†	C-2	C-2	*	C-2	†	C-109
72	I	I	C-57	C-45	C-2	C-1	C-6	C-6	C-107	C-22	C-12	C-130	*	C-41	C-9	C-141	C-2	C-2
73	III	R	C-21	C-69	C-1	†	C-23	C-70	C-21	†	C-176	C-146	C-24	†	C-20	C-97	†	†
74	I	I	C-124	C-96	C-177	C-12	C-10	C-18	C-123	†	C-150	C-71	C-177	†	C-182	C-6	C-6	C-102
75	IV	IV	C-23	C-2	†	C-2	C-12	C-176	C-2	†	C-2	*	C-2	C-12	C-71	C-21	†	C-12
76	II	R	*	C-63	C-6	C-24	C-22	C-107	C-24	C-1	C-38	C-44	C-12	C-21	C-12	C-2	C-2	C-5
77	VI	VI	C-112	†	C-13	C-6	C-3	†	C-143	C-2	C-183	†	C-6	C-1	C-79	C-184	C-107	C-9
78	VII	VII	C-12	†	C-38	C-102	C-78	†	†	†	C-21	C-2	C-2	C-109	C-8	C-12	C-1	C-1
79	I	I	C-22	C-24	C-94	C-6	C-2	C-140	†	†	C-3	C-109	C-13	C-2	C-22	C-28	C-2	C-2
80	VII	R	†	†	C-27	C-63	C-12	C-44	C-35	†	C-2	†	†	C-2	C-21	C-2	C-101	C-13
81	III	III	C-2	C-122	C-2	C-2	C-145	C-10	C-186	C-3	*	C-12	C-13	*	C-185	C-12	C-1	C-13
82	II	II	C-22	C-44	C-12	C-174	C-2	C-28	C-143	C-61	C-2	C-13	C-10	C-22	C-13	C-182	C-21	C-1
83	VI	VI	C-191	C-141	C-13	C-24	C-190	C-3	C-189	C-1	C-88	C-24	C-187	C-6	C-145	C-21	C-12	C-21
84	VI	R	†	†	†	†	†	†	†	†	†	†	†	†	†	†	†	†
85	VI	R	C-22	C-35	C-2	†	C-58	*	C-6	C-2	C-77	*	C-21	C-9	*	*	C-65	C-3
86	III	III	C-187	C-3	C-1	C-2	C-21	C-2	C-189	C-187	C-27	C-2	C-24	C-6	C-22	C-185	C-6	C-6
87	III	I	C-23	C-23	C-73	C-6	C-69	C-21	C-26	C-95	C-70	C-190	C-12	C-13	C-112	C-192	C-24	C-107
88	VII	VII	C-2	C-24	C-177	C-21	C-90	C-24	C-123	†	C-10	C-74	C-6	†	C-13	C-13	C-24	C-12
89	I	I	C-69	C-2	C-6	C-159	C-176	C-96	C-151	C-24	C-12	C-12	C-81	C-123	C-77	C-167	C-18	C-6
90	VII	R	C-26	C-12	C-12	†	C-9	C-12	C-12	†	C-6	C-3	†	†	†	C-71	†	†
91	IV	IV	C-23	C-3	C-2	C-107	C-54	C-23	C-2	C-6	C-3	C-61	*	C-177	C-24	C-6	C-9	*
92	I	I	C-56	C-12	C-93	C-9	C-22	*	C-18	C-3	C-21	C-2	C-139	*	C-22	C-135	C-12	C-6

† misconnected window
* noisy window

[8.19]. The computer time for preprocessing, postprocessing, encoding, and parsing without binary extraction of the pattern is about 50 s per fingerprint.

8.4 Summary

In this study, an approach using string and tree systems for syntactic pattern recognition of fingerprints is investigated. A set of context-free languages was used as the first-level classifier, and, in order to increase the number of classes, a set of stochastic context-free languages was developed to subclassify the output of the first-level classifier. Since the volume of fingerprints collected runs into millions, a set of tree languages was designed to generate as many as classes as desired for minimizing the search time.

In conclusion, it is easy to see that the two approaches, namely string and tree systems, are complementary to each other in an automatic fingerprint identification system. However, it is clear that for a practical application more investigation is still needed.

References

8.1 C. E. CHAPEL: Fingerprinting—A Manual of Identification (Coward McCann, Inc. 1941)
8.2 C. E. CHAPEL: The Science of Fingerprints (U.S. Government Printing Office 1963)
8.3 M. ELECCION: IEEE Spectrum **XX**, XXXX (1973)
8.4 M. TRAURING: Nature **XX**, 938–940 (1963)
8.5 C. R. KINGSTON: Problems in Semi-Automated Fingerprint Classification; in Law Enforcement Science and Technology (Academic Press 1967)
8.6 J. H. WEGSTEIN: NBS Tech. Note 538 (U.S. Government Printing Office 1970)
8.7 J. H. WEGSTEIN, J. F. RAFFERTY: Machine Oriented Fingerprint Classification; in Law Enforcement Science and Technology (Academic Press 1967)
8.8 J. H. WEGSTEIN, J. F. RAFFERTY: NBX Tech. Note 466 (U.S. Government Printing Office 1968)
8.9 C. B. SHELMAN: Machine Classification of Fingerprints; in Law Enforcement Science and Technology (Academic Press 1967)
8.10 C. B. SHELMAN: Development and Evaluation of a Reader of Fingerprint Minutiae, Rept. XM-2478-X1 (Cornell Aeronautical Lab, Buffalo, NY 1969)
8.11 C. B. SHELMAN: Evaluation of an Improved Reader of Fingerprint Minutiae, Rept. XM-2478-X1 (Cornell Aeronautical Lab, Buffalo, NY 1969)
8.12 C. B. SHELMAN: Fingerprint Reader Improvements and Registration Techniques, Rept. XM-2478-X-3 (Cornell Aeronautical Lab, Buffalo, NY 1970)
8.13 E. MAROM: Fingerprint Classification and Identification Using Optical Methods; in Law Enforcement Science and Technology (Academic Press 1967)
8.14 V. V. HORVATH, J. M. HOLEMAN, C. Q. LEMOND: in Law Enforcement Science and Technology (Academic Press 1967), pp. 485–492
8.15 A. GRASSELLI: in Methodologies of Pattern Recognition, ed. by S. WATANABE (Academic Press 1969), pp. 253–273
8.16 W. J. HANKLEY, J. T. TOU: in Pictorial Pattern Recognition, ed. by G. C. CHENG, R. S. LEDLEY, D. K. POLLOCK, A. ROSENFELD (Thompson Book Co., Washington, D.C. 1968), pp. 411–456
8.17 J. T. TOU: On Feature Encoding in Picture Processing by Computer, Proc. 7th Ann. Allerton Conf. Circuits System Theory, University of Illinois (1969)
8.18 G. LEVI, F. SIROVICH: Information Science **4**, 327–356 (1972)
8.19 B. MOAYER, K. S. FU: Tech. Rept. TR-EE 74–36, School of Electrical Engr., Purdue University, Lafayette, IN (1974)
8.20 B. MOAYER, K. S. FU: Pattern Recognition **7**, 1–24 (1975)

8.21 G. S. SIDHU, R. T. BOUTE: IEEE Trans. C-**21**, 1206–1215 (1972)
8.22 B. CHARTRES, J. FLORENTIN: J. Automatic Computing Machinery **15**, 447–464 (1968)
8.23 J. EARLY: Comm. Automatic Computing Machinery **13**, 94–102 (1970)
8.24 K. S. FU: Sequential Methods in Pattern Recognition and Machine Learning (Academic Press 1968)
8.25 K. S. FU: Syntactic Methods in Pattern Recognition (Academic Press 1974)
8.26 P. H. SWAIN, K. S. FU: Tech. Rept. TR-EE 70–20, School of Electrical Engr., Purdue University, Lafayette, IN (1970)
8.27 H. C. LEE, K. S. FU: A Syntactic Pattern Recognition System with Learning Capabilities; in Proc. COINS-72, ed. by J. T. TOU (Plenum Press, New York 1972)
8.28 T. L. BOOTH: Probabilistic Representation of Formal Languages; in IEEE Conf. Record of the 1969 10th Ann. Symp. Switching Automata (1969)
8.29 B. MOAYER, K. S. FU: A Syntactic Approach to Fingerprint Pattern Recognition; in Proc. 1st Intern. Joint Conf. Pattern Recognition, Washington, D.C. (1973)
8.30 R. A. KIRSCH: IRE Trans. EC-**13**, 363–376 (1964)
8.31 N. F. DACEY: Pattern Recognition **2**, 11–31 (1970)
8.32 J. L. PFALTZ, A. ROSENFELD: Web Grammars; in Proc. Joint Intern. Conf. Artificial Intelligence, Washington, D.C. (1969)
8.33 J. FEDER: Information Sciences **3**, 225–241 (1971)
8.34 I. PAVLIDIS: J. Automatic Computing Machinery **19**, 11–22 (1972)
8.35 K. S. FU, B. K. BHARGAVA: IEEE Trans. C-**22**, 1087–1099 (1973)
8.36 L. S. G. KOVASZNAY, H. M. JOSEPH: Proc. IRE **43**, 560–570 (1955)
8.37 K. S. FU, T. L. BOOTH: IEEE Trans. SMC-**5**, 95,409 (1975)
8.38 B. K. BHARGAVA, K. S. FU: Transformations and Inference of Tree Grammars for Syntactic Pattern Recognition; IEEE Intern. Conf. System, Man Cybernetics, Dallas, TX (1974)

9. Modeling of Earth Resources Satellite Data

J. M. BRAYER, P. H. SWAIN, and K. S. FU

With 25 Figures

In this chapter, we demonstrate how web grammars may be used to develop syntactic models for a complex class of images. The images considered are pictures of urban areas obtained by the Earth Resources Technology Satellite (LANDSAT). We shall address specifically two basic questions: 1) Can a human designer familiar with web grammar theory construct web models of real images? And 2), are the resulting models useful? Affirmative answers to these questions will demonstrate the utility of the grammatical modeling approach in a real and nontrivial picture processing application. Specifically, as we indicate in the following section, such an approach holds promise for improving our ability to make practical use of multispectral data collected by satellites—for inventorying our natural resources, for planning the orderly growth of our urban areas, and for otherwise monitoring the condition and utilization of our environment.

9.1 The Satellite Data

The data used for this analysis is actually the result of work by TODD and BAUMGARDNER [9.1] at the Laboratory for Applications of Remote Sensing, Purdue University. In their study, a portion of a frame of multispectral scanner data collected by the LANDSAT-1 satellite over Marion County (Indianapolis), Indiana, was processed by multivariate statistical classification methods for the purposes of mapping land use and developing a system for monitoring land use.

The raw satellite data consisted of an array of picture elements (pixels), each pixel representing a ground area of approximately 60×70 m. The multispectral scanner aboard the LANDSAT measures reflected energy from each pixel in four wavelength bands, two in the visible and two in the infrared portions of the spectrum. Thus each pixel in the data is actually comprised of a four-dimensional vector of reflected energy values.

The Marion County data was initially viewed on a digital imaging display for purposes of orientation. Next, several small areas were defined for a clustering algorithm, which was instructed to find fourteen spectral clusters in the data. A spectral cluster is a group of points not necessarily spatially close but having similar reflectance characteristics and thus presumably representing similar land use. Each cluster was approximated by a multivariate Gaussian (normal) distribution and a Bayes classifier was constructed. All of the pixels were then classified and subsequently some classes judged to represent very similar land use were combined. A pseudo-graylevel picture of the resulting nine classes (one graylevel per class) is shown in Fig. 9.1 a. Several classes are shown individually in Figs.

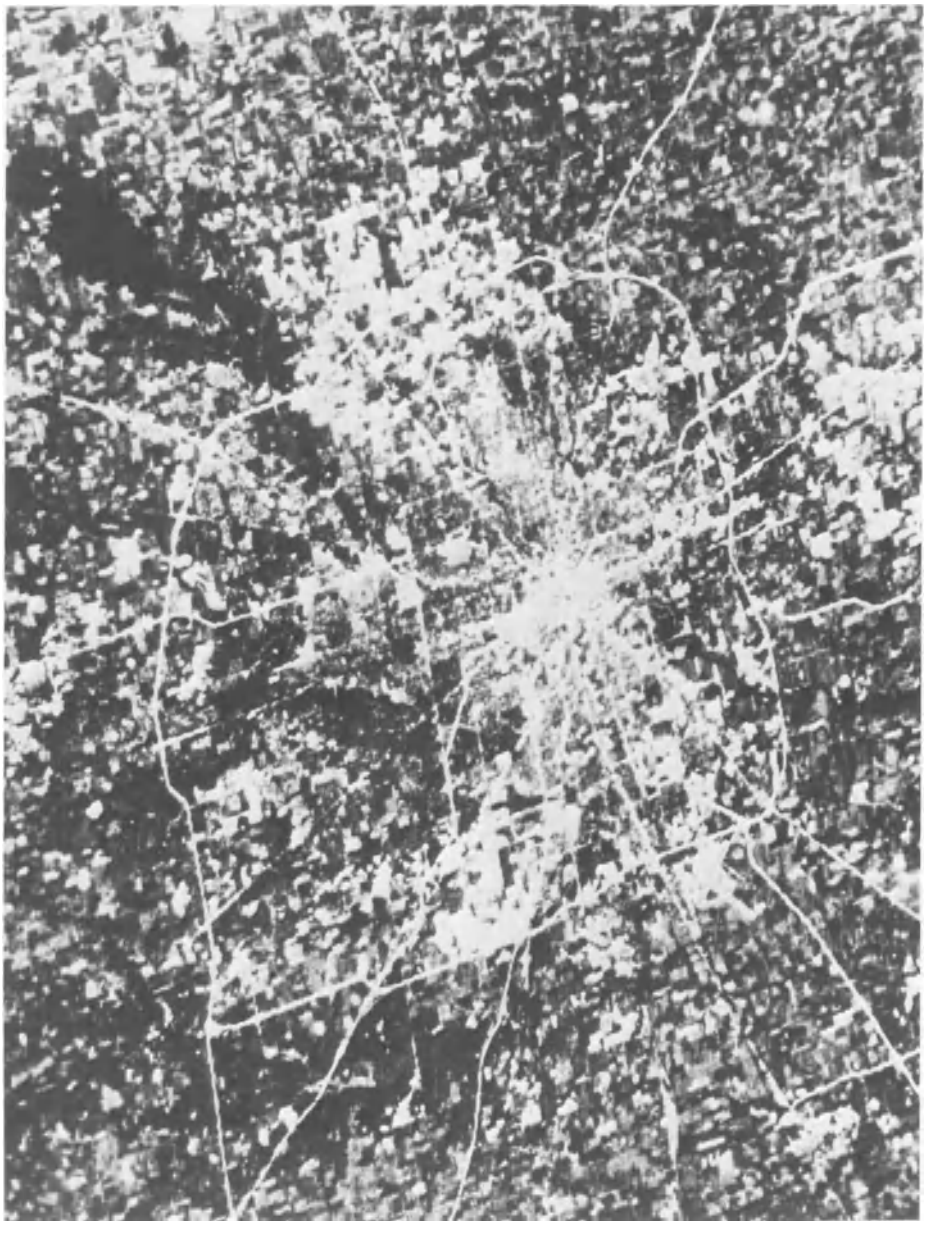


Fig. 9.1a–h. Results of pointwise Bayesian classification a. 9 classes

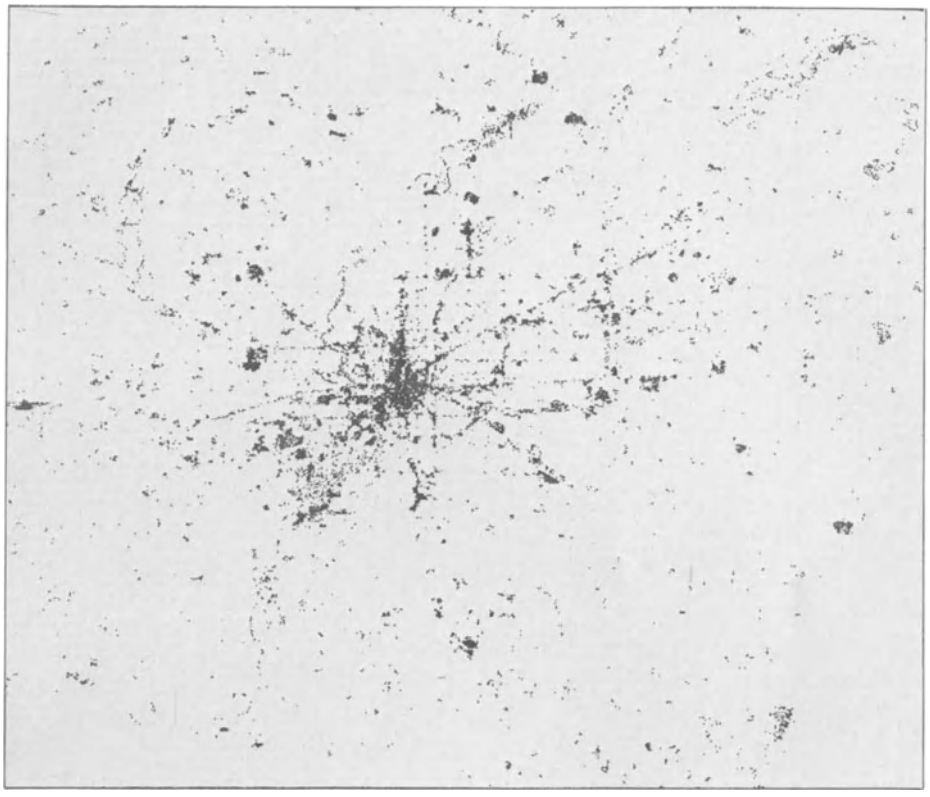


Fig. 9.1 b. "Commercial"

9.1 b to 9.1 h. Figure 9.2 shows both the cloud and the cloud shadow classes. A line printer plot of a small section of the composite classified data is given in Fig. 9.3 with different symbols representing different land use classes. A key to the symbol used for each class is given in Table 9.1.

Accurate discrimination of the land use classes defined in the TODD and BAUMGARDNER study depends on three key assumptions: 1) that the land use classes have unique spectral signatures, 2) that the set of land use classes defined is exhaustive for the scene, and 3) that each pixel contains a single "pure" land use class. None of these assumptions is perfectly satisfied in practice, however, and so there is room for improvement in the classification results. We shall show that the spatial information (such as shape, orientation, context) in the data can be used to improve on the classification results and that web grammars provide a convenient formalism for modeling the spatial information.

This data was selected for several reasons. First of all, this set of data comprises a complex, naturally occurring scene. It can be regarded as a typical picture in which each picture element contains on the order of three or four bits of information. There is some justification for arguing that if web grammars are

Fig. 9.1 c. "Inner City"

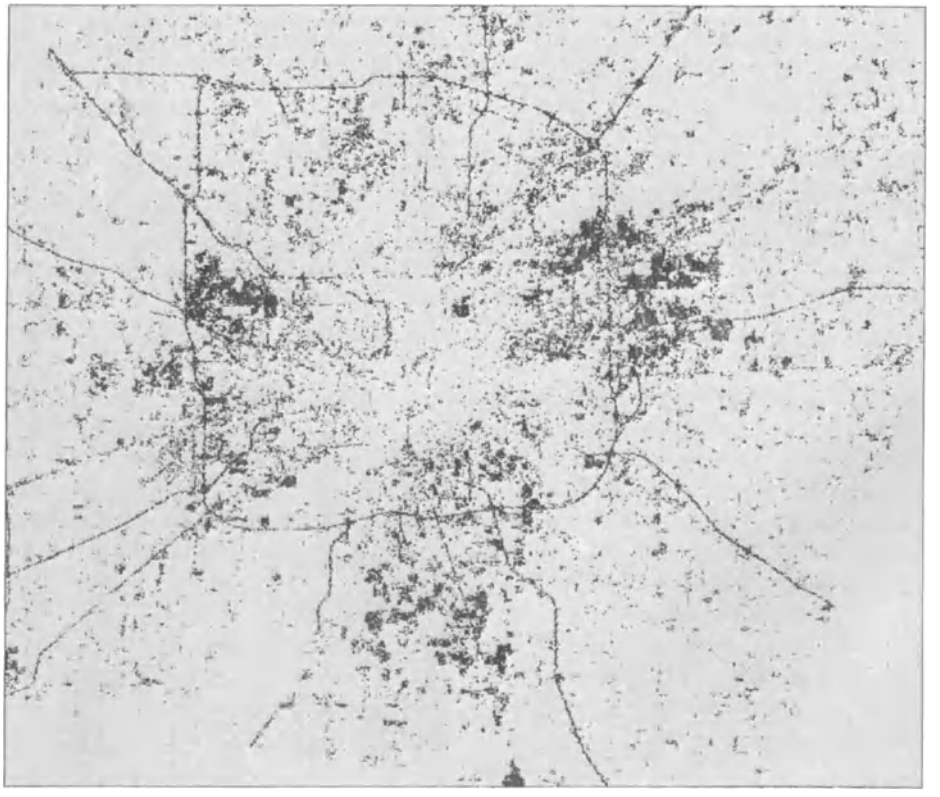


Fig. 9.1 d. "Suburban"

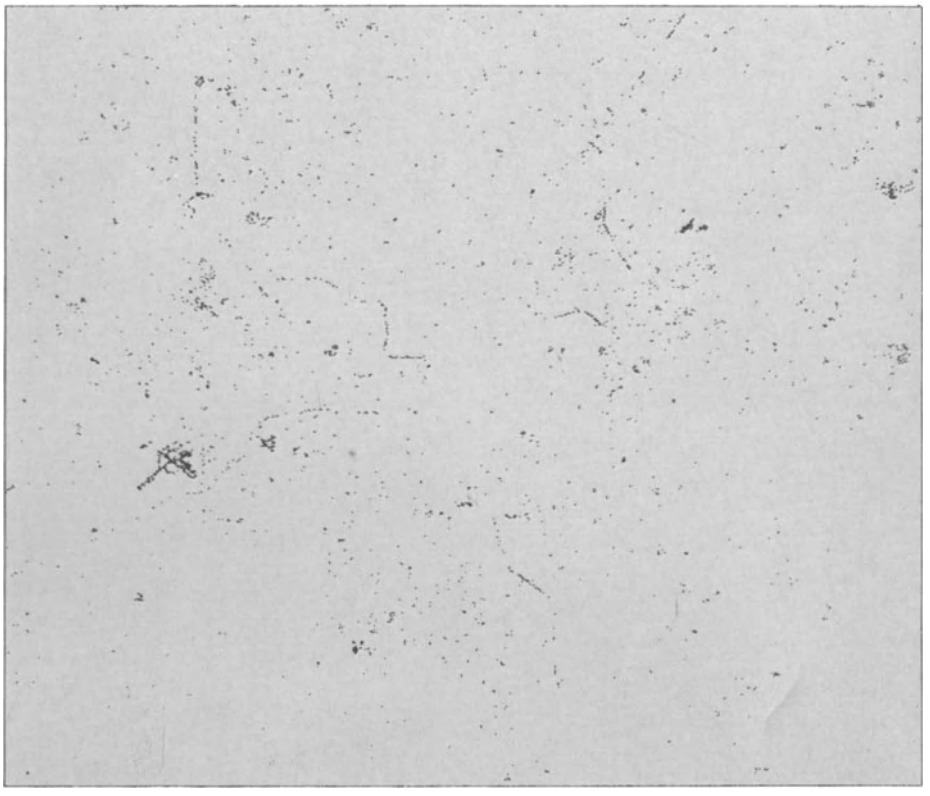


Fig. 9.1 e. "Concrete"

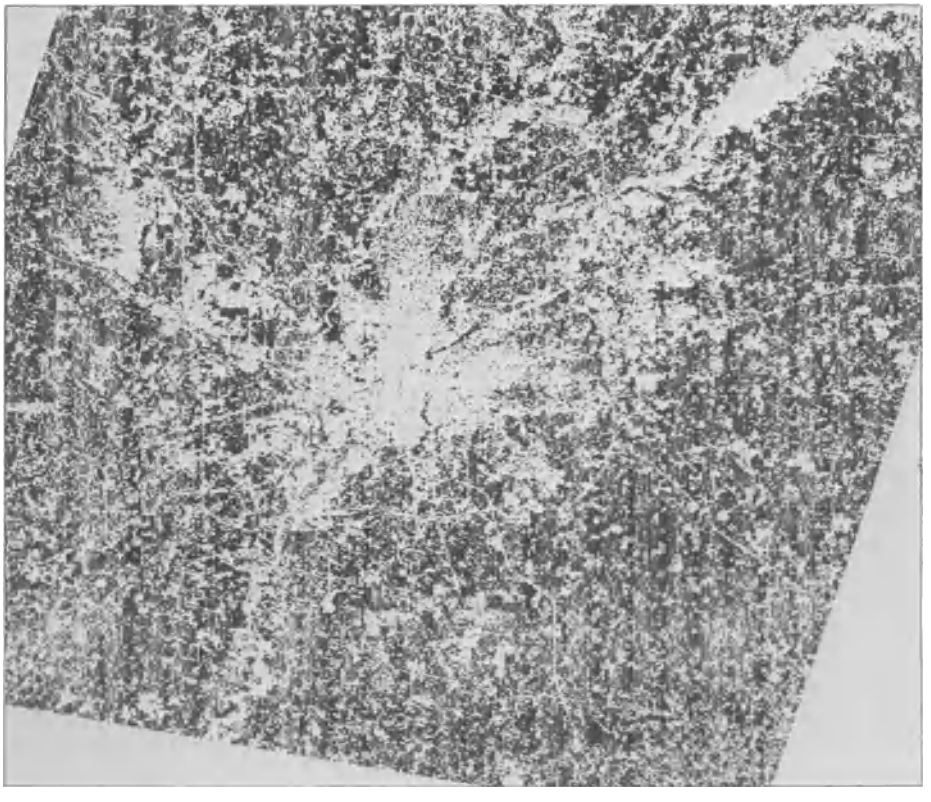


Fig. 9.1 f. "Open Grassy"

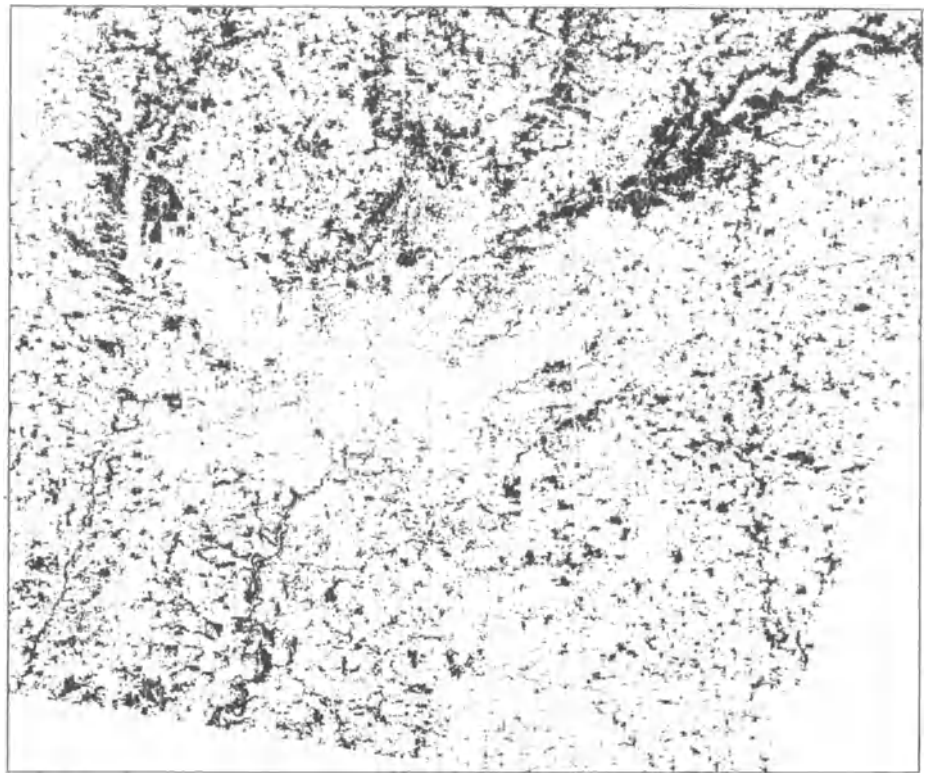


Fig. 9.1 g. "Wooded"

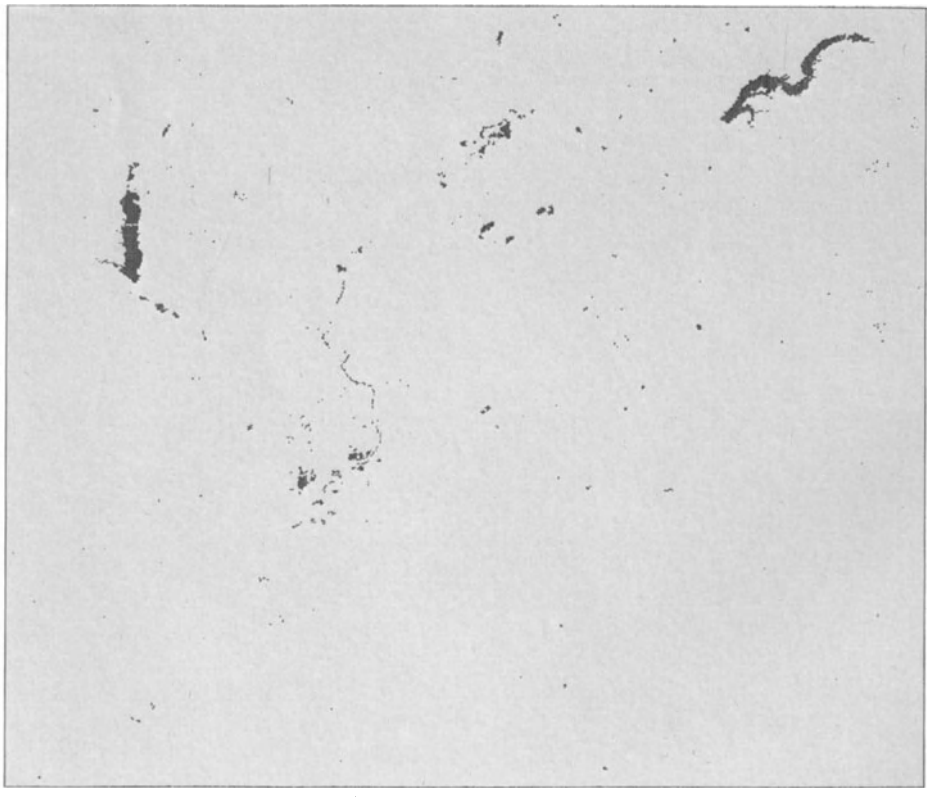


Fig. 9.1 h. "Water"

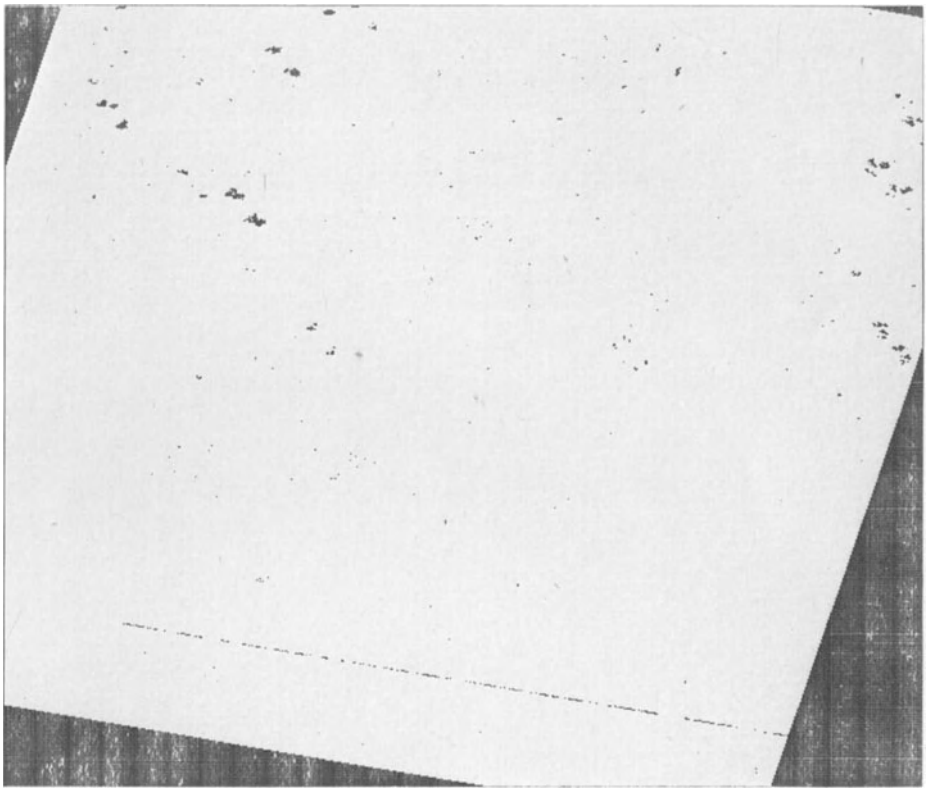


Fig. 9.2. The clouds/cloud shadows class

useful in representing the information in pictures of this complexity, then they should be useful in a large variety of picture processing applications.

Second, there is *a priori* reason to believe that there is syntactic structure present. Statements about the scene such as, "The urban-commercial area is surrounded by the older residential area which is surrounded by the suburbs", are verbal expressions of multidimensional syntactic relationships. The question is

Table 9.1. Land use classification key

Land use	Symbol
Commercial	C
Inner City	I
Suburban	S
Concrete	X
Open Grassy	O
Wooded	W
Water	1, 2, 3, or 4
Cloud	Blank
Cloud Shadow	*

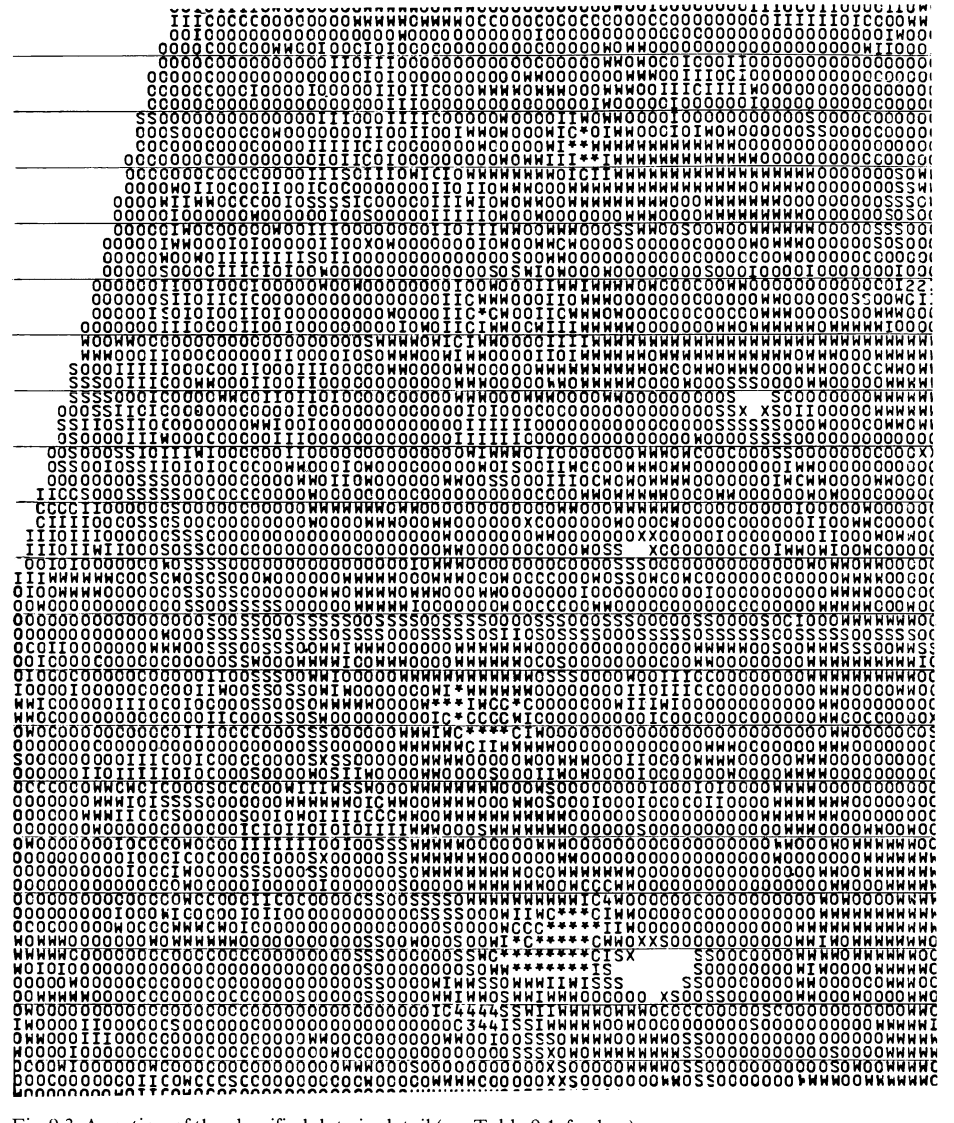


Fig. 9.3. A section of the classified data in detail (see Table 9.1. for key)

whether these abstract verbal ideas can be transformed into useful algorithms and whether web grammars provide a useful formalism for implementing such algorithms.

Finally, as we pointed out earlier, the land-use analysis of this data represents a significant practical application. The ability of satellites to gather data about the terrestrial environment has far outdistanced the ability of computers and man to analyze it. But since such analysis can provide important information on crops, minerals, and urbanization patterns, there exists an urgent need for new, more sophisticated, and more effective analysis techniques.

9.2 The Model

The spatial dependencies of the various classes were noted by TODD and BAUMGARDNER in verbal form. Some manual preprocessing based on these dependencies was used to improve the accuracy of the classification. These spatial dependencies are clearly evident from Figs. 9.1 and 9.2. A hierarchical graph model for these relationships can be constructed as shown in Fig. 9.4. This model was constructed directly from observations of the classified data and of aerial photographs of the corresponding region. For simplicity, there are some relationships between the entities in the figure which have not been included; for example, the fact that the SCENE is made up of the EARTH and CLOUD PAIRs (i.e., clouds and shadows). The CLOUD PAIRs *obscure* the EARTH, so a relation 0 for "obscures" could be shown linking CLOUD PAIRs to EARTH. Also, if a pair of entities are related, then their descendants are also related. However, these relations are shown only at the level they first occur. The form of this diagram is the same as the derivation diagram for a web grammar [1].

This scene consists, at the highest level, of the EARTH obscured by CLOUD PAIRs. Each CLOUD PAIR consists of a CLOUD and a SHADOW, related by a distance-and-angle R. A CLOUD consists mostly of points classified as clouds (blank) but also points classified as concrete (X) and as suburban (S). This confusion seems to arise because both concrete and clouds are highly reflective. The suburban class is a mixture of concrete and grass. A SHADOW tends to consist mostly of points classified as shadows (*) but also points classified as commercial (C) and inner city (I). The confusion here seems to occur because the commercial and inner city classes consist largely of asphalt rooftops with low reflectance.

[1] For more information on web grammars, the reader may consult Chapter 1 of this book and [1.74, 75].

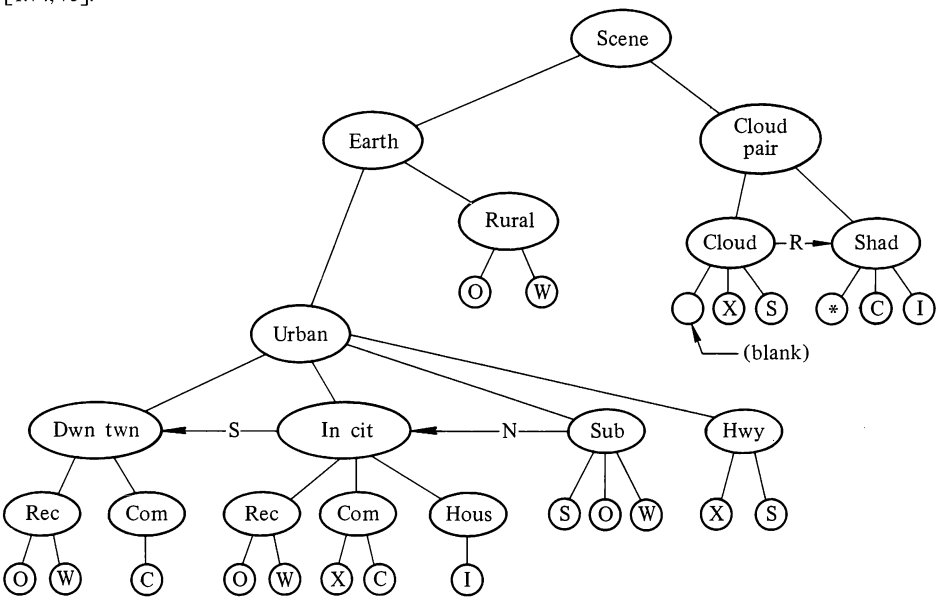


Fig. 9.4. A hierarchical graph model of the scene

The EARTH consists of URBAN and RURAL areas. The RURAL area consists of open grassy (O) and wooded (W) areas. The URBAN area consists of the DOWNTOWN area, surrounded by the INNER CITY area, with nearby SUBURBAN areas and a system of HIGHWAYS. The DOWNTOWN area is characterized by the fact it contains the largest concentration of commercial land use. The INNER CITY area surrounds the DOWNTOWN and contains a high concentration of inner city points. The SUBURBAN and HIGHWAY areas are both near the DOWNTOWN and contain mostly suburban-classified points. They are distinguished by the fact that HIGHWAYS occur in linear patterns.

This model is constructed from a fairly intuitive notion of what a city is and how it is laid out. In that sense the model is general and could be applied to many urban areas. The use of a web grammar model up to this point in the analysis is not particulalry significant aside from the fact that the derivation diagram provides a convenient formalism for representing relationships.

This model is now used to guide the analysis of the picture. In essence this analysis is an attempt to verify the model and to make it more specific. Each subentity of the picture and each relationship can now be elaborated and tested separately.

9.3 Details of the Analysis

9.3.1 Clouds and Shadows

As pointed out in the previous section, clouds and shadows are characterized by the fact that a cloud is a bright area which has associated with it a congruent dark area (the shadow) at a certain specific distance and orientation. It is found that the bright areas are generally classified by the pointwise classifier as clouds, but sometimes they are mistakenly classified as concrete or suburban. The dark areas are usually classified as shadows but sometimes as commercial or inner city. Examples can be observed in Fig. 9.3.

At this point there are several types of analysis which could be performed. Knowledge of web grammar theory helps to understand the differences between the types and finally to select a proper one.

If the relationship R between clouds and shadows is not known from some other source such as sun angle and height of clouds, the analysis process must determine R by some type of cross-correlation operation. For one-dimensional patterns, this process of finding similar patterns an arbitrary number of symbols away in the pattern can be modelled by a context-sensitive string operation. This is almost certainly also a context-sensitive problem in a web system for two-dimensional patterns such as clouds and shadows.

If the relation R is known from other sources the processing is simplified. Shape matching of a cloud and its partner shadow can be performed to confirm that they actually are pairs. This process (illustrated in Fig. 9.5) is similar to the one-dimensional problem of being able to recognize all strings of the form ww^R (where w^R is w reversed). This is known to be a context-free operation in a string system and thus is probably also context free in a web system.

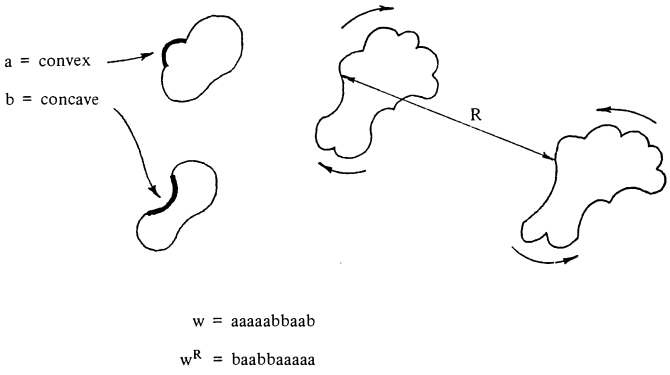


Fig. 9.5. Context-free recognition of clouds

Finally, an even simpler type of recognition would be to check a finite radius around a given cloud point. If there are more cloud points, this verifies the classification of the original point. Then an equivalent area at a distance given by the relation R can be searched. This process is illustrated by Fig. 9.6. Since the radius r of the area searched is finite, the process is essentially finite state.

Before discussing the method actually chosen and the effect on the data, it is useful to review what is already known about these algorithms without even looking at the data. First of all it should be noted that each of the algorithms is intuitively motivated and heuristically constructed. That is, the actual recognition algorithms are not some type of formal web automation. They might be considered web parsing algorithms but they are really not even that. Rather they are syntactically based translators. This means that the output is not a decision ("yes, it belongs to L", or "no, it doesn't") but rather the output is another picture modified from the input picture according to some syntactic rules. The web grammar theory contributes in the sense that it provides general information about these syntactic rules. The same form of rules may have been used by the designer in the past, so the same recognition algorithm can be used. Also general results on recognition time, storage requirements and accuracy can be utilized to make a choice between several algorithms.

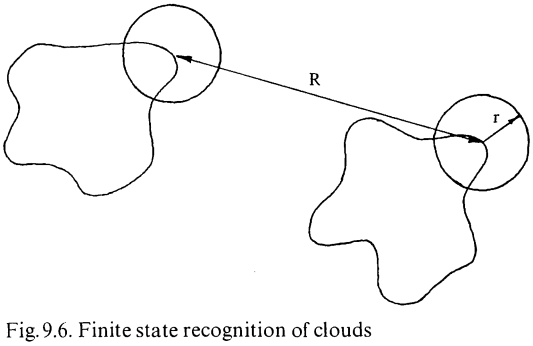


Fig. 9.6. Finite state recognition of clouds

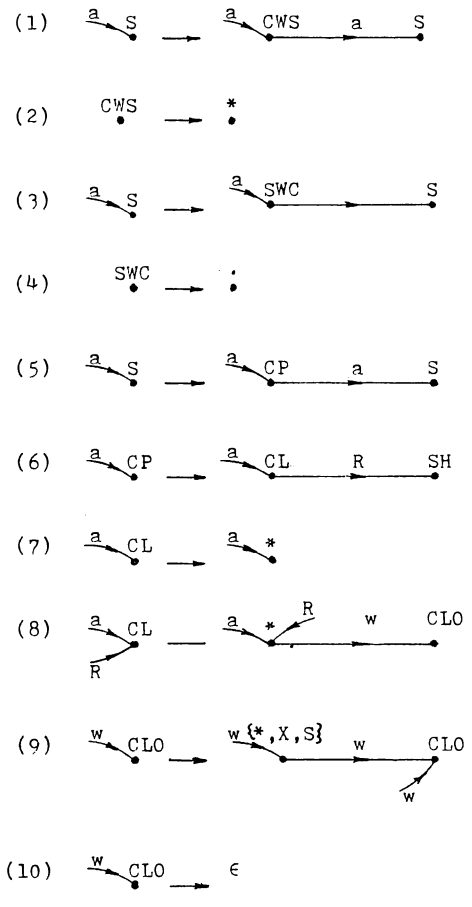


Fig. 9.7. The cloud/shadow grammar

In the case of the clouds and shadows the simplest possible algorithm was tested first. The picture is scanned left-to-right and top-to-bottom. Whenever a shadow point is encountered, the translator searches a finite window at a distance and angle away given by relation *R*. If a cloud point is found, the pair qualifies as a cloud-shadow pair and neighboring *, X, and S points within the window around the cloud are also interpreted as cloud points. Likewise if the pattern qualifies as a pair, neighboring, C and I points within a similar window centered at the original shadow point are classified as shadow points. Note that the notation has changed from Table 9.1 so that cloud and shadow points are now represented by * and . respectively.

A web grammar depicting this analysis is shown in Fig. 9.7. Rules (1)–(4) generate cloud and shadow points that are not part of a pair. These points will be regarded as noise, not as true clouds and shadows. Rule (5) shows that a cloud-shadow pair (CP) can occur anywhere in the picture (the relation "a" represents "arbitrary" relationships). Rule (6) shows that a cloud-shadow pair consists of a

cloud and a shadow separated by the relation R. Rules (7)–(10) show how a cloud can occur. The corresponding rules for a shadow are similar and are not shown. Rules (7) and (8) show that any cloud must contain at least one * point. Rule (9) shows that once one * point is detected, then any point labelled *, X or S occurring within the window (i.e., within the relation w) is classified as a cloud point. Rule (10) terminates the search when the entire window has been scanned.

This grammar models the essential features of the recognition algorithm and therefore is good for illustration, but it is not complete in detail. Some complexities are buried in the relation w which models the window. A grammar which modelled this window in detail would contain a sequence of counting states that simulate the scanning of the window. This more detailed grammar would contain a few more nonterminals and rules but would not better illustrate the complexity of the recognition process. As long as the window is of finite size, a regular-linear grammar can be found.

The results of applying the recognition algorithm to a small section of the data are shown in detail in Figs. 9.8 and 9.9. The points labelled *, X, S are clouds, whereas ., C, I are shadows. It is clear from this diagram that the size of the clouds and shadows has been increased. While there is no absolute way to check the

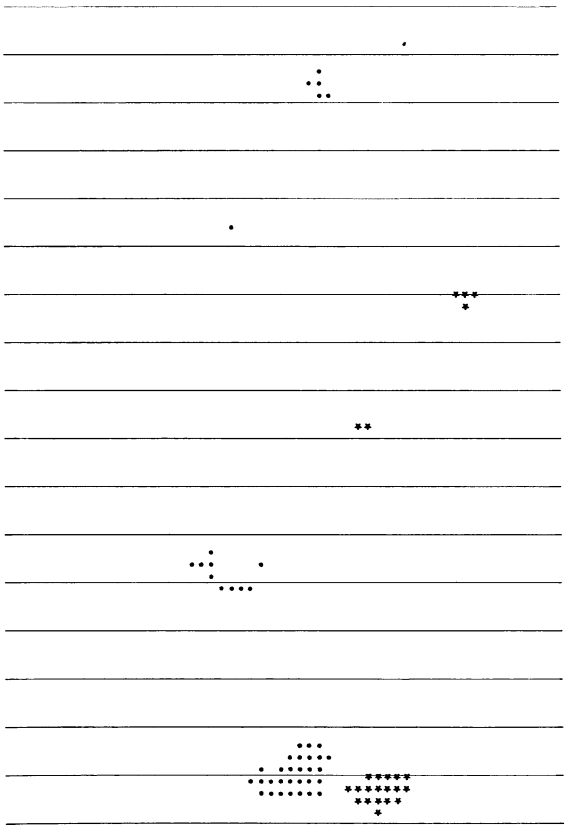


Fig. 9.8. Clouds and shadows before processing

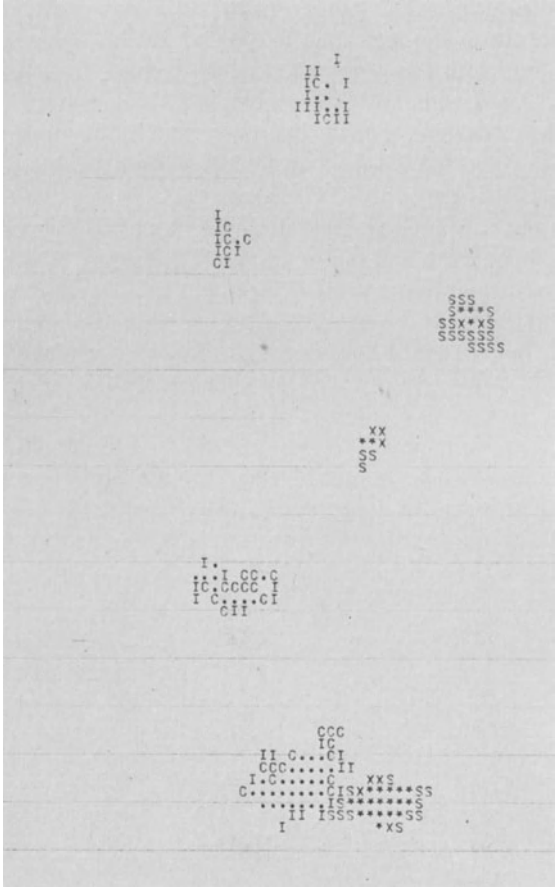


Fig. 9.9. Clouds and shadows after processing

accuracy of this enhancement, inspection of Fig. 9.3 shows, for example, that points classified as concrete and commercial points appear with high probability around the edges of apparent clouds in an otherwise rural (mostly O and W) area. Since there is also evidence that the spectral responses of these areas are similar (i.e., highly reflective), the enhancement is probably correct most of the time. Figure 9.10 shows the results for the entire LANDSAT frame. The algorithm is effective in separating the clouds and shadows from background noise. Note particularly the case of the scanner error (linear pattern near the bottom of Fig. 9.10 b). A few anomalies have been introduced by the algorithm however. In Fig. 9.10 a, notice the perfectly rectangular "cloud" just above right-center.

The role of the grammar in this case has been to model an otherwise ad hoc recognition or context analysis process. The web grammar model was used to gain insights into the complexity of several possible recognition algorithms and provide a fairly accurate model of these clouds and shadows as judged by the recognition performance.

Fig. 9.10 a. The cloud/shadow class after processing

Fig. 9.10 b. Points removed from the cloud/shadow class

9.3.2 The Downtown Area

The downtown area is the center of the urban complex and as such provides a focus for the recognition of patterns associated with the city. The urban complex may be characterized by a web-grammatical rule such as shown in Fig. 9.11. The symbol U represents the urban complex, and it can occur anywhere in the scene (arbitrary relation a). The right-hand side of Fig. 9.11 shows how the urban complex is structured. It consists of a downtown area (DWN) surrounded by (the relation s) the inner city area (IC). The suburban area (SUB) appears in patches outside but nearby (the relation n) the inner city. The highway areas (HWY) occur throughout all parts of the downtown, inner city, and suburbs in linear patterns and extend out into rural areas (the relation t).

Because some of these relationships are fairly well defined, the other areas can be found more easily once any one of the areas is located. In general when two patterns are connected with a known relationship it makes sense to locate the better defined pattern first and then try to recognize the other. In this case the downtown area is best defined. It is characterized by being the largest concentration of commercial land use. It can be represented by a stochastic web grammar which generates entities containing a high percentage of commercial points. Such a grammar is shown in Fig. 9.12. In this grammar the parameters α_1, α_2, β, and γ associated with each rule give the probability of applying that rule given DWN ($\alpha_1+\alpha_2+\beta+\gamma=1$). Rule (1) generates commercial points; rule (2) generates points with other arbitrary labels (A). Making $\alpha_1 \gg \alpha_2$ ensures that the entity generated will consist predominantly of commercial points. Rule (3) allows the downtown to consist of several nearby (relation n) entities. Finally rule (4) terminates the generation. Thus the grammar in Fig. 9.12 generates mostly commercial land use clus-

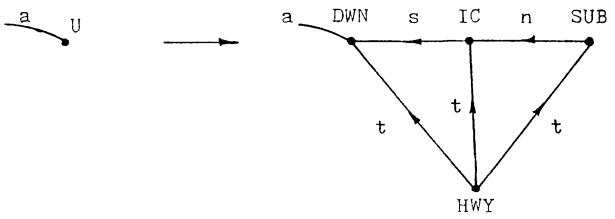


Fig. 9.11. Web grammar rule for the urban complex

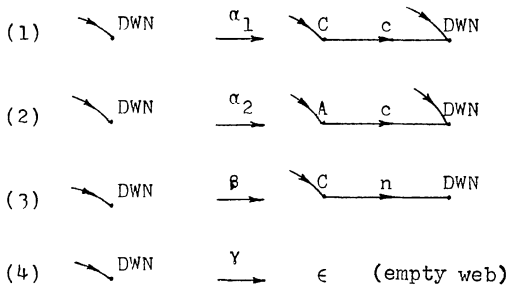


Fig. 9.12. Downtown grammar

ters, which crudely approximates the nature of the downtown area. One aspect of the downtown that the grammar does not model is the fact that the density of commercial points is dependent on the distance from the center of downtown. This means that the entity near the center of the city will be larger and denser than others around it. This could be modelled in the grammar by ensuring that other parts of the urban complex grammar cannot generate larger entities. Thus the recognition algorithm searches for the largest connected cluster of commercial land.

Finally there is a problem of how to represent the results of the downtown analysis for further processing of inner city and suburban areas. It is impractical to check every potential inner city point against every downtown point to see whether it is already part of the downtown. A more practical solution is to represent the outer boundary of the downtown area so that it can be determined if a point is inside or outside the boundary. This could be done accurately by using Fourier or polynomial shape descriptors; but if precise representation is not particularly important, the downtown may be represented by a circle with center at the center of mass of the entity and with radius $\sqrt{A/\pi}$, where A is the area of the entity. The result is shown in Fig. 9.13. The algorithm is fairly effective in representing the downtown area despite the crudeness of the approximations made.

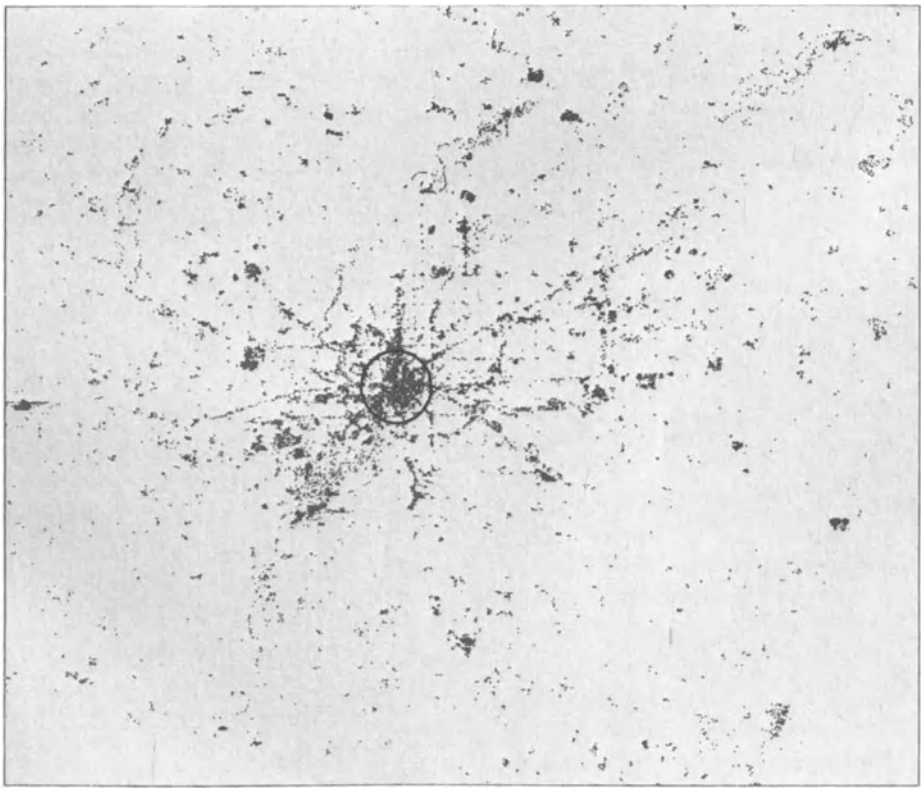


Fig. 9.13. The downtown

The grammar used in this case is context free, and it plays a descriptive modelling role similar to that of the grammar in the clouds/shadows analysis. By analogous methods, we have also been able to model the inner city surrounding the downtown [1.75].

9.4 Inferring a Grammar for the Highways

In many instances, the pattern of highways in satellite-collected imagery is the feature with the most obvious syntactic content. In this section we illustrate an approach for synthesizing grammars for the case of the highways in the data set we have been considering.

Our procedure will be to select a "training set" from which inferences about the grammar can be made, hypothesize a set of highway primitives, determine a set of predicates for detecting these primitives, formulate a set of low-level grammar rules, and proceed to the development of increasingly higher level rules.

As previously, the lowest level primitives are taken to be the classified pixels themselves. The predicates we need for working to higher level constructs are not known but can be assumed to be of a fairly simple form due to the "linear" structure of the highways. Several candidate forms will be used to analyze the

Fig. 9.14. Highway inference set

scene, and those judged to produce results which are most suitable will be retained.

The training or "inference" data set selected for use here is the portion of the frame shown in Fig. 9.14. The actual highways, needed for supervised training, were manually drawn in by referring to aerial photographs of the scene taken at a much lower altitude. The pixels which should be classified as highways were entered into the data with the symbol Q as shown in Fig. 9.15.

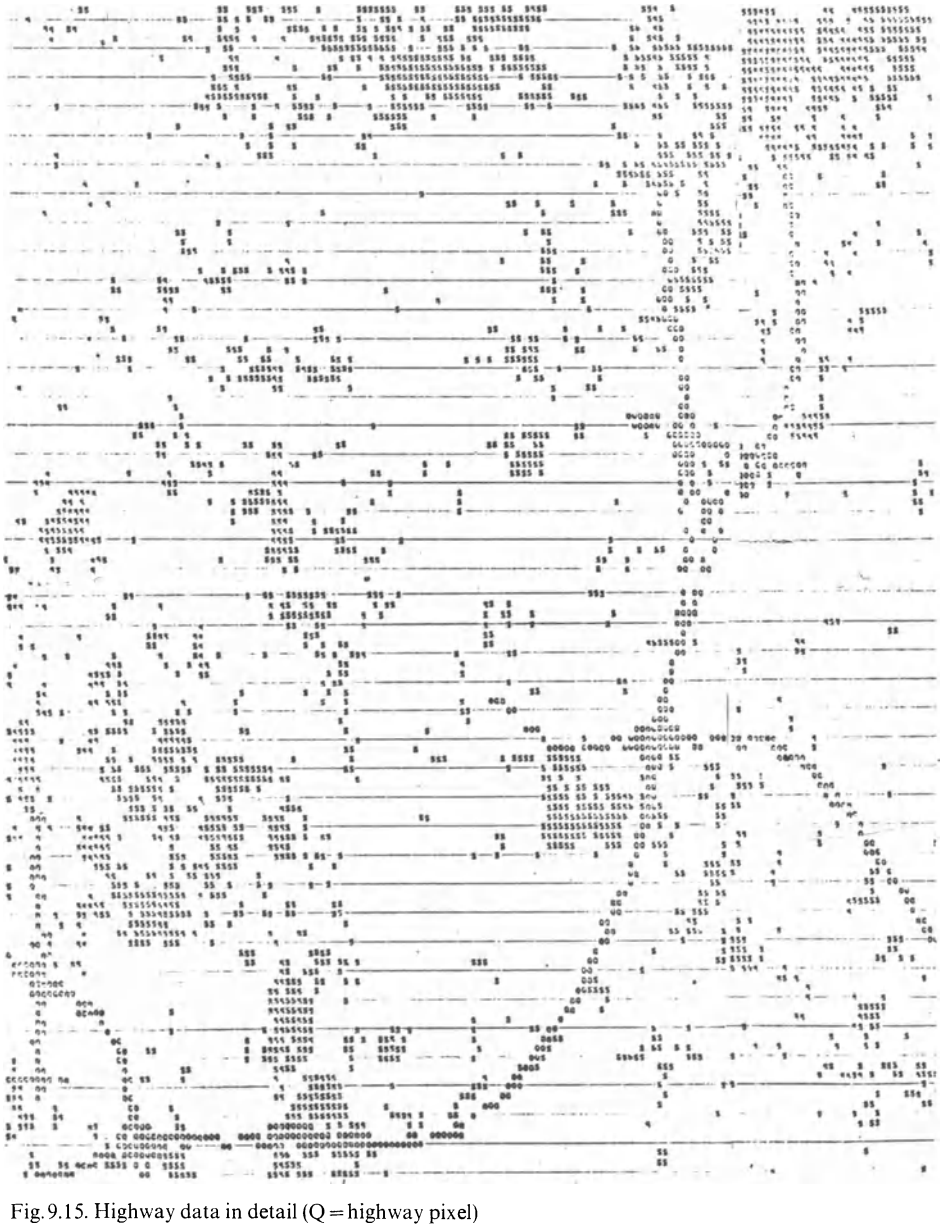


Fig. 9.15. Highway data in detail (Q = highway pixel)

Pattern	Ideal distribution	Symbol	Explanation
1		A	Vertical line
2		B	Upward slanting line
3		C	Horizontal line
4		D	Downward slanting line
5		E	Very dense
6		BLK	None of these

Fig. 9.16. The highway primitives

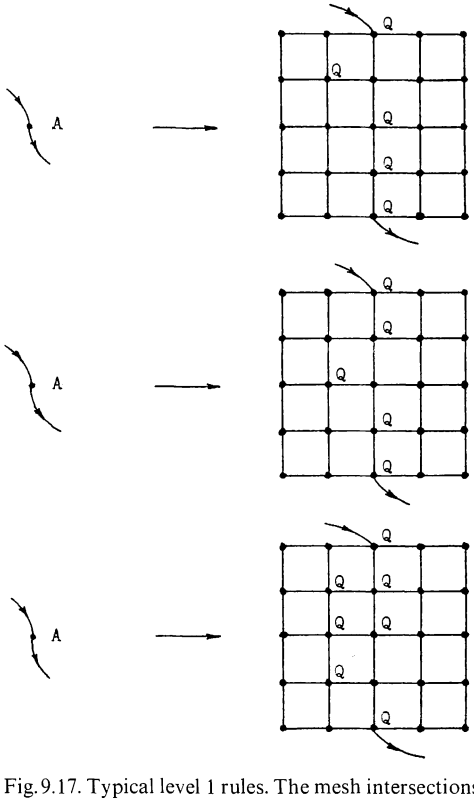
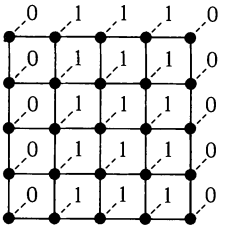


Fig. 9.17. Typical level 1 rules. The mesh intersections correspond to image pixels

The first step in the analysis is to learn the lowest level rules. The individual pixels are taken to be the primitives; the lowest level rules generate groups of pixels which are to be labelled in some desired pattern. A rule is useful in representing the picture. At this low level it is useful to represent patterns that roughly approximate over a small area those shown idealized in Fig. 9.16. The small area is selected, for convenience, to be a 5×5-pixel mesh. Thus, for example, the lowest level rules rewriting a vertical line segment A are of the type shown in Fig. 9.17.

It can be seen from Fig. 9.17 that there are many patterns of the twenty-five pixels that approximate a vertical line. One way of compactly representing all these rules is by means of a predicate. We use a function defined over the labels of the twenty-five nodes of the array such that if the value of the function exceeds a threshold then the pattern is accepted. An example of the specification of a predicate and calculation of a predicate value is shown in Fig. 9.18. The inference of the lowest level rules then corresponds to the search for predicates which accurately select examples of the desired patterns (e.g., vertical line) from the picture. Once the predicate set is determined, the grammar rules that model it are just all those whose right-hand-side arrays satisfy the predicate.

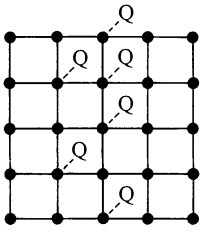
Coefficient array: c (i, j) =



Value of label: k (i, j) = 1 if pixel = Q
= 0 otherwise

Predictate: $P = \sum_{i=1}^{5} \sum_{j=1}^{5} c(i, j)\ k(i, j)$

Example pattern:



$$P = (0 + 0 + 1 + 0 + 0) + (0 + 1 + 1 + 0 + 0) + (0 + 0 + 1 + 0 + 0) + (0 + 1 + 0 + 0 + 0) + (0 + 0 + 1 + 0 + 0)$$
$$= 6$$

Fig. 9.18. Example of predicate evaluation

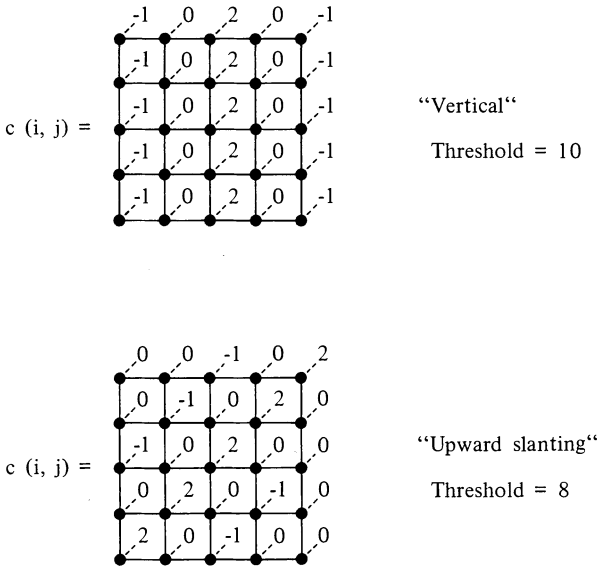


Fig. 9.19. Coefficient arrays for selected predicates

Note that nothing has been said about how the embedding edges are to be determined, because at this point they are not yet determined. They will be affected by the next higher level of inference. Those embeddings shown in Fig. 9.17 are merely illustrative.

All possible predicates adequately representative of the training samples can be used. In this case, for each desired pattern in Fig. 9.17 several possible predicates were selected and tried on the data. Examples of the predicates that were selected on the basis of best performance on the data are shown in Fig. 9.19. The performance of these predicates on the picture data is shown in Fig. 9.20 and a tabular evaluation is shown in Table 9.2. Note in Fig. 9.20 that the characters correspond to those used in Fig. 9.16. According to Table 9.2, we have succeeded in recognizing roughly half of the highway points while incurring a reasonably low number of "false alarms" (non-highway points identified as highway). We expect the higher level rules in the grammar to improve on these results by making still greater use of context.

Table 9.2. Performance of selected level 1 predicates

		Judgment by predicate Class	
	Class	Non-highway	Highway
Judgment by analyst	Non-highway	18514 93.9%	579 2.9%
	Highway	316 1.6%	320 1.6%

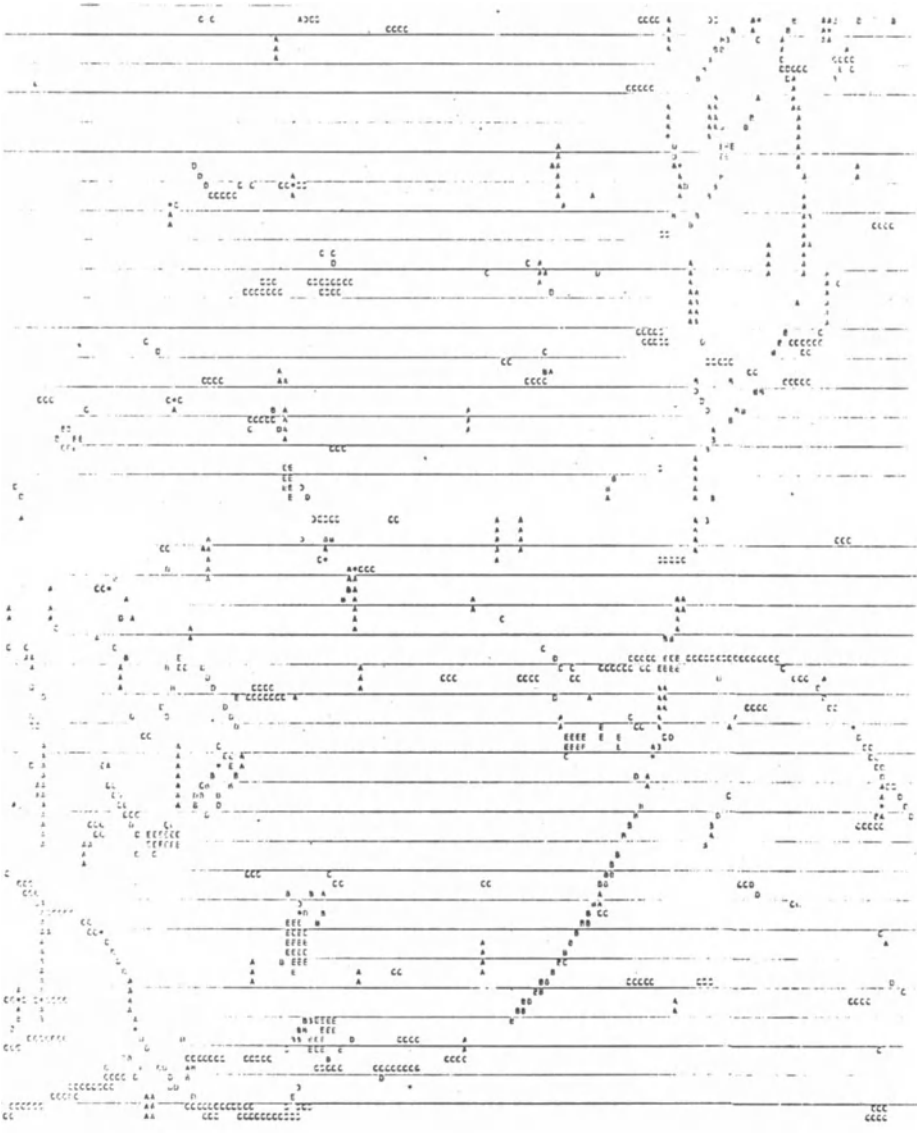


Fig. 9.20. Primitives found in the training set

Once the set of first-level rules rewriting level 1 nonterminals as primitives has been determined, the same type of process can be repeated to learn the next higher level rules. This time level 2 nonterminals generate groups of level 1 nonterminals. However, the level 2 and level 1 nonterminal sets need not be disjoint, so that a single nonterminal may be in both sets and may occur on both sides of a rule. Thus recursive rules are possible.

The form of the rules being sought in this second level is given in Fig. 9.21. Note again that the embedding edges in Fig. 9.21 are not yet fully specified. The

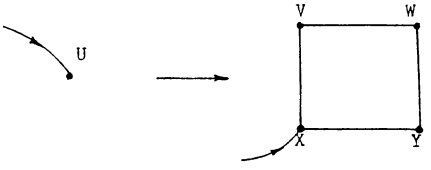


Fig. 9.21. Form of level 2 rules

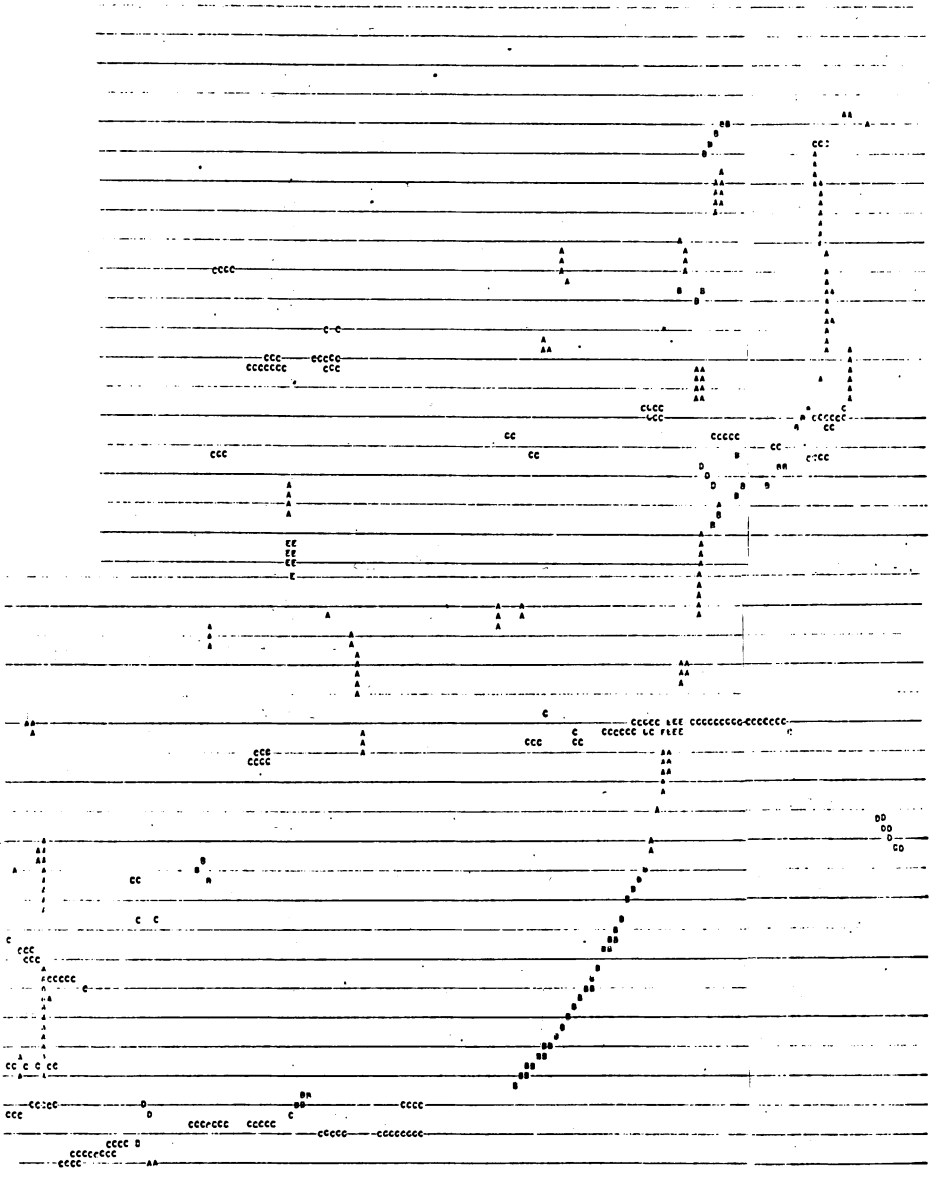


Fig. 9.22. Highway points found in the 4-tuples

embedding edges shown provide only a position reference point. Also this form is only one of many possible. If it represents the data well it will be used; if not, another form will be used.

Once the form has been selected, the data must be searched to see which level 1 nonterminals are most likely to occur in the V, W, X, and Y positions. There are 1080 possible different rules of this form. There are 5 different possible level 1 nonterminals in position X (blank is excluded) and 6 possibilities in each other position (see Fig. 9.16). A search was performed on the training data and 210 of the 1080 possible patterns were found to actually occur in the neighborhood of the true highways. Using this set of rules the entire picture was processed, resulting in the classified highway pattern shown in Fig. 9.22. The tabular evaluation for this analysis is given in Table 9.3. The results are overall slightly better than those in Table 9.2. The major difference is the significant reduction in the number of non-highway points that are classified as highway points.

Table 9.3. Performance of 210 level 2 rules

	Class	Judgment by rules class	
		Non-highway	Highway
Judgment by analyst	Non-highway	18938 96.0%	152 0.7%
	Highway	387 2.0%	249 1.3%

An analysis was then made to determine if the set of 210 rules could be reduced without significantly affecting the accuracy of the classification. Only those rules with significant probability of occurrence were retained. There were 69 of these rules and their right-hand sides are shown in Fig. 9.24 in the format shown in Fig. 9.23 (recall Fig. 9.21). The result of using this set of rules on the data is shown in Fig. 9.25 and Table 9.4. There is relatively little difference between the performance of this reduced set and that of the larger set.

The left-hand sides of Fig. 9.24 do not yet have the nonterminal symbols assigned. This must be done by returning to the training set and observing where in the picture the pattern on the right-hand side occurs most often. This is essen-

Table 9.4. Performance of 69 level 2 rules

	Class	Judgment by rules class	
		Non-highway	Highway
Judgment by analyst	Non-highway	18979 96.2%	114 0.6%
	Highway	431 2.2%	205 1.0%

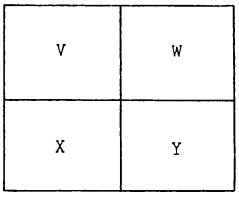


Fig. 9.23. Code for Fig. 9.24

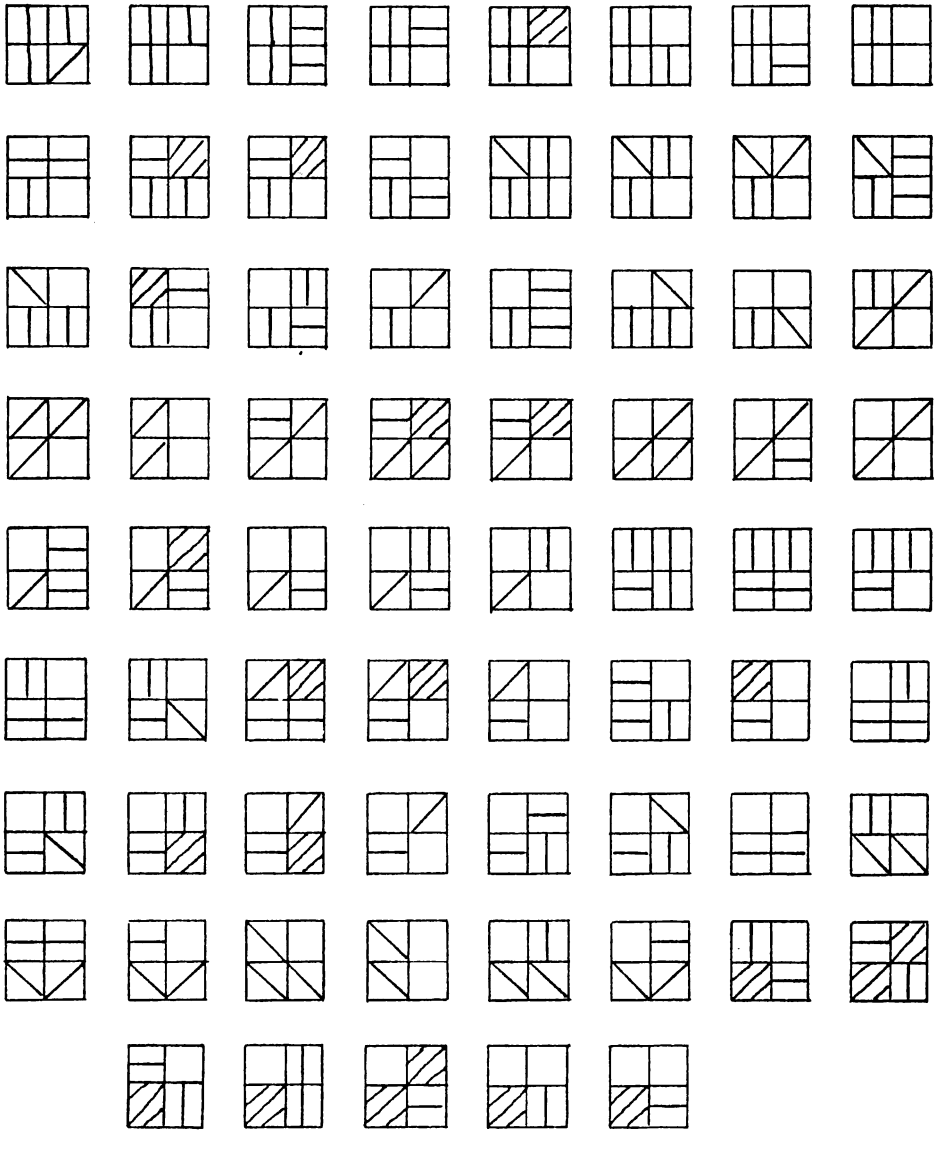


Fig. 9.24. The most-representative 4-tuples

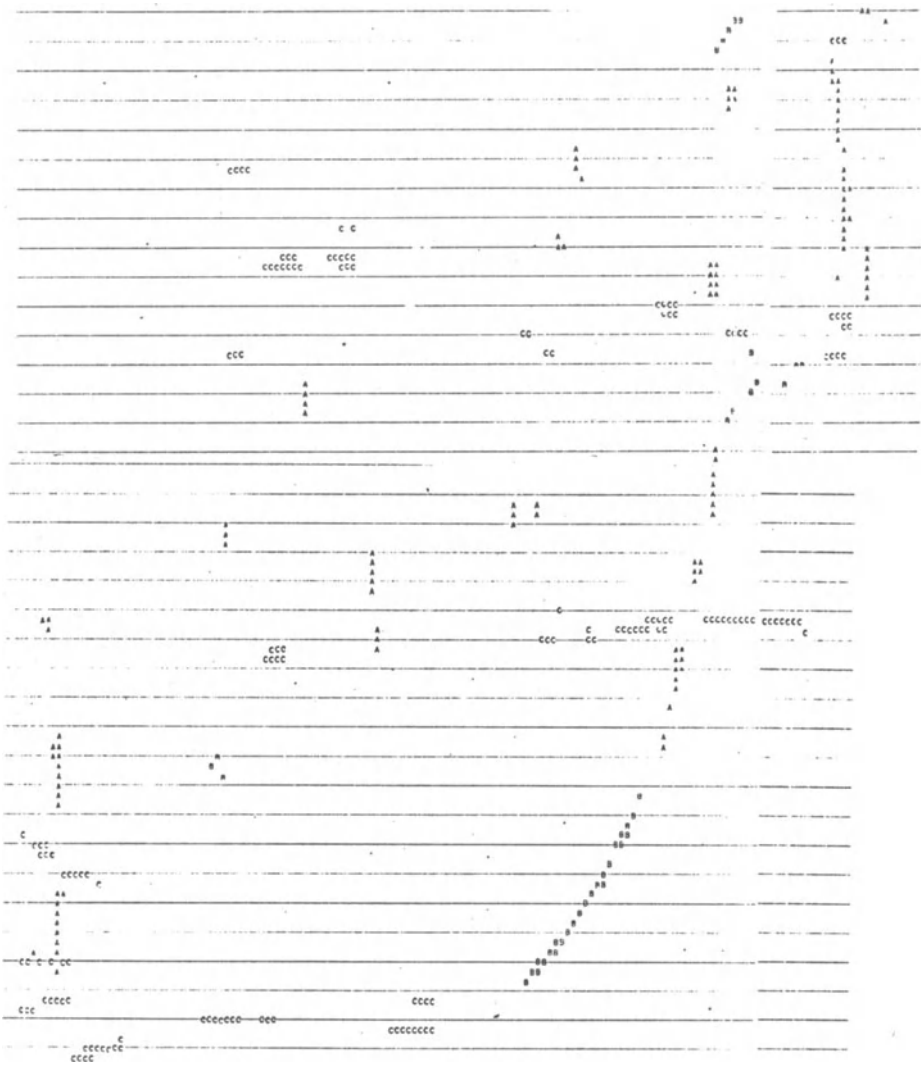


Fig. 9.25. Highway points found in reduced set of 4-tuples

tially the same method used to assign symbols to level 1 nonterminals. This process can again be the source of recursive rules. For example, the pattern shown in the rule in the upper-left-hand corner of Fig. 9.24 consists of three level 1 vertical line nonterminals. If it occurs most frequently in a vertical stretch, the entire pattern (i.e., the left-hand-side nonterminal) can be called a vertical line and assigned the same symbol. The result is a recursive rule that can generate vertical lines of any length.

The last part of the web rule learning process to be considered is the embedding. Knowledge of the level 2 rules should give whatever information is needed

about the context of a level 1 rule. These level 2 rules specify the context into which the level 1 nonterminals will be embedded and thus give the number and arrangement of embedding edges on the left-hand side of the level 1 rule. The naming of the relations and the mapping to the right-hand side of the level 1 rule must still be done by the analyst. These decisions reflect judgments not only about the connectivity of the overall patterns but also about the complexity of the computation required for recognition. These problems are only partly understood at this point and require more study before satisfactory choices can be made automatically.

The performance of the grammar we have developed to this point in the example leaves much to be desired because the "false dismissal" rate (highway points not recognized) is, for practical purposes, too high. Much of the problem can be alleviated by modifying the set of predicates applied at the lowest level of the analysis in order to reduce the false dismissal rate at that stage. This would certainly raise the false alarm rate, but at successively higher levels in the grammar we should be able to achieve a marked improvement in recognition accuracy, evidencing greater utilization of the syntactic content of the data which our visual sense and our intuition tell us is there.

9.5 Summary

The goals of this chapter were to construct web grammar models for complex image data and to demonstrate that these models can be useful in aiding the analysis (classification) task. To accomplish these goals, we developed an overall web model which described the syntax of a satellite-collected image. We developed subgrammars to describe the clouds and shadows, the "downtown" areas, and the highways. The grammars were useful because they provided guidance in the development of algorithms to recognize complexly structured land-use classes in the imagery which could not be successfully recognized based on their pointwise (spectral) properties alone.

Obviously we have only begun to scratch the surface. The grammatical inference process is very highly dependent on the "semantic teacher", i.e., the human data analyst interacting with the computer programs. And at this point, he has little more than his intuition and his eyesight to contribute to the modelling process. More automatic or at least more systematic means must be developed for grammatical inference in such highly complex image environments.

References

9.1 W.J.TODD, M.F.BAUMGARDNER: Information Note 101673, Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana (1973).

10. Industrial Objects and Machine Parts Recognition

T. VÁMOS

With 20 Figures

This chapter outlines the role of syntactic methods in industrial objects pattern recognition, emphasizing the advantages and special functions of the structural methods in the different phases of the problem solving concerned. A short review of the leading ideas as a logical historical-intellectual edifice shows the determination of the approaches by the specific goals, apparatus and project progress. A special project as a paradigm for the whole problem is reported in detail with references to general considerations. The hand-eye project hardware is described in detail; this is based on a well-equipped standard minicomputer, a laser-deflector input and a 6+1 degree of freedom output. The hardware is only an experimental realization device of the software which includes the special operating system for this task, a disc-based handling of library programs, a dictionary-type semantic information storage, and a man-machine dialog with the visualization of the different problem-solving phases. Most details are devoted to the pattern recognition, a template-matching type of preprocessing, the definition and finding of picture primitives and a context-free programmed grammar using some Bayesian parameters for fast heuristics. The handling of lists, semantic procedures and the analysis by synthesis are specially elaborated. The grammatical-structural method proves to be the most powerful device for pattern recognition of industrial objects combined pragmatically with other ones.

No unique approach exists for different problems. The human pattern and situation recognition sometimes uses all our five organs of sense combined, especially the eye which has a very complex nature to identify shape, dimension, relative position, general view, special detail, texture, color, etc. The human brain controls the orchestra of this sensory manifold backing on memory and logic which uses previous information, associations and inference. The much less powerful machine intelligence can even less rely on one particular tool and that is the justification of speaking not only about pattern recognition in general, but specially about recognition of aerial pictures, blood cells, chromosomes, chest x-ray photographs, fingerprints, alphanumeric characters, EEG and ECG diagrams, human faces and last but not least industrial objects. Specialties of the task define the selection of means. What are the special characteristics of industrial objects?

— Sharp, well-defined contours (edges);
— the contour-elements—as being machine-made—are simple curves, mostly lines and arcs;
— the differences between the object classes and individual objects are also well defined and suitable to be expressed by quantitative characteristics;
— industrial objects have also a structure which can easily be described;

— three-dimensional characteristics are essential, although recognition on the basis of two-dimensional pictures is possible in most cases.

If by a very rough classification pattern recognition methods are divided into statistical and structural ones, the above features conclude to the preponderance of structural treatment in the case of industrial objects. Preponderance rather than exclusiveness is stressed because some auxiliary processes and the use of stochastic grammars prove the weakness of all these classifications and "pure" approaches. Nevertheless industrial objects, e.g., machine parts and electronic components, are subjects of pattern recognition mostly for automatizing feeding and assembly tasks. This means that the recognition is aimed at

— identification of the object
— determination of its position and locations of the object parts
— recognition of potential essential differences (e.g., faults).

The structural-syntactic method is one of the possible ones for the first task, for the second—until now—the best and most reasonable, for the third a very effective tool combined sometimes with other processes. Identification based on texture, color, some easily measurable dimensions, shape characteristics can be used as shortcuts towards an effective structural treatment. To give a more detailed inspection of the problem, after a short review of the most important issues of the last decade a special project will be described with some references to general considerations.

10.1 A Short Review of Leading Ideas

Syntactic methods—at some level of complexity—could hardly be omitted from the industrial pattern recognition projects. On the other hand, as has been pointed out, because no sterile syntactic processes can be found, the success of a project needs a more pragmatic approach than a paradigm. Pattern recognition of industrial objects is closely related to the robot projects of artificial intelligence laboratories. Simplified problem solvers based on some characteristic measurements (dimensions, color, texture, etc.) are less ambitious than the highly intelligent grammatical search. Consequently most of the ideas applied until now are the results, or at least have their roots in the efforts, of a few schools. MIT [10.1], Stanford Artificial Intelligence Laboratories [10.2, 3] and the Stanford Research Institute [10.4] are the forerunners. After a little while the Department of Machine Intelligence of the University of Edinburgh [10.5] and the Electrotechnical Laboratory Tokyo [10.6, 7] followed suit. Recently in Japan the Hitachi [10.8] and Osaka University groups [10.8–10] were the dominant pioneers; in the US the Urbana [10.11, 12], Berkeley [10.13] and Purdue Universities [10.14]; in Europe the Moscow [10.15], Kiev [10.16], Karlsruhe [10.17], the Milan [10.18] and Budapest [10.19] groups joined the research activity. A brief summary of all these projects shows an evolution of very similar philosophies, as a logical consequence of problems encountered. The start has been always the identification of polyhedra, beginning after the early influential thesis of ROBERTS [10.20] with the classical work of GUZMAN [10.21, 22]. This is the simplest form of 3D objects

with sharp vertices and edges, flat, uniformly reflecting surfaces. Edge detection techniques had to be developed first. An excellent summary of these [10.23–31] contains all relevant achievements; it should only be referred here to the basic classes of edge detection methods used mostly for industrial objects.

Gradient methods calculate point by point the light intensity differences and the alignment of the results to form continuous boundaries.

Template methods correlate the grey-level patterns with some simple templates.

The techniques of handling line-elements and to complete them into full lines (straight or curved) between vertices, nodes or similar limits show only small variations. The next challenging step is the analysis of regions within the boundary of the edge lines the recognition of the interrelations of surfaces and complex bodies. Following GUZMAN especially at MIT, HUFFMAN [10.32], CLOVES [10.33], ORBAN [10.34]; WALTZ [10.35] and FALK [10.36] at Stanford developed these ideas to separate different connecting bodies, even hidden by each other in some part, and to discriminate the bodies from their background and shadows. A significant step has been the attempt to handle curved surfaces by AGIN [10.37].

A method using an opposite-directed philosophy is *region-growing*. This can be applied with advantage in scene analysis where the meaning of picture-boundary is a priori definite (e.g., outdoor scenes with sky at the picture top, a workplace on a table) (YAKIMOVSKY [10.38]).

As edge detection was first applied on uniformly illuminated objects scanned point by point, the direct *detection of surfaces* led to the idea of beam or plane-like illumination yielding edges or curvatures by the change of direction of the light beam on the object (Fig. 10.1) and defining the surface by parallel bundles [10.17, 39].

Putting these picture elements together, as edge-types, surface-types, the grammatically described graph structure has been the obvious outcome. The identity of objects was embodied in some kind of parsing tree containing mostly not only different picture elements as nodes and connections as branches, but some more *characteristic relations* concerning the type of concatenation [10.22, 40–42].

The philosophy of most early works was the creation of an autonomous system which can describe any object of a restricted type without any special

a.) intersection of one light-plane

b.) input picture

c.) preprocessed input

Fig. 10.1 a–c. Pattern preprocessing by ETL Tokio

previous knowledge and human intervention. It became soon evident that the inclusion of *a priori knowledge* is not only very useful, but also a challenge for a more intelligent solution [10.1,3,4,9].

Semantic information can be handled by a computer only as a content of a syntactic structure and that was the point where the elegance of the syntactic method showed its superior possibilities. Both pioneering achievements of artificial intelligence—the procedural grammars of MINSKY, PAPERT [10.43], HEWITT [10.44] and WINOGRAD [10.45] and the conceptual understanding of SHANKS [10.46], WILKES [10.47,48]—fertilized the field of real object recognition. The next logical step was the inclusion of man-machine interaction potentialities. As WINSTON points out: a shrewd teacher is needed for a practical robot system. A display dialog offering the possibility to correct the computer operation and to introduce new knowledge and ideas for shortcuts of the problem has been the consequence of this reasoning [10.1,3].

The *dialog* system is another reason to use grammars. Information retrieval for visualization of pattern elements and complexes such as displaying lists, enhancing, correcting information on structures, translation of understandable human communications and vice-versa can hardly be imagined without the framework of the description structure for the original task [10.49,50].

As coming nearer to a realistic, effective and nevertheless sophisticated robot the *teaching-operating* activities received a more defined contour. The teaching period results in a general model; an idealized description and the differences of the description reached by the special operation activity are measured as individual features of a general concept [10.3,9,51]. Stochastic methods are most apt for this purpose, and starting from a grammatical structural description the stochastic grammars [10.24,52–55] are the adequate answers.

Simple school exercises can do well with bottom-up analysis, treating one or two trees, not very complex ones and unlimited in computer time. As real time, real environment operation is needed, only combined, top-down-bottom-up parsing is practical, linked with a heuristic fast search in the most feasible directions. These heuristics can be helped by semantic information and the use of the fuzzy concept suggested by ZADEH [10.56,57] allowing indefinite statements similar to our real life judgments. Fuzzy grammars containing coefficients for advisable directions on a graph endowed the structural methods with a new device towards human-like (i.e., more feasible) inference. A *goal-directed monitor* [10.4] used by the SRI group is another aid for decision making in the parsing process. A figure of interest expressing the importance of a branching from the point of view of the goal (finding the object) is attributed to each branch. The idea of weighting the parsing tree is somehow synonymous with the coefficients obtained by a learning process. Some more characteristic details are always observed by a recognizer in spite of different views or illumination conditions, some less characteristic ones are blurred during the feature extraction procedure. Decisive differences between objects to be recognized in one specific scenery can also be weighted and this organizes a "conscious" search instead of a "stupid" one [10.58]. By this sophisticated goal monitoring, the structural methods receive a real intellectual power.

Objects are mostly viewed by two-dimensional imaging devices. A *three-dimensional* optical or tactile scanning can also be imagined, but considering sen-

sorial and computational difficulties this can hardly be applied for a real-time practical system. A two-dimensional picture should be sufficient for recognition, as is natural for the human being; however the determination of the spatial locations is left for 3D calculations. This is the general approach applied everywhere, but the point where the transition of the 2D–3D view should occur is different. The problem depends also on the kind of a priori knowledge. If the object is identified and the dimensions are known before, on the basis of the difference in location between the real and model object, all needed points are computable. The checking of these calculations occurs for most applications by tactile sensors. The cited invariance of structural descriptions allows an easy identification of the object details and attribution of real measures to dimensions seen in perspective.

Much attention has been devoted to *transformation* problems. Three-dimensional objects appear from different, unexpected views, especially for an optical device which is either fixed or mobile but only in fixed directions and a decision should be made of which one [10.5, 8, 17, 23, 59, 60]. Structural descriptions are inherently invariant, a very important argument to be applied. This invariance is of course not unlimited; a pencil from one angle of sight looks like a rod, from the other extreme as a circle, but these two descriptions are invariant within very broad limits. The way of handling this particular problem will be referred to later.

Concluding this brief review of the dominating ideas found in most advanced three-dimensional pattern recognition projects and especially industrial-oriented eye-hand undertakings, we must not forget the importance of the *software support* which is an indispensable requisite of real system building. These are

— a flexible, thrifty, real-time monitor
— one or several languages for different level of system design and operation
— a data structure.

10.2 Levels of Knowledge

While abstaining from being anthropomorphic it is always worthwhile to study human analogies. The hierarchical structure of intellectual activity led also the first attempts of pattern recognition to the obvious idea of division between learning and active period and task. A deeper and more practical consideration realized four levels.

1) General *system building* for all possible tasks of a certain activity (e.g., recognition and handling of industrial objects, i.e., intelligent aid for robot-manipulators). This includes a very extensible system being able to describe a broad class of objects and activities, programming aid for local technical staff with no expert knowledge of computer science. With respect to the human analog, this is a typical system engineering task.

2) System building and learning for a *specific workplace.* This should also be flexible considering the fact that in the lifetime of the workshop installation the arrangement and especially the products can vary several times. The analogy here is the production design.

3) Teaching *one task* (generally a complex one) including the understanding of *one scenery*. That is, the production engineer's or foreman's duty to teach an unskilled worker to do his job.

4) The *actual system activity*. Transfer or assembly or both of a limited number of objects in at the very most a slightly different sequence. The task is rather the identification than the general recognition of the object itself, finding position, orientation, deviations (faults), a simple reasoning to select the next activity, adaptive control of operation (the choice of tools included), the adequate grasping, the transfer while avoiding obstacles, fine fitting and mounting, asking for help if the problem to be solved is beyond comprehension. This is just the unskilled worker's job supervised by a foreman.

The project cited is ambitious to serve all four levels. To *serve* and not autonomously solve—this refers to the man-machine interaction where the optimal distribution of activity between man and computer is also an object of the research activity and should be very flexible too, depending on the nature of the given task, and financial and technical considerations of the given application and period.

The *range of computer* to be used is also variable. A few years ago only level 4 could be imagined for a minicomputer, all higher activities were designed for a medium-big one. New powerful minicomputers are able to satisfy level 3 and are penetrating the range of the highest ones too. In the near future some functions of level 4 can be delegated to distributed microprocessors. The other object of research is the optimization of these hardware potentialities, the combination of different computing levels included. This latter contains also the problem of economical software interfaces.

10.3 Hardware

This defines the input, the output and the intermediate dialog help of the PR process. A short summary of the equipment used is given here emphasizing the important fact that this particular system is not a prototype for further possible manufacturing, but an experimental device to study the problem and the solution methods under nearly real circumstances. This consideration has defined a series of the recent hardware solution.

10.3.1 Input

The designer has a choice of visual and tactile input. Returning to the human analogy the combination of the two can be regarded as a possibly complete weapon, the visual input carrying more information to the recognition, the tactile as more bounded to mechanical handling of the last fitting operations. Acoustic waves can play an intermediate role.

Visual input is generally based on TV or laser scanner. More accurate devices, e.g., image dissectors, are too slow and expensive for industrial use. TV-*input* is the most feasible one from the point of view of economics, easy realization, automatic or controlled zooming and tracking, and color sensing. The use of semiconductor photosensitive arrays facilitates a direct digital input with parallel

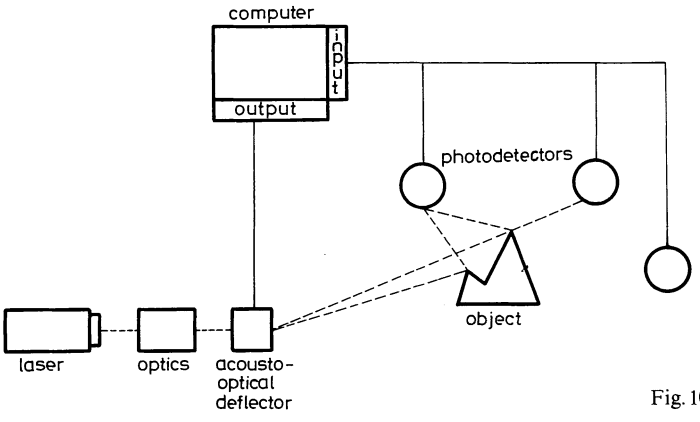


Fig. 10.2. Laser deflector input

processing possibilities. Future developments based on prospective special devices should most probably be a very feasible solution for the combined problem of input, picture processing and feature extraction.

Recognition of industrial objects needs mostly the detection of edges. The most important and characteristic edges are the contour lines. An obvious way of contour-line detection is realized by a programmable *laser-deflector*. The system uses the scanning electron microscope principle. The laser beam controlled by an

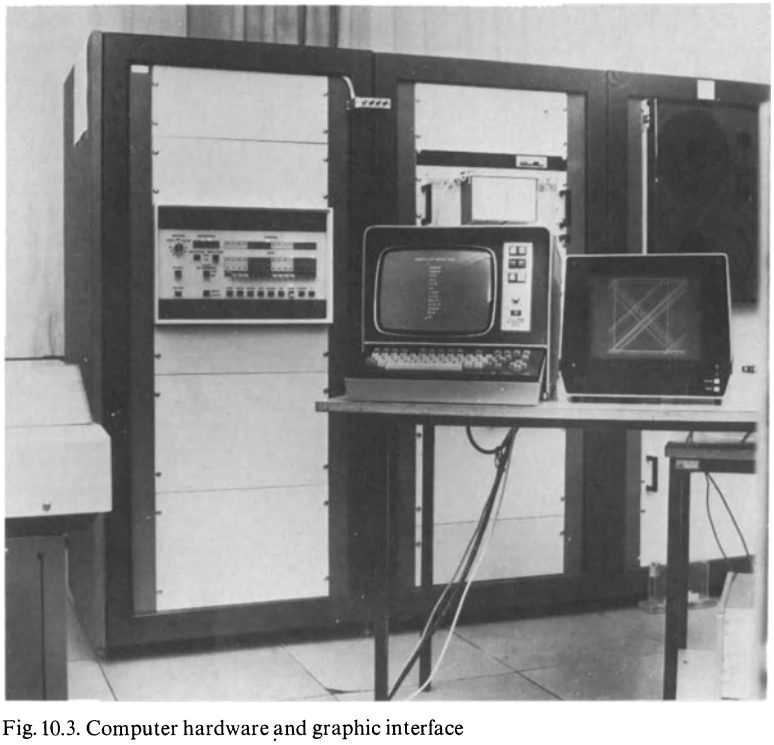


Fig. 10.3. Computer hardware and graphic interface

acousto-optical deflector with a resolution of 1000 points in each dimension is detected by photosensors mounted in different directions (Fig. 10.2). If an edge is left by the beam, at least one of the detectors indicates a sharp change in the output signal.

Laser beam-assisted edge detection is superior to the TV camera by

— the automatic selection of the contour lines from the different grey-level complete pictures;
— the providing of the address of the contour point directly through the discrete deflector positioning signal;
— less noise.

The laser-beam method can be extended by distance measurements enabling the transformation of the two-dimensional picture into a three-dimensional one.

As previously emphasized, the system is based on a man-machine interaction. This concerns the teaching and the operating phase, the various steps of recognition included. The dialog is helped by a simple *graphic display* which can visualize the patterns and pattern elements processed by the computer (Fig. 10.3). The minicomputer used is a typical 16 bit one with a 1 μs cycle time, microprogrammable, the core memory 16 K configuration, one minidisc and two tapes as background.

10.3.2 Output

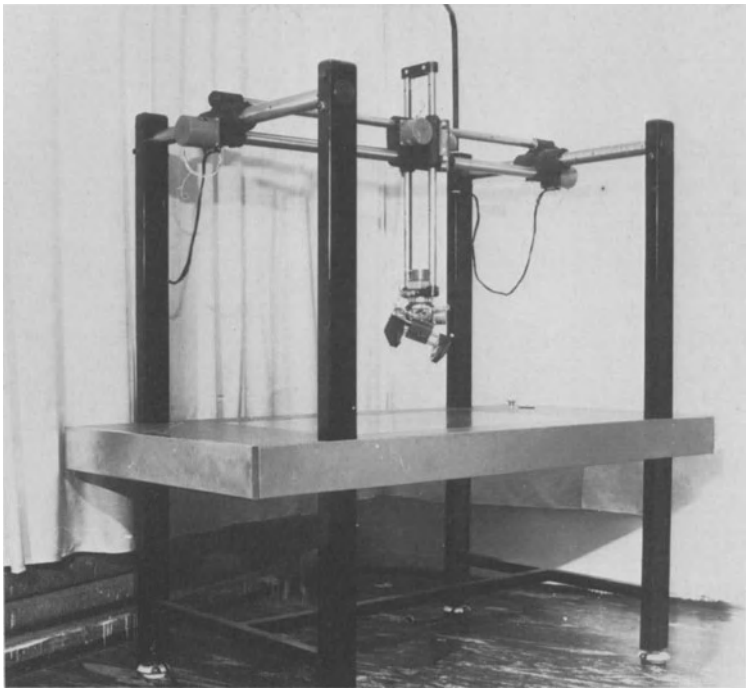
The pattern recognition system controls as its output the activity of robot hand (Fig. 10.4). This has an arm with 6 degrees of freedom plus a gripper operation. Although the visual information is primary, tactile and force sensors are also built in the hand. All degrees of freedom have individual stepping motor servos, controlled by digital interpolations similar to the machine-tool controllers used for direct numerical control. The computer issues position commands for this control unit and receives position feedback.

10.4 Software

The hardware is only an experimental tool for the software development. A special operating system (Fig. 10.5) has the duty to control individual program elements and overlay facilities (which are extremely essential in handling several very different but deeply connected tasks within a minicomputer). The system supervisor controls also the presence of the actual knowledge (programs and data) from the total information amount, collected during the higher levels of activity (learning, system design) according to Section 10.2. The design applies the ideas of WINOGRAD's project [10.45] by using a library for programs and a dictionary for semantic information. Dialog and input/output control are natural ingredients.

10.4.1 The Dictionary

Figure 10.6 contains all kinds of information about the objects. This relates to the name, object class, related objects (as for example a washer for a screw), related



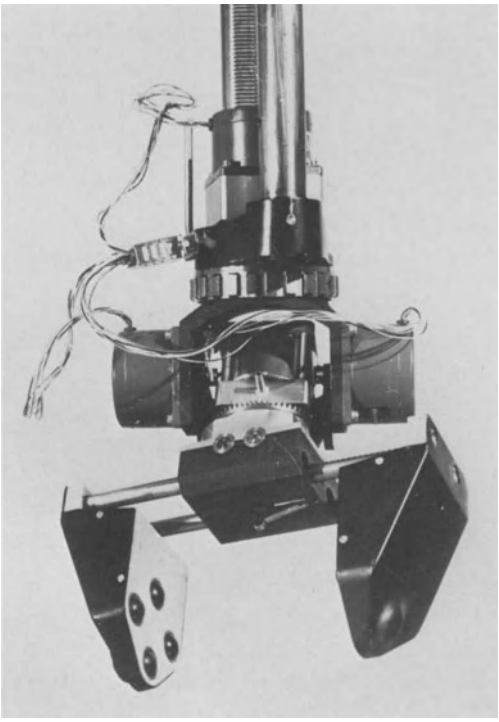


Fig. 10.4. (a) Robot hardware, (b) Hand

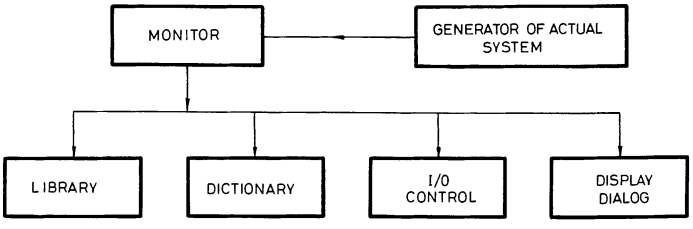


Fig. 10.5. Software system

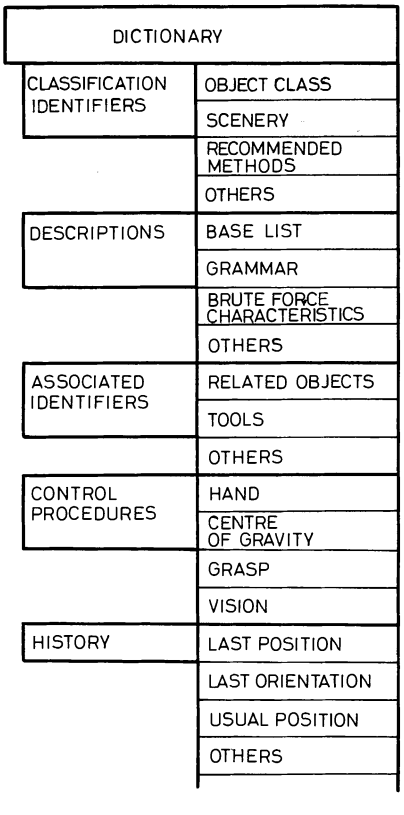


Fig. 10.6. The dictionary

tools (e.g., a key-wrench), advised ways of recognition, all kinds of descriptions needed (i.e., different methods, different views), dimensions, references for handling (e.g., position and location for grasping, force needed or allowed, etc.), history (last operation, location, orientation) and other useful information (e.g., instructions for the local checking of some possibly faulty detail, etc.).

The dictionary—according to the aforesaid—has two editions, a complete one and an abbreviation for the actual system.

The *library* is composed for the most part of different pattern recognition algorithms and therefore it will be treated in detail later.

10.4.2 Software Tools

The selection of the appropriate software tools is more imperative than for any other computer application. The four levels of activity suppose different kinds of people interacting with the system, all having a very different background in computer programming. The system is a living one, open for continuous extension, modification, not only of the activity, but also of hardware included with the computer. The system building should suppose the cooperation of design people having barely more communication than the program documentation. Economizing the computer—as a basic condition for realistic applications—mostly excludes the use of general high level languages. Consequently, as an intermediate solution, a well documentable, easily portable and extendable system-design language should be applied, most obviously some kind of *macroprocessors*. The operator of the actual system is supposed to be an employee of lower education. A very simple, strictly controlled, easily understandable *command language* should serve his dialog, mostly guided by a menu. A natural language understanding system is out of the question because of economy and reliability reasons.

Most important is the application of an economical and flexible *list processor*. The whole system—handling of picture element strings, grammatical rules, objects in a scenery with special context, relations between picture parts, attributes, program segments for grammatical inference, predicate calculus and stochastic or fuzzy logic based on decision-making and dialog handling—needs list manipulations. LISP as the classical tool for artificial intelligence is not economical enough and has several provisions which are not essentially needed for the special pur-

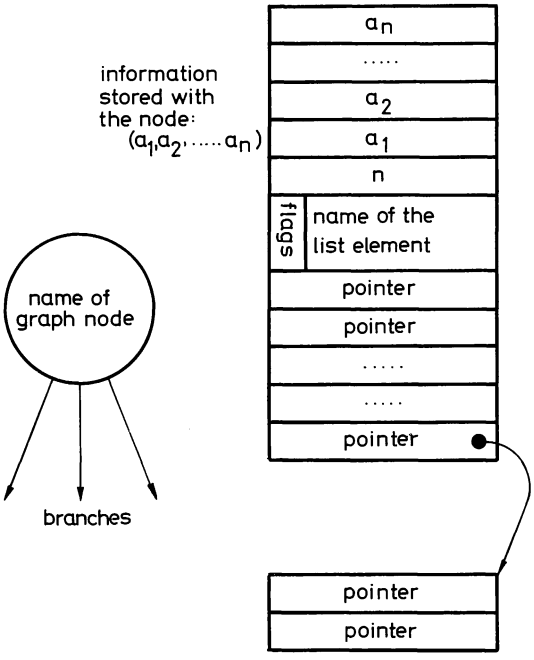


Fig. 10.7. Node structure of list processing

pose. The language used, called LIDI [10.61], has the following essential characteristics:

— a direct representative of arbitrary directed graph;
— a free assignment of listed attributes (values without pointers) and pointers is permitted;
— the language (originally written in FORTRAN) can be used by programs written in any other code;
— a simple input format, economical internal format-conversion, garbage collection.

Figure 10.7 shows the structure of a node. The head of the node located between attributes and pointers contains the name, flags indicating the presence of sequential attributes, pointers, termination or continuation. The first element of the sequential list indicates the number of the sequential elements, the last pointer (if the continuation flag is on), the address of the continuation. This procedure results in a very flexible change in list content and structure.

10.5 Pattern Recognition

The procedure is not different from the scheme of [10.24].

10.5.1 Preprocessing

The format of the input is in 8×8 points blocks, each bit representing the two possible levels of the point (edge, non-edge). A slight difference exists in handling picture points, where the whole surface within a contour is dark or the signal change directly assigns the edges. The use of different, adjustable levels of TV pictures provides a result somehow similar to the use of different coded grey-levels, a possibility for distinction of interior edges within the total contour. The meaning of this different-level picture is evaluated separately and combined only at higher level analysis. In contrast to the human intelligence, the machine system described handles separately the different types of usual information obtained one by one (different levels, views, etc.). The concurrence of these occurs in the dictionary.

Elementary line segments are detected from this small window as later parts of straight lines or arcs. A modified template method helps a fast computing, effec-

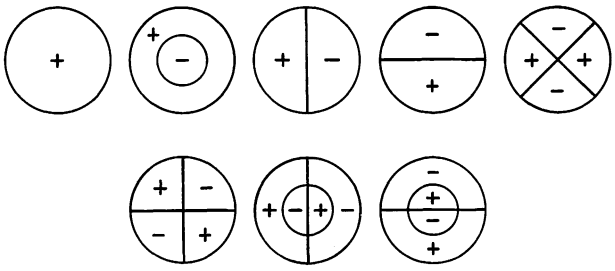


Fig. 10.8. Hueckel's templates

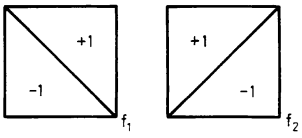


Fig. 10.9. Simplified templates

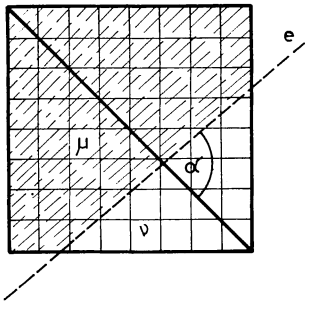


Fig. 10.10. Template matching

tive filtering, supposing a picture not completely covered by noise. Much work has been done in this field referred to in Section 10.1. The chosen way is a simplified version of HUECKEL's [10.62]. HUECKEL uses 8 different circular templates which yield the basic frequency components of an infinite Fourier series convolution comparison (Fig. 10.8). The application of only two rectangular templates (Fig. 10.9) can determine the direction and the location of the line element, separating two different grey-levels of the window.

If we allow only two values in $\underline{f}_1$ and $\underline{f}_2$, viz., 1 or -1, it can easily be seen that (Fig. 10.10)

$$\frac{\int f f_1}{\int f f_2} = \operatorname{tg} \alpha \tag{10.1}$$

independent of the values μ and ν of $\underline{f}$ on the two sides of the line $\underline{e}$. The ratio of the dark and bright points and a little logical consideration gives the location of this line.

The whole program takes about 500 machine code instructions without any cycles, floating point operations, i.e., one window can be processed on the minicomputer cited within 0.5–0.8 ms [10.63]. The method is very reliable. Supposing a noise with a binomial distribution having 5 errors in the 64 points square, the

9	9	9	9	8	7	6	5
9	9	6	6	7	7	5	4
9	9	8	7	7	6	4	3
9	9	8	7	7	6	4	3
9	9	8	7	7	6	4	3
8	8	7	7	9	5	4	2
8	8	6	8	8	5	3	2
8	8	7	7	6	5	3	2

a.)

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

b.)

1	1	1	0	0	1	1	0
1	0	1	1	1	1	0	1
1	1	1	0	1	0	1	1
0	1	0	1	1	0	1	0
1	0	1	0	0	0	0	0
0	1	0	0	1	1	0	1
1	0	0	1	0	0	0	0
0	1	0	0	0	1	1	0

c.)

Fig. 10.11a–c. Results compared with Hueckel's

probability of detecting the edge with an error not greater than 1 point anywhere is over 95%.

Figure 10.11 (a) shows the original pattern published by HUECKEL and completed in the corners for having rectangular shape. The edge computed by the method above is shown by a continuous line, the original HUECKEL result by a dotted one. Figures 10.11 (b–c) indicate two-level pictures, a noiseless one and another with more than 30% noise.

10.5.2 Feature Extraction

Line elements are evaluated to form the picture primitives. In most cases this requires the additional processing of overlapping windows to detect the exact location of direction or curvature changes. The *feature extraction* of primitives is a very simple examination of elementary-line slopes and thereafter the second order differences result in determination of lines and arcs separated one by one by nodes (nodes understood also as concatenation points). These calculations provide also the attributes of the primitives (Fig. 10.12), as the coordinates of the points A, B, C, K.

In this way the primitives are classified in four types: the quoted straight lines, arcs, nodes, further undefined types being uncertain entitites.

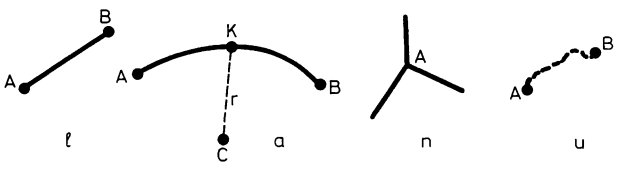


Fig. 10.12. Picture primitives

10.5.3 The Grammar

A context-free grammar is used to build up the machine parts from the picture primitives, extracted by the briefly described FEX (feature extraction) routines. A 7-tuple is defined by

$$G=(V_T, V_N, S, \varrho, \sigma, \tau, P), \text{ where} \tag{10.2}$$

$V_T=\{l, a, n, u\}$ terminal symbols

$V_N=\{\text{object, lobe, GPP}\}$ nonterminals

$S=$ object name

$\varrho=$ relations between two elements of $V_T \cup V_N$

$\sigma=$ attributes assigned to the elements $V_T \cup V_N$

$\tau=$ relevance grades assigned to V_N elements

$P=$ production rules.

The terminal symbols are the results of the feature extraction procedure described above.

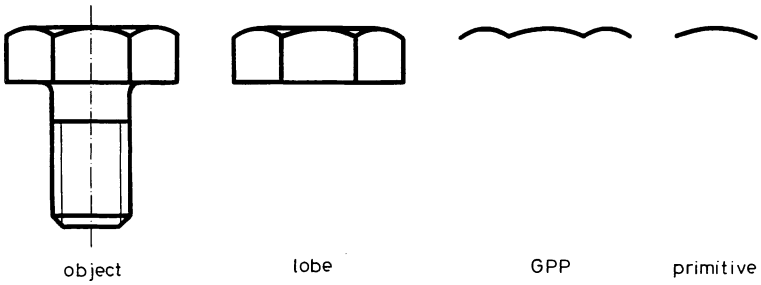


Fig. 10.13. Picture hierarchy

The nonterminals are the different levels of the object description hierarchy (Fig. 10.13). The GPPs (Generalized Picture Primitives) are composed of 2–5 primitives, somehow related and characteristic ones. These are fixed for a given class of patterns in the teaching period (Fig. 10.14). The selection of GPPs can be made automatically by a simple analysis of the adjacency occurrence but as to the practical usefulness, the human selection can be faster and more effective. The easy way to do this is given by the display dialog during the teaching phase. The GPPs also have relations between each other and attributes for qualitative and quantitative description details.

The next higher level is nicknamed *lobes*. The idea is borrowed from Hegel, who jokes on essential differences, arguing that the essential difference between man and animal should be the lobe, because to animal has a lobe. Lobes are composite GPPs, decisively characteristic for a special object such as a sequence of cogs for a gear or a set of balls for a ball bearing. The lobes can similarly be defined as the GPPs.

Relations ϱ are characteristics between the symbols. An early work of SHAW [10.64] established only four types of relations, describing the different modes of concatenation. EVANS [10.65] suggested a free choice. In this application—at

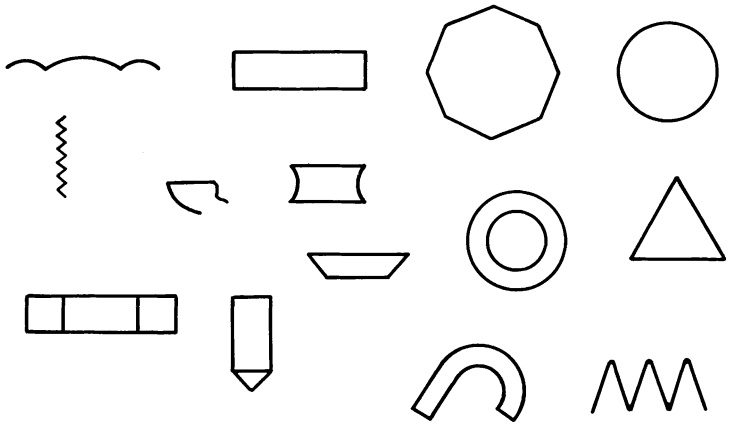


Fig. 10.14. Generalized picture primitives

$V_T = \{arc\}$

$V_N = \{ARC, ARC\ 2, LOOP, LA\}$

$\varrho = \{=, \neq, |\varphi_1 - \varphi_2| > 90°\}$

$\sigma = \{A, B, C, K, \varphi_1\}$

P :

0 / ARC $\leftarrow$ |(arc)( )(A=A_1, B=B_1, φ= ARCTG (A_1,B_1), K=K_1, C=C_1)(A,B)|

1 / ARC $\leftarrow$ |(ARC, ARC)(A_1=A_2, $B_1 \neq B_2$, $K_1 \neq K_2$, $|\varphi_1 - \varphi_2| > 90°$) (A=$B_1$, B=$B_2$, K=$K_1$, L=$K_2$, C=$A_1$)|

2 / LOOP $\leftarrow$ | (ARC 2, ARC)(A_1=A_2, B_1=B_2, $K_1 \neq K_2$, $L_1 \neq K_2$) (A=A_1, B=B_1, C=C_1)(A, B)|

3 / LA $\leftarrow$ |(LOOP, ARC)(A_1=A_2, $B_1 \neq B_2$, $C_1 \neq B_2$)|

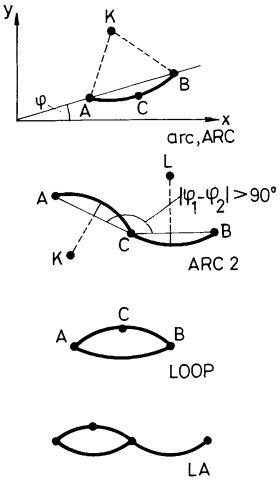


Fig. 10.15. Picture grammar

least for level 1 and mostly for 2 also of Section 10.2—the program should possess the ability to define new relations as it is needed by the situation. Relations more frequently applied are: concatenation, loops, parallelism, left, right, below, over, inside, etc. The relations are also identifiers of subroutines calculating the true values of the definitions on the basis of the variables given by the attributes (Fig. 10.15).

The *attributes* σ are mostly numerical values, e.g., slopes, end-point coordinates, radii, distances, etc., listed in a preselected, strict order with respect to the node of the grammar graph representing the symbol.

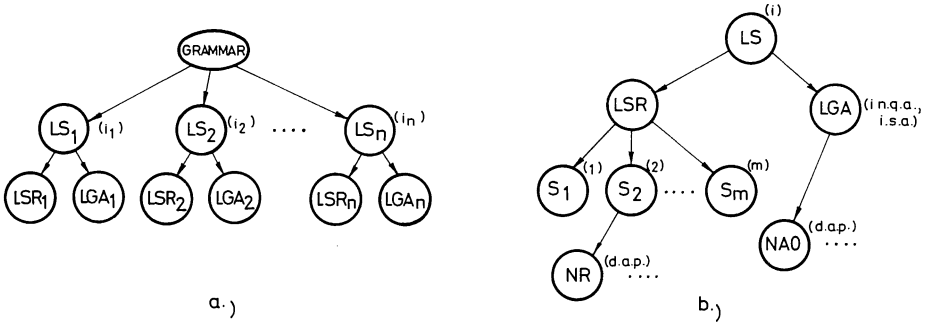
The *relevance grades* τ will be discussed later in Subsection 10.5.5. At this point it should only be mentioned that these are coefficients assigned to the branchings of the graphs, indicating the recommended shortest paths on the search graph.

10.5.4 The Organization of the Grammar

The grammar represents a permanent data structure for a given scene [10.66]. The rules are divided into two groups:

— containing only terminal symbols on the right side (e.g., rule O, Fig. 10.15)
— containing only nonterminals on the right side (e.g., rules 1–3, Fig. 10.15).

According to the Chomsky-normal form this can always be done and simplifies the later processing. First class rules serve as the preprocessors of the list obtained by the feature extraction, a list composed only by primitives, their attributes and concatenation sequence. The list of these rules will be the common basis of all grammars to be treated. Thus these rules are actually not included in the grammar list. The latter are the rules of the second group, structured as Fig. 10.16.a. Each nonterminal left-side symbol points towards two lists, one com-



LS - Left-side Symbol
i - number of the rule
LSR - List of right-side Symbols and Relations
LGA - List for Generation of the quantitative Attributes of the left-side symbol
S - Symbol
NR - Name of geometrical Relation
NAO - Name of Assignment Operation
d.a.p. - description of actual parameters
i.n.q.a. - indication of number of quantitative attributes
i.s.a. - indication of symmetrical quantitative attributes

Fig. 10.16a and b. Structure of the grammar

posed of right-side nonterminals and relations between them, the other to the generation of the quantitative attributes of the left-side symbol. As the left-side symbol is composed of the right-side ones, the quantitative attributes of the left-side symbol should be calculated from the components according to operations listed here. Figure 10.16.b shows a detailed part and Fig. 10.17 the realization for rule 1 of Fig. 10.15.

Accordingly the grammar works not only as usually like a parsing procedure, but also controls a series of subroutines calculating relations and assigning attributes. This necessitates two additional lists which are only referred to in the grammar: one for the names of the geometrical relations (Fig. 10.18.a) and one for the generated quantitative attribute sets (Fig. 10.18.b).

The grammar is a programmed one. The numbers of the rules control the procedure which indicates the hierarchy of the nonterminals. The calculation of

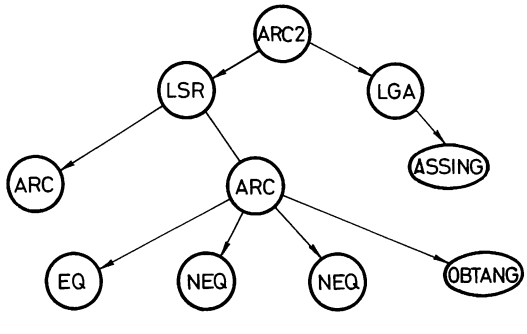
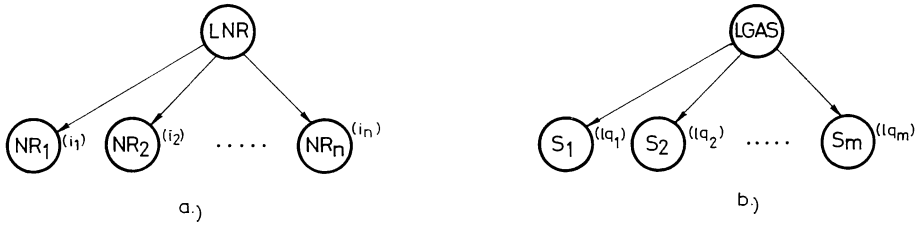


Fig. 10.17. Structure of a rule



LNR – List of Names of Geometrical Relations
NR – Name of geometrical Relation
i – switching index

LGAS – List of Generated quantitative Attribute Sets
s – Symbol (Syntactical type of picture element)
lq – List of quantitative attributes of elements with syntactical type S for actual input picture

Fig. 10.18a and b. Complementary lists

attributes as semantic information is conducted also this way. The data structure and the algorithms are more simple than the usual way; the procedure enables an interactive system (Subsect. 10.5.5) to provide all syntactic (structural) and semantic (pictorial) information at every parallel step. The semantic information is created in an associative list which joins the grammar.

There is a slight difference from the usual parsing algorithms in the inputs, too. It is not a simple string but a list which contains some topology. The essential information of the topology is transferred to the semantics. The whole analysis can be considered as a syntax controlled semantic one.

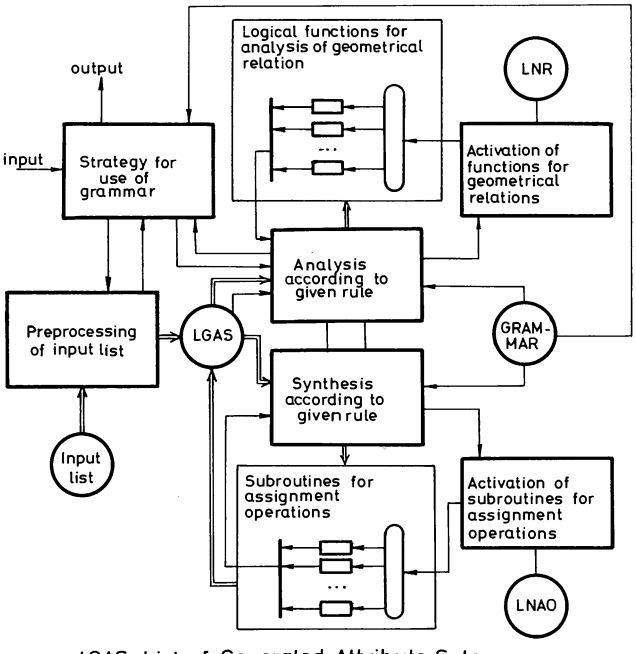
In teaching the system the *grammars*, the *relations* and *assignment operations* for attributes are set as semipermanent data structures. These are called semipermanent because they are usually unchanged during the actual recognition operation (level 4, Sect. 10.2), but can be extended or modified always by operator assistance.

The grammatical analysis is driven by a set of subroutines:

— analysis according to a given rule
— synthesis according to a given rule
— organization of the relation-checking routines
— organization of the assignment routines
— grammatical analysis strategy (determining the sequence of the rules to be treated)
— other service-routines.

These are the standard backbone of the system. The analysis works with synthesizing attempts. Progressing bottom-up and taking one by one the first right-side symbols, the analysis tries to compose the left-side ones, which are mostly of the higher syntactic level. In a well-organized sequence the number of back trackings is minimized. The relations and attributes are checked after each success in the analysis; the attributes of the left-side more complex symbols are calculated after the rules of the right-side.

The whole procedure scheme is indicated in Fig. 10.19, circles denote lists, boxes program-segments, double lines denote information transfer and single ones control.



LGAS-List of Generated Attribute Sets

LNR -List of Names of geometrical Relations

LNAO-List of Names of Assignment Operations

Fig. 10.19. Block scheme of the syntactic analysis

10.5.5 Some Heuristics

The lobes by definition are highly characteristic picture elements. It would be plausible that the recognition should be concentrated on only looking at the lobe if a structure is somehow established, or looking only for the relevant lobe if the contour is listed before. This procedure could save the most tedious work of grammatical analysis, search through a complex tree and structures filled mostly be irrelevant elements. This obvious way of human recognition is not directly viable for two reasons: one is the inherent difference in parallel and serial methods, the other is the uncertainty of lobe identification because of noise and different view conditions. Some considerations can help to reach an intermediate result. During the teaching period the objects are analyzed several times, under different circumstances. A measure of relevance of some detail can be obtained in the following way [10.58]:

Let us denote the GPPs by $l_1, l_2, \dots l_n$ and the lobes composed by them by $L_1 \dots L_N$.

Consequently:

$$L_i = l_{i1}, \; l_{i2}, \dots l_{iJ}, \quad \text{where} \quad J \leqq n \tag{10.3}$$

We have m different objects: $O_1, O_2, \dots O_m$.

The $\underline{k}$th object has been optically processed and the picture analyzed during the learning period $N(O_k)$ times. The probability of occurrence of the O_k object among the whole set during the learning is

$$p(O_k) \sim \frac{N(O_k)}{\sum_{l=1}^{m} N(O_l)} (k=1, \dots m) . \tag{10.4}$$

The conditional probability of the case in which the O_k object contains the L_i lobe

$$p(L_i|O_k) \sim \frac{N(L_i|O_k)}{N(O_k)} . \tag{10.5}$$

$N(L_i|O_k)$ is the number of observations, where the lobe L_i belongs to the object O_k. For these it is valid that

$$\sum_{i=1}^{N} p(L_i|O_k) \sim \frac{\sum_{i=1}^{N} N(L_i|O_k)}{N(O_k)} = 1 . \tag{10.6}$$

The conditional probability of the case that a recognized L_i lobe identifies the O_k object can be obtained from the Bayers-formula

$$p(O_k|L_i) = \frac{p(O_k)\, p(L_i|O_k)}{\sum_{j=1}^{m} p(O_j)\, p(L_i|O_j)} . \tag{10.7}$$

This conditional distribution is the result of the learning phase. On the basis of the table of conditional distribution for lobes and objects in the recognition phase, we can find only the most probable lobes and have a short-cut through their identification.

The recognition phase collects the GPPs of the viewed object on the basis of the extracted primitives and the grammatical analysis using the GPPs defined in the problem. This algorithm is the same during both learning and recognition. The result is a list of GPPs found in the given picture. This list is the *measured lobe* $\underline{L}_M$. The $\underline{L}_M$ is to be matched to the list of possible $\underline{L}_i$s obtained during the learning. A practical measure of distance is defined by the difference $d(L_M, L_i)$ in the lobe lists. In our experiment $d(L_M, L_i) \leqq 3$, e.g., only those lobes are taken into consideration which differ maximum by three GPPs from the measured $\underline{L}_M$. By the limitation of the distance the comparison algorithm will be very fast.

For every lobe a grade of recognition can be defined by means of the distances.

$$q(L_i, L_M) = \begin{cases} \dfrac{[1+d(L_M, L_i)]^{-1}}{\sum_{j=1}^{r} [1+d(L_M, L_j)]^{-1}} & \text{if} \quad \underline{d}(L_M, L_j) \leqq 3 \\ 0 & \text{if} \quad \underline{d}(L_M, L_j) > 3 \end{cases} \tag{10.8}$$

r is the number of lobes having a distance $\leqq 3$.

This definition is an arbitrary one. It is easy to compute, it has no singularities and divides fairly sharply the probability values. The distribution is obviously regular.

By these $q(L_i, L_m) - s$ and $p(O_k|L_i) - s$

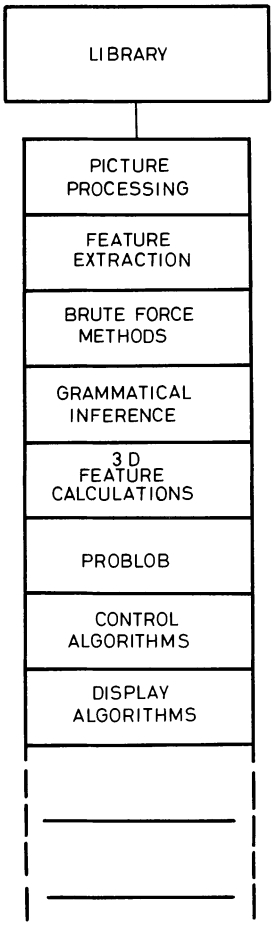


Fig. 10.20. The library

calculated in the learning phase, the grade of the object being $\underline{O}_k$ is

$$Q(O_k, L_M) = \sum_{l=1}^{N} q(L_l, L_M)\, p(O_k | L_l) \tag{10.9}$$

$$\sum_{k=1}^{m} Q(O_k, L_M) = 1\,. \tag{10.10}$$

The $Q(O_k, L_M)$ values are ordered by magnitude; this list is applied as the τ numbers of the grammar (Subsect. 10.5.3).

This yields some preferred routes in the analysis process accelerating it several times. The algorithm is listed in the library under the name PROBLOB. Another important short-cut can be the use of geometric relations and attributes between the lobes. If an object is identified by a brute-force method, the structure can be quoted from the dictionary as the grammatical description of the object. Neglecting all other details, the lobe as most important for final identification or handling can be selected by saving the regular analysis work.

As it can be seen, the library (Fig. 10.20) which contains fairly different but—within the dictionary data base—concurring methods and programs creates a type of procedural grammar in close relation to the *dictionary*. The graphic display *dialog* by the possible visualization of all intermediate steps of pattern recognition (the string of the extracted primitives, lobes, GPPs, brute-force chracteristics, stored models) and essential data of the objects (e.g., center of gravity, surface direction and point of contact for grasping, last location, orientation, etc.) is an integrated part of the system both in teaching (levels 2 and 3) and during actual work under operator supervision (level 4). Machine heuristics and man-machine interaction are not alternatives, but ideal complements.

10.6 Conclusions

The primacy of the objective has been emphasized several times in this chapter. Recognition of industrial objects, especially machine parts, is aimed at helping their manipulation. An intrinsic relation between the method of recognition and control of manipulation suggests itself. Grammatical recognition method is chosen also because of the practical application of the description obtained for solving robot control.

The grammatical method is the only reasonable way for the problem definition, looking for an object, or mostly for two, envisaging a definite operation and establishing a new relation between them considering their recent circumstances. The decision-tree is straightforward through the recognition methods outlined in the preceding chapters, definition of exact location, orientation and thereafter looking for the best grasping surfaces and directions depending on

a) the initial location and orientation of the object and the neighboring ones,

b) stability conditions during transfer,

c) final or intermediate location and orientation of the object considering also the possible route of reaching this situation, and initial conditions of the next operation (e.g., fixing, fitting, etc.).

All these are done on the basis of a very detailed structural description of the whole scenery concerned, inclusive of the possibility of looking only for those details which are relevant for the forthcoming decision.

The command language as a high level communication tool between operator and robot has a lowest level, a resolution of the complex-content bearing, simple looking command (e.g., put, or transfer) to really simple movement instructions. These latter are the machine-code translation of point-to-point trajectory instructions understandable by a special numerical control tool, containing already no information on the context. Between these two extreme levels an intermediate one is needed, which controls the computation from the general task towards the special movement detail. This level considers the issues enumerated under a)–c) above and acts consequently over the grammars of the recognition phase. The linkage of these descriptions is realized also in the semantic dictionary.

Control of adaptive manipulators (more common robots) proved to be an excellent example of the intellectual power of the grammatical methods. Recognition, decision making and mechanical control are semantically linked to perform a rather consistent system, one of the most ambitious enterprises of the human intellectual. Within the past 15 years this goal was of course not reached on full industrial scale and in the full realization of the objective, but a lot of partial practical applications already exist and several brilliant and future-building ideas are well outlined. An encouraging number of progressive experiments have far exceeded the first attempts and natural failures.

The grammatical method has been the most important theoretical contribution to all of them—without disregarding other aids and tools. This is only too obvious as the human language is the highest achievement of the human intellectual progress, the natural means to define and control all our human activities.

Acknowledgement. The author is indebted to express his thanks to the members of his research group especially to Mrs. GALLÓ, Mr. VASSY and Mr. MÉRÖ for their contribution to the common project and helpful remarks and Mrs. HETÉNYI for the preparation of the manuscript.

References

10.1 P. H. WINSTON: *Machine Intelligence* **7**, 431–463 (1972)
10.2 J. MCCARTHY: MEMO AIM-228, Stanford Artificial Intelligence Laboratory (1973)
10.3 J. A. FELDMAN, G. M. FELDMAN, G. FALK, G. GRAPE, J. PEARLMAN, I. SOBEL, I. M. TENENBAUM: in *Proc. of the Intern. Joint Conf. on Artificial Intelligence* (Washington, D.C. 1969), pp. 521–526
10.4 P. E. HART, R. E. FIKES, J. D. GARVEY, W. I. NILSSON, D. NITZAN, I. M. TENENBAUM, B. M. WILBER: Stanford Research Institute Rept. (1972)
10.5 H. G. BARROW, G. F. CRAWFORD: *Machine Intelligence* **7**, 465–480 (1972)
10.6 M. JOUTAKI, K. SATO, T. NAGATA: Japan Electronic Engineering **51**, 24–29, 76 (1971)
10.7 Y. SHIRAI: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence*, Tbilisi-USSR (1975), ed. by The Artificial Intelligence Lab., Cambridge, MA, pp. 674–687
10.8 M. EJIRI, T. UNO, H. YODA, T. GOTO: in *Proc. of the 2nd Intern. Joint Conf. on Artificial Intelligence* (London, 1971), pp. 350–358
10.9 M. YACHIDA, S. TSUJI: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence*, Tbilisi-USSR (1975), ed. by The Artificial Intelligence Lab., Cambridge, MA, pp. 819–826
10.10 S. TSUJI, R. FUJIWARA: in *Proc. of the 2nd Joint Conf. on Pattern Recognition* (Copenhagen, 1974), pp. 104–108

10.11 R.T. CHIEN, Y.H. CHANG: in *Proc. of the 2nd Joint Conf. on Pattern Recognition* (Copenhagen, 1974), pp.496–510
10.12 R.T. CHIEN, W. SNYDER: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by The Artificial Intelligence Laboratory, Cambridge, MA, pp. 742–748
10.13 L.S. COLES, A.M. ROBB, P.L. SINCLAIR, M.H. SMITH, R.R. SOBEK: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by the Artificial Intelligence, Laboratory, Cambridge, MA, pp. 749–757
10.14 Report of the Workshop on Advanced Automation. Purdue University, West Lafayette, Indiana, 1975
10.15 D.E. OKHOTSIMSKY, A.K. PLATONOV: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by The Artificial Intelligence Laboratory, Cambridge, MA, pp.798–802
10.16 V.I. RIBAK, G.L. HIMMELFARB, E.F. KUSHNER, A.I. VOLDIREV, A.N. VASIKOV, V.V. GANDOUK, V.M. KROT, I.N. KROUPKA, V.S. LANBIN: in *Problem of the Automata Theory, Robots and Computers,* ed. by the Ukrainian Ac. of Scs., Inst. of Cybernetics, Kiev, 1975 pp.3–11 (in Russian)
10.17 F. RÖCKER, A. KIESSLING: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by The Artificial Intelligence Laboratory, Cambridge, MA, pp. 669–673
10.18 G. GINI, M. GINI: *Aplikovaná Robotika* **2**, pp. VIII/1 (1975)
10.19 T. VÁMOS, Z. VASSY: in *Proc. of the âth World Congress of IFAC,* Boston, MA (1975), Part IV/D, 63.1
10.20 L.G. ROBERTS: Tech. Rept. No. 315, MIT Lincoln Lab., Lexington MA (1963)
10.21 A. GUZMÁN: MAC Tech. Rept. No. 59, MIT, Cambridge, MA (1968)
10.22 A. GUZMÁN: *Machine Intelligence* **6**, 325–376 (1971)
10.23 O. DUDA, P. HART: *Pattern classification and scene analysis* (John Wiley and Sons, New York, London, Sydney, Toronto 1973)
10.24 K.S. FU: *Syntactic methods in pattern recognition* (Academic Press, New York, London 1974)
10.25 A. ROSENFELD: *Picture processing by computer* (Academic Press, New York 1969)
10.26 G.G. VAINSTEIN, N.V. ZAVALISHIN, I.B. MUCHNIC: Automation and Remote Control **6**, 99–132 (1974) (in Russian)
10.27 L. KANAL: IEEE Trans. **IT-20**, 697–722 (1974)
10.28 A. ROSENFELD: Computing Surveys **5**, 81–108 (1973)
10.29 A. ROSENFELD: Computer Graphics and Image Processing **1**, 394–416 (1972)
10.30 A. ROSENFELD: Computer Graphics and Image Processing **3**, 178–194 (1974)
10.31 A. ROSENFELD: Computer Graphics and Image Processing **4**, 133–135 (1975)
10.32 D.A. HUFFMAN: Decision criteria for a class of "impossible objects", in *Proc. of the 1st Hawaii Intern. Conf. on System Sciences* (Honolulu 1968)
10.33 M.G. CLOVES: Artificial Intelligence **2**, 79–116 (1971)
10.34 R. ORBAN: AIM-192, ed. by The Artificial Intelligence Lab., Cambridge, MA (1970)
10.35 D.G. WALTZ: AI TR-271, ed. by the Artificial Intelligence Lab., Cambridge, MA (1972)
10.36 G. FALK: Ph. D. Thesis, ed. by the Computer Sc. Dept., Stanford Univ. (1970), CS-180, AIM-132
10.37 G. AGIN: Ph. D. Thesis, ed. by the Electrical Engg. Dept., Stanford Univ., AIM-173 (1972)
10.38 Y. YAKIMOVSKY, I.A. FELDMAN: A semantic-based decision theoretic region analyzer; in *Proc. of the 3rd Intern. Conf. on Artificial Intelligence* (Stanford, CA 1973)
10.39 S. TSUJI: The ETL-Robot; Densoken Nyusu (1970), No. 251 Denski gijutsu sogo kenkyujo, Tokio
10.40 C.R. BRICE, C.L. FENNEMA: Artificial Intelligence **1**, 205–226 (1970)
10.41 T.G. EVANS: A formalism for the description of complex objects and its implementation; in *Proc. of the 5th Intern. Congress on Cybernetics* (Namur-Belgium 1967)
10.42 H.G. BARROW, I.R. POPPLESTONE: *Machine Intelligence* **6**, pp. 377–396 (1971)
10.43 M. MINSKY, S. PAPERT: AIM-252, ed. by The Artificial Intelligence Lab., Cambridge, MA (1972)
10.44 C. HEWITT, C. PLANNER: AIM-137, ed. by The Artificial Intelligence Lab., Cambridge, MA (1967)
10.45 T. WINOGRAD: AI-TR-235, ed. by the Artificial Intelligence Lab., Cambridge, MA (1971)
10.46 R.C. SCHANK: AIM-76, ed. by Stanford AI Lab. (1968)

10.47 Y. WILKS: AIM-206, CS-377, ed. by Stanford AI Lab. (1973)
10.48 R. C. SCHANK, Y. WILKS: AIM-202, CS-368, ed. by Stanford AI Lab. (1973)
10.49 I. M. TENENBAUM, T. D. GARVEN, S. A. WEYL, H. C. WOLF: ISI—An interactive facility for scene analysis research; in *Proc. of the 3rd Intern. Conf. on Artificial Intelligence* (Stanford, CA 1973)
10.50 I. P. GOLDSTEIN: Artificial Intelligence **6**, 249–288 (1975)
10.51 S. TSUJI, A. NAKAMURA: in *Advance Papers of the* **4** *th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by the Artificial Intelligence Lab., Cambridge, MA, pp. 811–818
10.52 K. S. FU, P. H. SWAIN: On syntactic pattern recognition; in *Software Engineering*, Vol. 2 (Academic Press, New York 1971)
10.53 K. S. FU: Syntactic pattern recognition and stochastic languages; in *Frontiers of pattern recognition* (Academic Press, New York 1972)
10.54 P. H. SWAIN, K. S. FU: Pattern Recognition **4**, 83–100 (1972)
10.55 H. C. LEE, K. S. FU: TR-EE 72-17, ed. by the School of Electr. Engg., Purdue Univ., West Lafayette, IN (1972)
10.56 L. A. ZADEH: Information and Control **8**, 338 (1965)
10.57 L. A. ZADEH: IEEE Trans. **SMC-3**, 28–44 (1973)
10.58 T. VÁMOS, Z. VASSY: in *Proc. of the 1st Intern. Joint Conf. on Pattern Recognition* (Washington 1973), pp. 445
10.59 P. E. MART: Final Report, SRI Project 7642, ed. by the Stanford Research Inst., Menlo Park, CA (1969)
10.60 I. SOBEL: AIM-121, ed. by the Computer Science Dept., Stanford Univ. (1970)
10.61 M. UZSOKY, I. FIDRICH: Rept. No. 16, ed. by the Computer and Automation Inst. of the Hung. Acad. of Scs., Budapest (1973)
10.62 M. HUECKEL: AIM-105, ed. by the Stanford Research Inst., Menlo Park, CA (1969)
10.63 L. MÉRÖ, Z. VASSY: in *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR (1975), ed. by the Artificial Intelligence Lab., Cambridge, MA, pp. 650–655
10.64 A. C. SHAW: Report SLAC-84, ed. by the Stanford Linear Accelerator Center, Stanford Univ. (1968)
10.65 T. G. EVANS: Grammatical inference techniques in pattern analysis; in *Software Engineering*, Vol. 2 (Academic Press, New York 1971)
10.66 V. GALLO: A program for geometrical pattern recognition based on the linguistic method of the description and analysis of geometrical structures; *Advance Papers of the 4th Intern. Joint Conf. on Artificial Intelligence,* Tbilisi-USSR, Sept. 1975, e.dby the Artificial Intelligence Lab., Cambridge, Ma., pp. 628–634

Subject Index

Accidental whorl 182
Acoustic patterns 77, 81, 89
Adjacency graph 141
Analog signal processing 57
Angular characteristic 127
Array automata 17
— grammar 17
Attribute 258
— grammar 27
Augmented description 89
Automatic speech recognition 65

b-string 127
Baseline 31, 36, 129
Bayes formula 262
Bifurcation 180
Boundary tracing 143

Central pocket loop 181
Chain code 10, 201
Character configuration two-dimensional 147, 151
— recognition 95, 148
Chicoder 103, 119
Chinese character 96, 133
— — recognition 95
— reading machine 95
Chromosomes 14, 133
Clock model 58
— pulse detector 59
Clustering 215
Communication system model 51
Coarticulation 66, 71
— vowel-consonant-vowel 66
Component adjacency graph 129
Concave arc 127
Containment code 186
Context-free grammar 4, 14, 34, 190, 256
— for Chinese character 101
— language 4, 14
— programmed grammar 14, 243
Context-sensitive grammar 4, 14
— language 4, 14
Continuous speech 70
Contour tracing 107
Convex polygon 11
— set 125
Coordinate grammar 147, 151, 156, 159
CRAWL 108
Cross-correlation 224

Data structure 144
— tablet 147
Decision-theoretic approach 1, 26
Decomposition 128
— hierarchical 126
— of complex shapes 126
— of polygon with holes 132
Derivation 4, 21
— diagram 223
— tree 5, 25
Derivative 91
Divergence 180
Double loop 182

Earth resources 215
Edge detection 245
Electrocardiogram (ECG) 31, 39, 42
— recognition 42
— rhythm interpretation 51
Electrocardiograph 42
Encoding 95, 114, 116
— digital to analog 57
English cursive script 8, 9
Error norm 37
Error-correcting parsing 27

Fast Fourier transform 67
Feature 31, 68
— distinctive 69
— extraction 31, 42, 68, 70, 90, 186, 203, 256
— phonetic 69
Fingerprint Classification 179
— Identification 182
— Pattern 179
Finite-state automaton 13, 18, 91
— (regular) grammar 4, 23
— — language 4, 13, 23
Formant cluster 79, 84
— frequency 70

Formant loci 85
— pattern 85
Frication noise description 80
Fuzzy grammar 246

Generalized chain encoding 127
— Picture Primitives (GPP) 257
Gradient method 245
Grammar 1, 3
— with coordinates 27, 147
Grammatical inference 7, 205
Graph grammar 19

Hand-eye project 243
Handwritten numeral 133, 141
Heart 51
— rhythm system of 52, 53
Henry's classification 179, 187
Hierarchical graph model 223
Hypothesis evaluation 85

Imbedding 236
Industrial object 243, 264
Insertion rule 89
Invariance
— size 144
— translation and rotation 144

Land use 215
Laser deflector 243, 249
Left recursion 158
Lexical-dependent modifier 85
Linear mean square approximation 37
LISP 253
List processor 253
Lobe 257, 261
Logic filtering 36
Loss function 88

Macroprocessor 253
Machine part 243, 264
Mair's family tree 187
Markov chain 58
Mathematical notation 147
— arithmetic 147, 150, 156
— matrix 147, 157
Medial axis transformation 125
Morphology of Chinese characters 98

N-Attaching Point Entity (NAPE) 18
Nasal spectra 84
Noisy channel 51, 53
Nonterminals 3, 35
Numeral classification 128

Pacemaker 53
Parse list 26
Parser 6
Parsing 2, 23, 26, 225, 246
— bottom-up 26, 195
— efficiency 158
— top-down 18, 26, 147, 148, 195
— tree 6
Partition 147, 148, 151—155
Pattern grammar 13
— high-dimensional 16
— primitives in terms of regions 1, 7
— segmentation 6
PDL 17
Peak 31, 32
— amplitude 31, 36, 44
— boundary 32, 33, 44
— detection 31
— duration 31, 36, 44
— extremum 32, 33, 44
— recognition 31, 36
Picture description language 1
— recognition 1
Pitch period 67
Piecewise linear approximation 32, 43
— polynomial approximation 36
Phonetic transcription 67
Phrase-structure grammar 3
Plain arch 180
Plain whorl 180
Plex grammar 18
Polygonal approximation 125, 126, 141
— figures 12, 13
Predicate 232
Preprocessing 5, 254, 223
Primary graph 13
— subset 11, 12
Primitive extraction 6
— selection 7, 14
— emphasizing boundaries 10
Principal component analysis 68
Procedural grammar 246
Production 3
— probability 197
Pseudo-syllable segments 85

Radial loop 180
Region-growing 245
Regular expression 34, 91
Relational graph 2
— matrix 3
Relevance grade 258
Rewriting rule 2, 3
Right linear grammar 92
Robot 244
— hand 250—251

Sampling matrix 184, 191
Scene analysis 1, 13
Search 108
— tree 57, 60
Segmentation 6, 38, 74, 77, 98
— grammar 74
— syntax-controlled 70
— unipolar 41
Semantic information 8, 27, 150, 246, 260
— routine 40, 41
Shape perception 125
— primitive 125
Short-time spectra 69
Signal-to-noise ratio 36
SINcoder 103, 104
Sinotype 103
Sinowriter 103, 119
Skeleton 10, 125
Spatial dependency 223
— information 217
Spectral derivative 79
Spectogram 67, 84
Speech analysis 67
— perception 65
— model for 65
— recognition system 70
— traces 68
Spiral 126, 127
Split-and-merge algorithm 141
Start symbol 3
State transition 39
Stochastic finite-state automaton (SFSA) 51, 54, 85
— grammar 27, 85, 196
— language 27, 66
Syllable nuclei 71
Syntactic (structural) approach 1, 51, 144, 183
— pattern recognition 1, 47
— system 5
Syntax analysis 2, 23, 195
— analyzer 6, 25
Syntax-directed recognition 147, 148

Tactile sensor 247
Template-matching 6, 119, 243, 255
Tented arch 180
Terminals 3, 35
Training 232
Transition probability function 54
Transversal filter 67
Tree automata 21, 23, 204
— grammar 21, 23, 198, 203
— search 55

Ulna loop 180

Vocal tract behavior 66
Vowel reduction 66

Waveform 31, 42
— baseline for 38
— bipolar 39
— first difference of 33
— unipolar 28, 40
Web grammar 18, 215
— stochastic 230

Zero crossing density 81

Communication and Cybernetics

Editors: K. S. Fu, W. D. Keidel, H. Wolter

Vol. 1 W. Meyer-Eppler

Grundlagen und Anwendungen der Informationstheorie
2. Auflage, neubearbeitet und erweitert von G. Heike und K. Löhn

Vol. 2 B. Malmberg

Structural Linguistics and Human Communication
An Introduction into the Mechanism of Language and the Methodology of Linguistics. 2nd revised edition

Vol. 3 J. L. Flanagan

Speech Analysis / Synthesis and Perception
2nd edition

Vol. 4 G. Herdan

The Advanced Theory of Language as Choice and Chance

Vol. 5 G. Hammarström

Linguistische Einheiten im Rahmen der modernen Sprachwissenschaft

Vol. 6 J. Peters

Einführung in die allgemeine Informationstheorie

Vol. 7 K. Weltner

The Measurement of Verbal Information in Psychology and Education
Translated from the German by B. M. Crook

Vol. 8 **Facts and Models in Hearing**
Proceedings of the Symposium on Psychophysical Models and Physiological Facts in Hearing, held at Tutzing, Oberbayern, Federal Republic of Germany, April 22–26, 1974. Edited by E. Zwicker, E. Terhardt

Vol. 9 G. Hammarström

Linguistic Units and Items

Springer-Verlag Berlin Heidelberg New York

Communication and Cybernetics

(continued)

Vol. 10 **Digital Pattern Recognition**
Edited by K.S.Fu. With contributions by T.M.Cover, E.Diday, K.S.Fu, A. Rosenfeld, J. C. Simon, T. J. Wagner, J. S. Weszka, J. J. Wolf

Vol. 11 **Structure and Process in Speech Perception**
Proceedings of the Symposium on Dynamic Aspects pf Speech Perception, held at I.P.O., Eindhoven, Netherlands, August 4–6, 1975. Edited by A. Cohen and S. G. Nooteboom

Vol. 12 J. D. Markel, A. H. Gray Jr.
Linear Prediction of Speech

Vol. 13 R. G. Busnel, A. Classe
Whistled Languages

Topics in Applied Physics

Founded by H. K. V. Lotsch

Vol. 6 **Picture Processing and Digital Filtering**
Edited by T. S. Huang. With contributions by H. C. Andrews, F. C. Billingsley, J. G. Fiasconaro, B. R. Frieden, T. S. Huang, R. R. Read, J. L. Shanks, S. Treitel

Vol. 11 **Digital Picture Analysis**
Edited by A. Rosenfeld. With contributions by S.J.Dwyer, R.M.Haralick, C.A.Harlow, G. Lodwick, R. L. McIlwain, K. Preston, A. Rosenfeld, J.R.Ullmann

---

Springer-Verlag Berlin Heidelberg New York