



City Research Online

City St George's, University of London

Citation: Popovici, V. (1976). Application of Syntactic Pattern Recognition to Detect Classification in Optical Surface Inspection. (Unpublished Doctoral thesis, The City University)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/37960/>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

THE CITY UNIVERSITY
DEPARTMENT OF SYSTEMS SCIENCE

"APPLICATION OF SYNTACTIC PATTERN RECOGNITION
TO DETECT CLASSIFICATION IN OPTICAL SURFACE INSPECTION"

VOLKER POPOVICI

A thesis submitted for the award of the degree of
Doctor of Philosophy in Systems Engineering.

November 1976.

APPLICATION OF SYNTACTIC PATTERN RECOGNITION
 TO DEFECT CLASSIFICATION IN OPTICAL
SURFACE INSPECTION

Contents

	Page
Ch. 0 <u>Introduction</u>	1
Ch. 1 <u>Defect recognition in automatic surface inspection systems</u>	4
1.1 Reasons for automatic inspection 	4
1.2 The principle of optical surface inspection ..	5
1.3 The functions of signal processing 	8
1.3.1 Defect delineation 	9
1.3.2 Defect classification methods 	11
Ch. 2 <u>Formulation of the syntactic pattern recognition problem. Definitions and notations</u>	22
2.1 Selected topics in formal language theory ..	22
2.1.1 Sets of strings 	25
2.1.2 Procedures and algorithms 	25
2.1.3 Languages, grammars and recognizers ..	26
2.1.4 Properties of regular grammars 	28
2.2 Formal languages for pattern recognition 	31
2.3 General structure of a syntactic pattern recognition system 	34

	Page
Ch. 3 <u>Review of previous work in syntactic pattern recognition of imperfectly structured patterns</u>	37
3.1 Stochastic grammars and stochastic languages ..	39
3.2 Fuzzy grammars and fuzzy languages	42
3.3 Error correcting techniques	45
3.4 Grammatical inference	48
3.4.1 Problem statement	49
3.4.2 Basic definitions	49
3.4.3 Fundamental results in grammatical inference	50
3.4.4 Approximate language identification ..	51
3.4.5 Review of grammatical inference techniques	51
Ch. 4 <u>Syntactic representation of defects</u>	53
4.1 Selection of pattern primitives and composition descriptors	53
4.1.1 Pattern primitives	54
4.1.2 Composition descriptors	57
4.1.3 Evaluation of the separation capabilities of the pattern primitives and composition descriptors	60
4.2 The formalism for describing two-dimensional patterns as strings of pattern primitives and composition descriptors	60
4.3 Extraction of the syntactic representation of defects	62
4.3.1 Multiple defects	63
4.3.2 Merging separate defects	68
4.3.3 Non-contiguous defects	70
4.3.4 Synthesis of the string representation of defects from the sequence of SIUs	70
4.4 Programs for extracting the syntactic representa- tion of defects	72
4.4.1 Information structure	76
4.4.2 The program CLUMPL	80
4.4.3 The program LIPAN	82
4.4.4 The hash coding subroutine (HASH)	84

	Page
Ch. 5 <u>Inference of regular pattern grammars</u>	86
5.1 Algorithm for synthesizing a finite state automaton from a sample of a regular language ..	87
5.1.1 The Nerode relation 	87
5.1.2 The modified Nerode relation 	88
5.1.3 Correct extrapolation from f_T to f	90
5.2 Generalised algorithm for constructing FSA acceptors of multi-class two-dimensional patterns from samples of languages 	92
5.3 Practical implementation of the generalised algorithm 	95
5.3.1 Selection of the initial set A_f	96
5.3.2 Computation of the set B_r	96
5.3.3 Computation of the equivalence classes ..	96
5.3.4 Construction of the final A_f set 	97
5.3.5 Construction of the final equivalence classes 	97
5.3.6 Computation of the state transition matrix of the acceptor 	97
5.3.7 Preliminary performance estimation 	98
5.4 Description of the grammatical inference program (GRINF) 	99
5.4.1 Program input 	99
5.4.2 Program output 	99
5.4.3 Data bases 	99
5.4.4 The main program 	101
5.4.5 Subroutines called by the program GRINF ..	101
5.5 Conclusions 	105

	Page
Ch. 6 <u>Classification of syntactically imperfect patterns</u>	107
6.1 The concept of edit distance 	108
6.2 Algorithm for the computation of MNEO 	113
6.2.1 Computation of MNEO 	113
6.2.2 Computation of $V(T, S, c)$	114
6.2.3 Computation of $P(T, S)$	114
6.2.4 Computation of $L(T, S, c)$	115
6.2.5 Time and space complexity of the algorithm 	115
6.3 Generalized MNEO algorithm for the classifi- cation of multi-class two-dimensional patterns	116
6.3.1 Handling recursive productions 	117
6.3.2 Processing strings of syntactic elements	118
6.3.3 Classification of multi-class patterns ..	119
6.4 Program SYNREC for syntactic recognition of imperfect patterns 	120
6.4.1 Data bases 	121
6.4.2 The main program 	121
6.4.3 Subroutines 	123
Ch. 7 <u>Acquisition of a data base</u>	126
7.1 Introduction 	126
7.2 Description of the data base 	127
7.3 Functions and operation of the test rig 	130
7.3.1 Phase nr. 1 - operations performed by the operator 	132
7.3.2 Phase nr. 2 - operations performed auto- matically 	132
7.3.3 Error conditions 	134
7.4 Test rig control 	134
7.4.1 PDP-11 interface to the tape unit	135
7.4.2 PDP-11 interface to the test rig 	139
7.5 Formatting of data 	143

	Page
Ch. 8 <u>Computer simulation of the syntactic pattern recognition system</u>	148
8.1 Syntactic representation of the defects 	149
8.1.1 Selection of the capture rectangle dimensions 	149
8.1.2 Selection of the syntactic elements ..	150
8.2 Synthesis of the syntactic pattern classifier ..	161
8.2.1 Grammar inference for the classes of short defects with a variable structure	167
8.2.2 Synthesis of the MNEO distance pro- cessors for the classes of defects described by short strings with a variable structure	170
8.2.3 Direct synthesis of the MNEO distance processors for the classes of defects described by long strings with a simple structure 	177
8.3 Recognition performance of the syntactic defect classification system 	179
8.3.1 Analysis of defect rejections 	187
8.3.2 Analysis of misclassifications 	188
8.4 Conclusions 	188
Ch. 9 <u>Logic design of a real-time syntactic pattern recognition system</u>	190
9.1 General description of the proposed syntactic pattern recognition system 	190
9.2 Logic structure of the pattern description block	194
9.2.1 Extraction of pattern primitives 	194
9.2.2 Extraction of composition descriptors ..	196
9.2.3 Generating and counting the NOS syntactic elements	198
9.3 The SIU buffer memory 	201

(Chapter 9 continued)

	Page
9.4 Implementation of a syntactic pattern classifier using a microcomputer	203
9.4.1 Selection of the microcomputer configuration	206
9.4.2 Implementation of the MIN-DIST subroutine	208
9.4.3 Implementation of the UPDATE subroutine	209
9.5 Estimation of speed parameters	212
9.6 Conclusions	215
Ch. 10 <u>Conclusions</u>	216
<u>Appendix I</u>	218
<u>References</u>	225

LIST OF TABLES

1. Comparative evaluation of pattern recognition methods
2. Definition of pattern primitives
3. Definition of composition descriptors
4. Extraction of syntactic information units
5. Merging pattern primitives
6. Merging composition descriptors
7. Syntactic representation of defects
8. Data base
9. Syntactic representation of the defects used in the simulation experiment
10. Class distribution of the syntactic elements and of the string lengths
11. Transition matrices of the recognizers
12. Correction tables
13. Simulation results
14. Analysis of the classification errors

LIST OF ILLUSTRATIONS

- 1.1 SIRA 550 laser scanner
- 1.2 Photomultiplier output signal
- 1.3 Defect detection and delineation
- 1.4 Specular videoprint of a tinplate sheet
- 1.5 Matrix representation of an artificial pattern
- 1.6 Videoprint of an artificial pattern
- 1.7 Performance of the feature space classifier
- 1.8 Performance of the linguistic classifier

- 2.1 Block diagram of a pattern recognition system for on-line applications

- 4.1 Definition of the degree of connectivity
- 4.2 FSA transition matrix
- 4.3 Illustration of defect delineation and syntactic representation of the defects on the videoprint
- 4.4 Structure of EXSYN block
- 4.5 Merging defects
- 4.6 Flowchart of program extracting the syntactic representation of defects
- 4.7 Structure of data block
- 4.8 Hash coding procedure
- 4.9 Processing of scan sections
- 4.10 Processing merged defects

- 5.1 Flowchart of program GRINF
- 5.2 Determination of the equivalence classes
- 5.3 Subroutine TRANS
- 5.4 Language approximation

- 6.1 Languages defined by error operators
- 6.2 Errors introduced by approximations
- 6.3 Worst case situation
- 6.4 Array IDSP
- 6.5 Flowchart of program SYNREC
- 6.6 Flowchart of subroutine ACCEPT

- 7.1 Structure of test rig
- 7.2 Block diagram of the interface between computer and tape unit
- 7.3 PDP-11 interface to the electromechanical parts of the test rig
- 7.4 Structure of data generated by scanner
- 7.5 Data after formatting in PDP-11 computer
- 7.6 Data in CDC 7600 computer

- 9.1 Block diagram of syntactic pattern recognition system
- 9.2 Extraction of pattern primitives
- 9.3 Extraction of composition descriptors
- 9.4 Generating and counting the NOS syntactic elements
- 9.5 SIU buffer memory
- 9.6 General organization of classification program
- 9.7 Microcomputer based classifier
- 9.8 Flowchart of subroutine MIN-DIST
- 9.9 Flowchart of subroutine UPDATE

ACKNOWLEDGEMENTS

I wish to express my sincere thanks and gratitude first and foremost to Professor Ludwik Finkelstein for his constant encouragement, direction and helpful advice in the development of the programme of work reported in this thesis and its final presentation.

I am also indebted to the members of the research team on automatic inspection at the Instrument Systems Centre: L. [REDACTED], for regular discussions, guidance and coordination of this project and W.J. [REDACTED] whose parallel investigation resulted in the development of algorithms and programs used in this thesis and for his enthusiastic cooperation.

I wish to express my gratitude to Richard [REDACTED] Section Leader in the Optical Systems Department at SIRA Institute for his invaluable assistance in acquiring the data base and in the construction of an automatic data gathering system.

I also wish to thank the members of the Department of Computer Engineering at the Technical University Timisoara for their efforts and help which enabled me to carry out this work and the Ministry of Education, Bucharest, for their financial support.

Finally, I am grateful to Ms. Kaye [REDACTED] and Mrs. Audrey [REDACTED] for their skillful preparation of the typescript.

ABSTRACT

The thesis presents the results of a theoretical and experimental study on the application of syntactic pattern recognition to defect classification in optical surface inspection.

The inspected surface is interrogated with visible light and as a result of this interaction, the characteristics of the reflected light provide information about the surface. Signal processing techniques are applied to detect and delineate the defects on the surface. The syntactic pattern recognition system designed in the thesis operates on the specular videoprint, which is a black-white representation of the defects extracted from the specularly reflected light.

A pattern description language which is well suited for on-line processing is developed for the representation of two-dimensional defects as strings. The languages which describe the classes of defects are approximated by regular grammars obtained automatically using an efficient grammar inference algorithm. The classification of the defects is achieved using an algorithm based on the concept of edit operations which reduces the errors due to the approximation of languages and to the imperfect structure of the defects. A set of programs written in FORTRAN is used for performing a detailed analysis of the system in computer simulation.

The validity of the theoretical methods proposed is tested on a data base of real defects, acquired using an automated data gathering system. The experimental results obtained in simulation with this data

base indicate an acceptable performance of about 70 - 80% correct recognitions. A hardware implementation of the classification system is outlined and shown to be economical and capable of operating in real time.

The interpretation of the results reveals the limitations of the present system and suggests techniques for further investigation to improve the recognition performance.

CONVENTIONS OF REPRESENTATION

alphabet: Σ

terminal symbols: $a, b, c, d, \dots$

nonterminal symbols: A, B, C

start symbol: S

strings of nonterminals and terminals: $\alpha, \beta, \gamma, \delta, \dots$

strings of terminals only: $u, v, \dots, z$

empty string: λ, ε

LIST OF ABBREVIATIONS

BSS	-	beginning of scan section
DFU	-	data formatting unit
DMA	-	direct memory access
EOS	-	end of scan
ESS	-	end of scan section
FSA	-	finite state automaton
G	-	grammar
GI	-	grammatical inference
L(G)	-	language generated by grammar G
NOS	-	no scan section
RAM	-	random access memory
ROM	-	read only memory
SIU	-	syntactic information unit

0 . INTRODUCTION.

This thesis presents the results of a theoretical and experimental study on the application of syntactic pattern recognition to the classification of visually perceivable defects on a moving surface. This investigation is part of a research project carried out at the Instrument Systems Centre in cooperation with the SIRA Institute and the British Steel Corporation.

Syntactic pattern recognition methods were proposed about 12 years ago for problems in which feature space techniques did not provide an efficient solution. In certain situations syntactic methods offer potential advantages (simplicity and fast operation) but until recently the area of applicability had been limited to well-formed, noise-free patterns.

In the last 2 - 3 years, a series of new techniques have been suggested to overcome these difficulties, but their efficiency has not been proved convincingly with real data problems.

The main object of the research programme reported in this thesis was to select and develop syntactic methods into a complete and unitary pattern recognition system and to evaluate its performance by computer simulation using a representative, real data base.

The solutions proposed in this thesis represent developments and extensions of existing theoretical techniques and incorporate contributions which can be summarised as follows:

- 1) development of a language for the description of two-dimensional patterns which is suited for on-line processing.
- 2) formulation of an efficient algorithm for inference of approximate regular grammars for n-class, two-dimensional patterns and development of a corresponding FORTRAN program.
- 3) development of a pattern classification algorithm and program based on the concept of syntactic distance which reduces the errors due to the approximation of the pattern language and to the variability in the structure of patterns.
- 4) acquisition of a representative data base of surface defects using a specially designed, automated test rig controlled by a minicomputer.
- 5) validation of the theoretical contributions using this data base.

The thesis is structured in chapters which cover the subject as follows:

Chapter 1 establishes the general framework by describing briefly the structure of the automatic inspection system and the requirements imposed on the pattern classifier.

Chapter 2 introduces the definitions and notations used in the following chapters and formulates the syntactic pattern recognition problem.

Chapter 3 contains a brief survey and critical evaluation of previous connected work.

Chapter 4 presents the formalism which has been developed for describing syntactically two-dimensional defects as they are encountered on-line.

Chapter 5 is dedicated to the description of the algorithm employed for grammatical inference and outlines the structure of the corresponding program.

Chapter 6 gives a detailed presentation of the classification technique developed using syntactic distances and describes the organisation of the corresponding program.

Chapter 7 emphasises the need and importance of the automated test rig constructed and describes the structure of the data base used in this study.

Chapter 8 analyses the main results of the simulation of the syntactic pattern recognition system using the acquired data base and suggests an interpretation of these results.

Chapter 9 gives a brief description of the logical structure of a syntactic classifier based on the techniques developed in the thesis, in order to show its feasibility for real time applications.

The final conclusions summarise the results obtained and indicate problems which are open to further investigation.

1. DEFECT RECOGNITION IN AUTOMATIC SURFACE INSPECTION SYSTEMS.

This chapter gives a brief description of the automatic surface inspection system in which the pattern recognition automaton is intended to work, in order to establish the requirements imposed on its functional properties. A detailed presentation of the system is given in [1].

1.1 Reasons for automatic inspection

The inspection of product quality is an important and general requirement. At the present time many inspection procedures are carried out by human inspectors, but the automatic instrumental inspection would improve the efficiency in many respects, such as : objectivity, speed, consistency, sensitivity, reliability and elimination of fatigue.

The advantages of automatic inspection methods over human inspection become evident in high speed production line processes, because:

1. the information handling capacity of the human inspector is too low
2. these processes are automated right up to the inspection phase; automatic inspection instrumentation can be introduced as a feedback element to complete fully automatic control loops.

The perception and judgement capabilities of instruments are very limited and techniques of signal processing and artificial intelligence must be employed to replace these functions of the human inspector.

1.2 The principle of optical surface inspection. [1], [2]

The principle of automatic inspection in the present application is based on the interrogation of the product with an energy input which interacts with the surface so that the output energy is modulated corresponding to the properties of the interrogated region. The automated test rig which had been used to acquire the data base of defect patterns was built around an optical inspection system developed by the SIRA Institute, based on the flying-spot laser scanner type 550. For this case, the general inspection system can be regarded as a sequence of three operations:

- 1) the surface is interrogated by illuminating it with visible light.
- 2) the surface interacts with the light and consequently changes one or more of the characteristics of the light returned to the scanner.
- 3) these changes are measured and processed to give the desired information about the surface.

The essential components of the type 550 Laser Scanner are shown in fig. 1.1. A thin beam of light is scanned across the inspected surface by the rotation of a mirror polygon. A large cylindrical lens focuses the beam onto the surface, where it is reflected, scattered and absorbed. The system as shown senses only one parameter of the beam / surface interaction - namely, surface reflectivity. A photomultiplier is positioned at the specular angle with respect to the interrogating beam; the output voltage is proportional to the amount of light reaching the cathode.

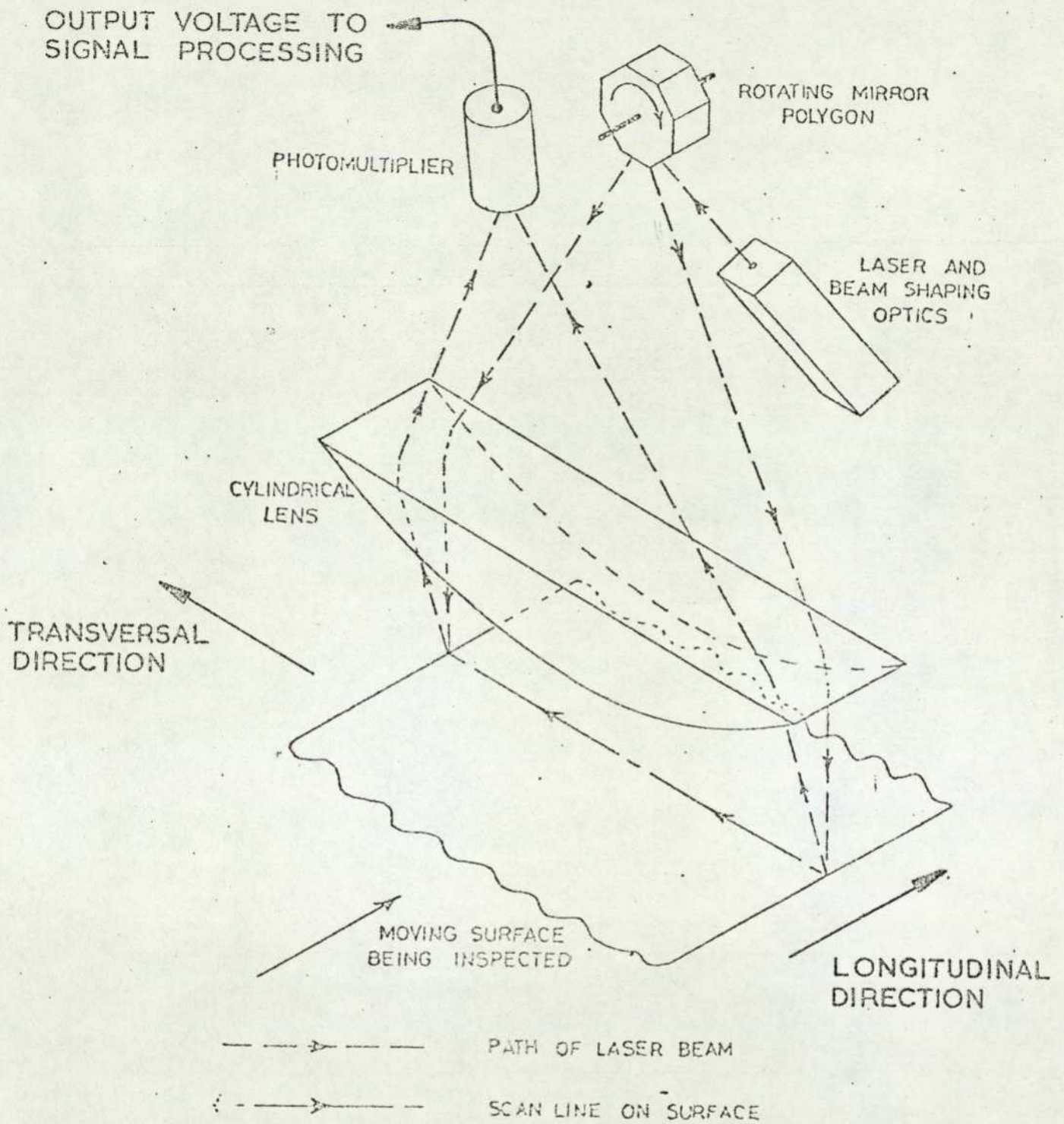


FIG. I.1 SIRA 550 LASER SCANNER

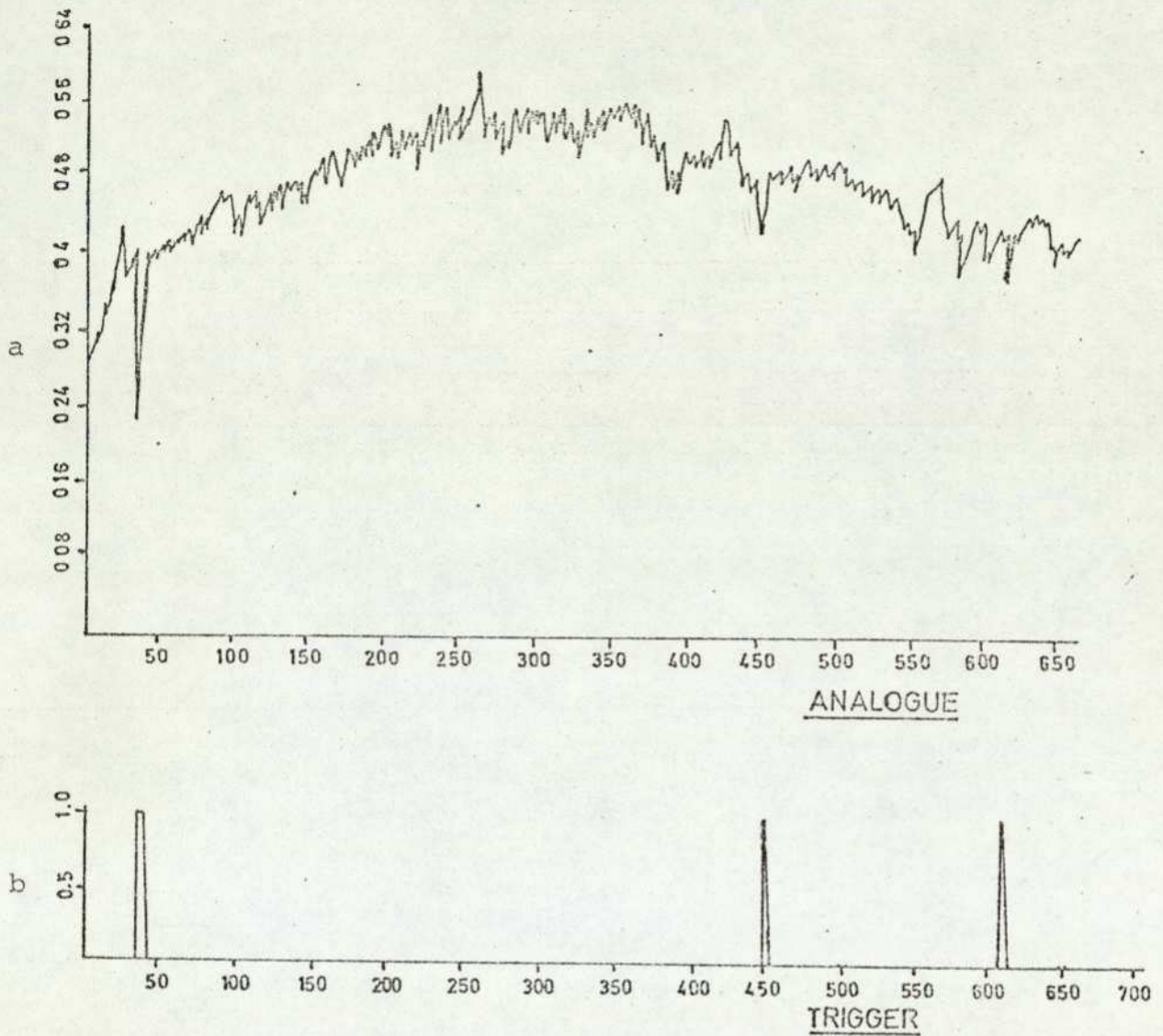


Fig. 1.2 Photomultiplier output signal

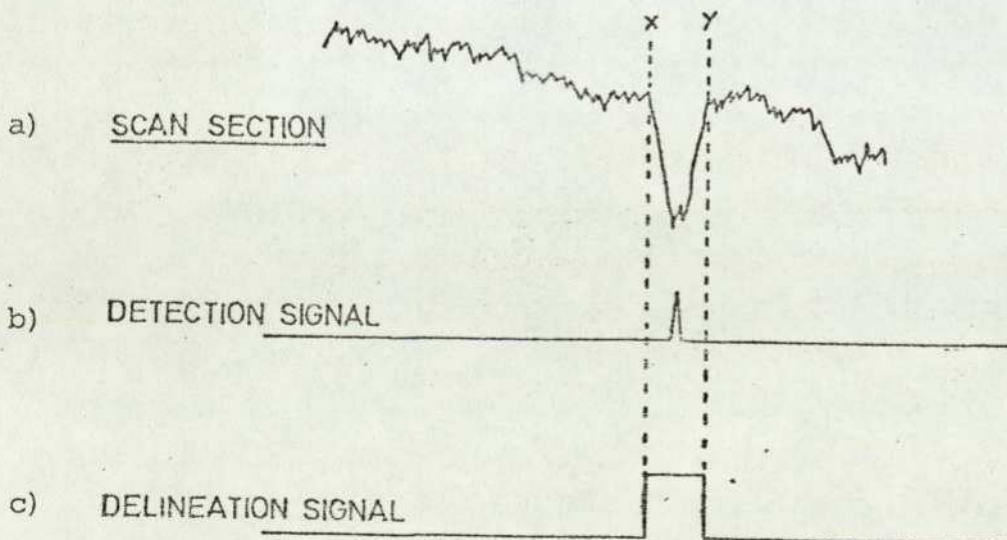


Fig. 1.3 Defect detection and delineation

With a perfect (mirror-like) surface, the output of the photomultiplier will be maximum, because the interaction is confined to a simple reflection. In practice, surfaces are far from perfect mirrors and the amount of light reaching the photomultiplier is reduced by scattering and absorption. The photomultiplier output signal (Fig. 1.2a) is reduced momentarily by the presence of the defect, hence defects are indicated by negative-going pulses.

By matching the speed of the rotation of the mirror polygon to the velocity of the moving surface, successive scans will cover the entire surface.

The signal at the output of the photomultiplier is processed by employing analogue filtering followed by threshold detection, to provide a very effective defect detection; this function of the inspection system is discussed in the next paragraph. In most applications defect classification is an additional requirement; pattern recognition techniques are a possible solution to this problem and will be investigated in detail in the following chapters.

1.3 The functions of signal processing

The general function of signal processing in an inspection system is to extract the required information from the product response to signal interrogation. In the present case, signal processing techniques are used to implement the following operations on the scanner output signal (fig. 1.3a):

- 1) defect detection : a signal is generated when a defect is present (fig. 1.3b)
- 2) defect delineation : involves finding the boundary of the defect and consists of two stages:

a) processing of the signals generated during individual scans of the surface, in order to select those scan sections which derive from surface defects (fig. 1.3c)

b) processing of the scan sections determined in the previous stage, so as to associate all those sections which derive from the same surface defect.

3) defect classification : determination of the nature of a defect delineated as mentioned before.

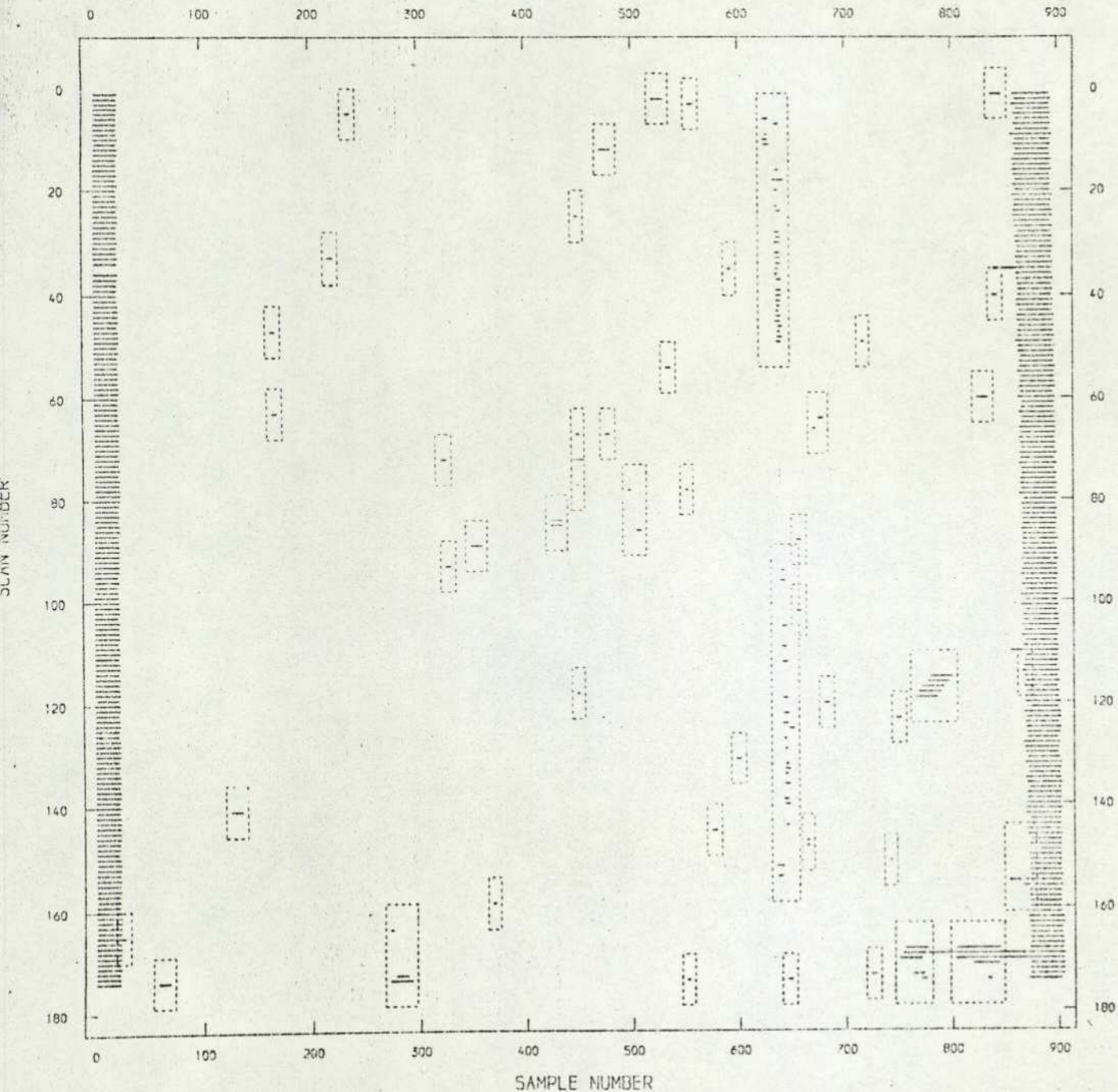
Defect detection can be regarded as a simplified case of the more general defect delineation and will, therefore, not be considered in more details.

1.3.1 Defect delineation

Defect delineation is an important prerequisite for reliable defect classification and has been investigated in detail in the report by W.J. Hill [3].

The signals which correspond to defects on the surface are negative-going pulses, which may last from a few microseconds, to one hundred microseconds and have a variable amplitude. These signals must be extracted from noise which is known to be substantially white, within the region of the spectrum occupied by the energy in the defects.

The first stage of defect delineation is implemented using a bank of matched filters chosen to cover the defect waveforms of interest. Each filter produces a pulse when it detects the waveform to which it is matched; the filter producing the best match determines the limits of the defect in the input signal. The result of this operation is a series of binary trigger samples from



TOP L/H
CORNER.
SCAN/SAMPLE

DEFECT
NUMBER.

1/ 833	1
3/ 515	2
2/ 549	3
0/ 229	4
1/ 619	5
7/ 466	6
20/ 443	7
28/ 213	8
30/ 586	9
35/ 835	10
42/ 159	11
44/ 711	12
49/ 527	13
55/ 820	14
58/ 160	15
59/ 665	16
62/ 443	17
62/ 470	18
67/ 316	19
72/ 443	20
73/ 491	21
73/ 545	22
79/ 419	23
83/ 649	24
84/ 344	25
89/ 321	26
87/ 630	27
97/ 649	28
109/ 862	29
110/ 761	30
113/ 443	31
115/ 675	32
118/ 743	33
126/ 592	34
136/ 120	35
140/ 549	36
142/ 658	37
144/ 849	38
146/ 736	39
154/ 363	40
159/ 267	41
160/ 19	42
163/ 746	43
163/ 798	44
168/ 719	45
169/ 53	46
169/ 546	47
169/ 640	48

10

Fig. 1.4 Specular videoprint of
a tinplate sheet.

each scan, where a trigger bit is set to one if it lies within a defect section, and to zero otherwise (fig. 1.2b). The image obtained in this manner for a number of successive scans is presented in fig. 1.4 and can be interpreted as a black-white representation of the defects; it will be referred to as a "videoprint".

The second stage of defect delineation associates trigger samples lying within a certain present distance of each other. To accommodate the tendency of many defects to be elongated in the rolling direction, different distances can be used for the longitudinal and transversal direction. Each trigger sample generates a "capture rectangle" and any other trigger sample falling within that rectangle becomes associated with the same defect. The boundaries of the defects delineated using this technique are illustrated in fig. 1.4, which represents the videopoint of a 1m x 1m sheet of triplate. (For practical reasons, the data base used throughout the thesis was obtained by scanning a number of 1m x 1m sheets of tinplate, rather than a continuous, long, strip; the sheets were cut at the BSC works from the strip because they contained defects.)

1.3.2 Defect classification methods

In the present application, defect classification can be implemented using two types of information provided by the scanner:

- a) the original analog signal at the output of the scanner (specular and off-specular)
- b) the videoprint (specular and off-specular)

The research work which is reported in this thesis is concerned with the classification of the defects using the information contained in the specular videoprint only. In the

initial phase of the investigation, a comparative study of the pattern recognition techniques described in the literature was carried out, in order to select the most promising methods. This evaluation was performed by taking into account the technical specifications of the process and of the optical inspection system, since these impose certain requirements on the pattern recognition system, which will be outlined in the following discussion.

The present project studied the general problem of surface inspection with reference to the inspection of tinplate for which an industrial need and a potential data base existed. Tinplate is produced as a continuous strip at a velocity of up to 25m/s. The data is generated, scan by scan, at a very high rate : if the sheet is 1 m wide and the signal contains 1000 resolved points across the scan width, renewed every centimeter, the information is generated at 10 Mbit/sec. Defect delineation must be implemented at this rate, but defect classification can be carried out at a lower rate, because only a small area of the surface will be covered by defects. An exact estimation of the average time available is difficult, but the order of magnitude is 10 msec, which excludes the use of a dedicated minicomputer programmed for a certain pattern recognition algorithm.

The physical nature of the defects varies in time and this leads to patterns characterised by a high degree of variability; even within the same class, the size and shape of the defects varies, though retaining a common structure.

In summary, a pattern classifier for defects on a moving surface must satisfy the following requirements:

(1) good performance for a considerable degree of pattern variability within a class.

(2) high speed operation.

(3) simple hardware implementation, to achieve high reliability and low cost.

(4) the classifier is designed using a finite set of training patterns; the method of classification must minimize the errors introduced by the extrapolation to a new set of patterns.

Considering these factors, the problem of selecting an efficient defect classification method was tackled in two stages:

1) A certain number of candidate techniques were analysed and compared according to the criteria listed before; this comparative study is presented in detail in [4] and the main results are summarized in table 1. It became evident at an early stage that no available technique will solve the problem completely satisfactorily using specular videoprint data only. Other types of information, like off-specular videoprints and analog signals in successive scans and possibly a combination of different pattern recognition methods will have to be used in order to achieve the required performance.

As a result of this first stage, template matching and logical decomposition techniques were rejected as inadequate for the present application. The techniques which emerged as potentially applicable were the feature space method and the linguistic method. Since an investigation of the feature space method using geometrical properties as features had already started within the research team,

TABLE 1. COMPARATIVE EVALUATION OF PATTERN RECOGNITION METHODS
FOR REAL TIME DEFECT CLASSIFICATION.

METHODS CRITERIA	TEMPLATE MATCHING	LOGICAL DECOM- POSITION	FEATURE SPACE WITH GEOMETRIC PROPERTIES	FEATURE SPACE USING TWO- DIMENSIONAL TRANSFORMS	LINGUISTIC
PERFORMANCE	unacceptable for reasonable cost	very good, not considering cost	generally good to be determined by simulation	to be deter- mined by simulation	to be deter- mined by simulation
SPEED	high	very high	acceptable	acceptable	satisfactor matched to the scanning system
COST	unacceptable for satisfactory performance	unacceptable for satisfactory performance	unacceptable	high	low
STORAGE REQUIREMENTS	very high	no storage necessary	acceptable	reduced	acceptable
POSITION INVARIANCE	no	possible using shift register techniques	possible	yes	yes
TOLERANCE OF NOISE, SHAPE AND SIZE VARIATIONS	unacceptable	good not considering cost	good	good	possible by including additional facilities
GENERALIZ- ATION POSSIBILITIES	yes	no	yes	yes	yes
CONFIDENT EXTRA- POLATION	no	no	yes	yes	yes
EASE OF DESIGN	good	difficult	good	good	difficult

the present study continued with the linguistic method and feature space method using as features the components of the two-dimensional BIFORE transform as the main options.

2) In the second stage of the preliminary study, it was decided to perform a number of experiments in simulation, using initially only artificial data, in order to obtain indications related to:

a) the performance of the techniques, as measured by % correct classifications.

b) relationship between performance and amount of noise corrupting the ideal patterns.

c) difficulties to be overcome in order to improve the performance.

Although the properties of the artificial patterns have been selected somewhat arbitrarily, the following factors were taken into account to ensure a reasonably sound basis for the test:

a) a nontrivial number of classes (6)

b) the number of samples per class great enough to cover a reasonable variability of properties within a class (20 prototypes and 60 patterns distorted by noise)

c) provision of a wide range of geometrical and topological properties of the patterns (connectivity, directionality, etc.)

Each pattern is in the form of a binary (black on white background) two-dimensional videoprint, stored in a matrix of 32 x 32 elements (fig. 1.5); a total number of 480 patterns in 6 classes are stored on magnetic tape as a permanent data file.

The 6 classes of patterns used in this experiment were the following (examples shown in fig. 1.6):

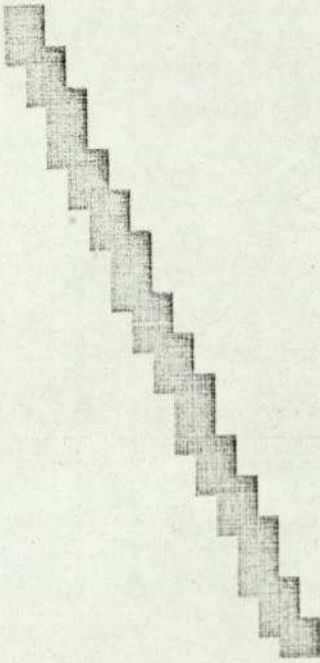
- a) rings of different thickness and radius
- b) linear quasihorizontal patterns of different shape and thickness
- c) linear quasivertical patterns of different shape and thickness.
- d) circular spots of different radius, distorted by randomly shifting adjacent rows of the matrix to left or right.
- e) horizontal broken lines
- f) small dots, randomly distributed within the matrix

The effect of noise and distortions was simulated by inverting the elements of the matrix (1 to 0, 0 to 1); within each class, four subclasses have been created, differing by the probability p of a cell in the videoprint being inverted ($p = 0.0; 0.01; 0.02; 0.03$).

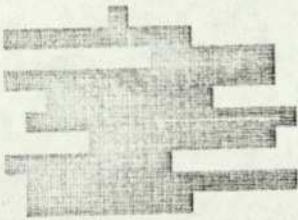
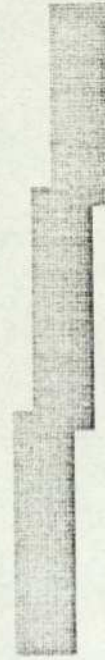
The FORTRAN program for generating the artificial patterns consists of cca 500 lines and is executed on the CDC 7600 computer in 20 sec., including the plotting of 120 patterns on microfilm.

The next step of the experiment was to develop a set of FORTRAN programs for designing and simulating simple pattern classifiers based on the feature space and linguistic methods, respectively. Finally, the performance of these classifiers was evaluated in simulation using as data the artificial patterns; the experiments are fully described in [4] and [5].

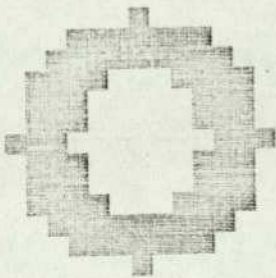
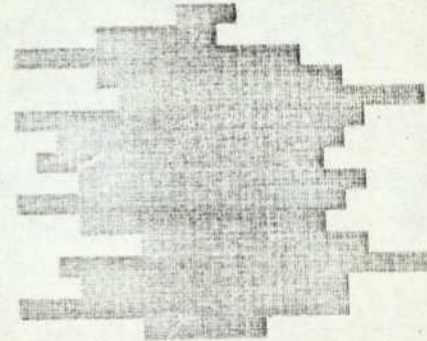
18



vertical lines



spots



rings

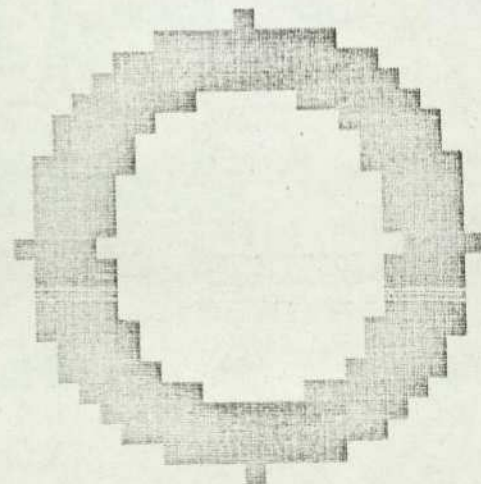
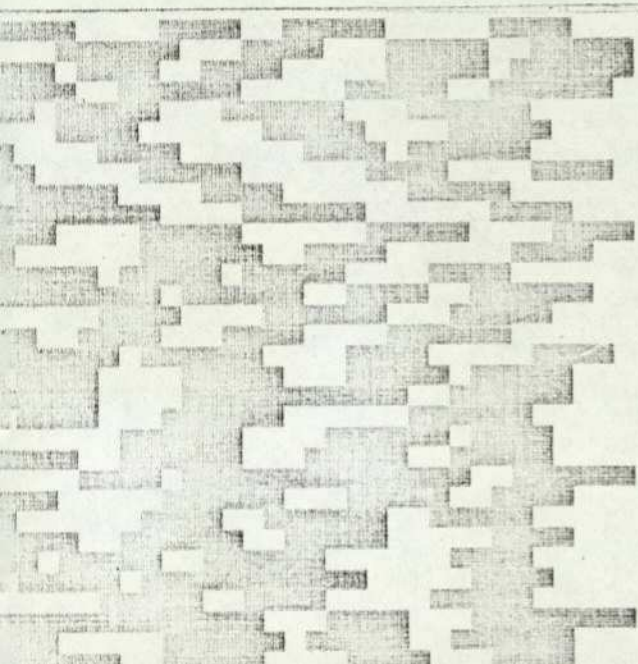
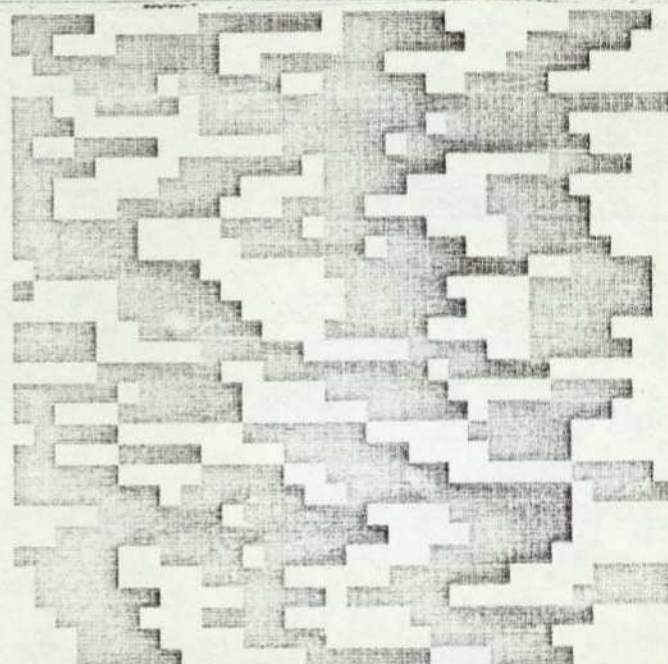


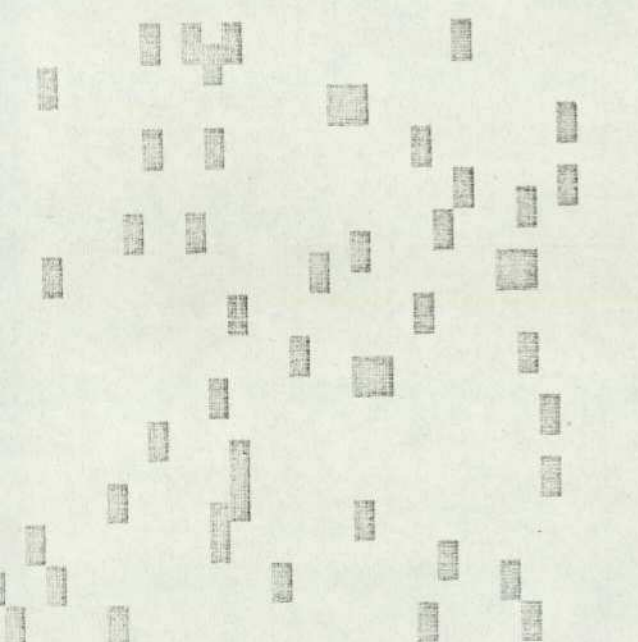
Fig. 1.6 Videoprints of artificial patterns



horizontal lines



broken lines



dots

The final conclusions of this preliminary study were the following:

1) The performance of the classifier based on the feature space method (using the components of the two-dimensional BIFORE transform as features) was satisfactory and relatively insensitive to noise (see fig. 1.7), but the computations involved were complex, therefore a fast hardware implementation would be expensive.

2) The performance of the linguistic classifier was excellent for noise-free patterns, but exhibited a fast degradation as the amount of noise increased (see fig 1.8, curve A). By applying a logical smoothing operator, using a 3×3 window before starting the recognition, the overall performance could be improved, but not significantly (see fig. 1.8 , curve B). The patterns mentioned before could be described using regular grammars and therefore the structure of the linguistic classifier was simple and the speed of operation very high. This is an essential advantage for on-line applications and finally it was decided to concentrate the research effort on the development of an advanced syntactic pattern recognition system, by incorporating additional facilities for:

- a) improving the recognition performance of distorted patterns,
- b) inferring grammars automatically, using a set of patterns for training.

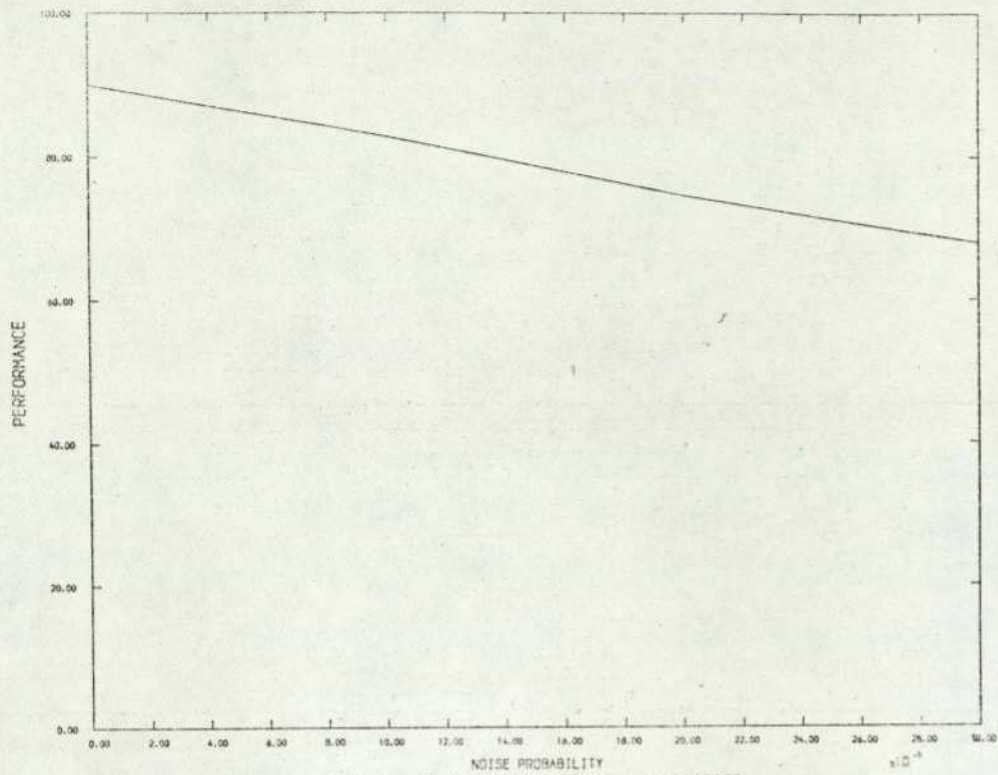


FIG. 1.7 FEATURE SPACE (BIFORE) CLASSIFIER

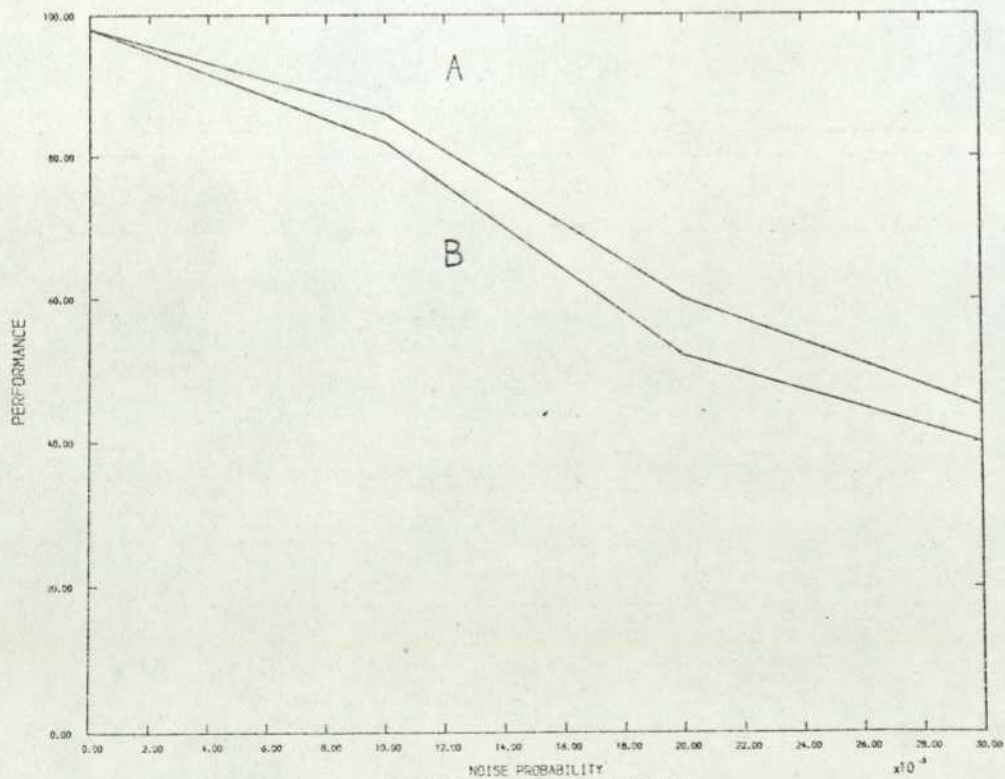


FIG. 1.8 LINGUISTIC CLASSIFIER

2. FORMULATION OF THE SYNTACTIC PATTERN RECOGNITION PROBLEM.

DEFINITIONS AND NOTATIONS

The syntactic pattern recognition method compares a structural description of classes of patterns using formal languages with a syntactic representation of a specific pattern in order to achieve a correct classification. The method is based on two concepts developed in formal language theory:

1) description of a large number of patterns belonging to a class using a small number of simple pattern primitives, relationships between them and a recursive grammar.

2) recognition of a pattern by parsing its syntactic representation in order to find the language to which it belongs.

This chapter introduces the definitions, notations and selected topics in formal language theory which are used in the thesis; these concepts will then be extended to incorporate features necessary to describe two-dimensional patterns. The last section describes the general structure of a pattern recognition system based on the syntactic method.

2.1 Selected topics in formal language theory

In order to make the thesis self-contained, this paragraph will be devoted to the introduction of the notations and definitions used in the thesis and to a brief discussion of selected results in formal language theory relevant to syntactic pattern recognition [6], [7].

Definitions and notations from set theory

Sets will be denoted by capital letters ($A, B, \dots$) and the elements of sets by little letters ($a, b, \dots$). If a is a member of A , this is expressed by the notation $a \in A$; if this property is not valid, $a \notin A$. The empty set, which has no members, is denoted by $\emptyset$.

The set A is included in set B , written $A \subseteq B$, if every element of A is also an element of B . If B contains at least one element not in A , then A is properly included in B , written $A \subset B$.

The union of sets A and B , written $A \cup B$, is the set containing all elements in B ; formally, $A \cup B = \{x | x \in A \text{ or } x \in B\}$

The intersection of sets A and B , written $A \cap B$, is the set of all elements that are both in A and B ; formally,

$$A \cap B = \{x | x \in A \text{ and } x \in B\}.$$

An ordered pair, denoted (a, b) , consists of a and b in that order. The Cartesian product of sets A and B , denoted $A \times B$ is defined as

$$\{(a, b) | a \in A, \text{ and } b \in B\}$$

A relation from a set A to a set B is any subset of $A \times B$. If R is a relation from A to B , we write $a R b$ whenever (a, b) is in R .

Let A be a set and R a relation on A . We say that R is

- (a) reflexive if aRa for all $a \in A$
- (b) symmetric if aRb implies bRa for $a, b \in A$.
- (c) transitive if aRb and bRc implies aRc , for $a, b, c \in A$.

An equivalence relation is a relation which is reflexive, symmetric and transitive. An equivalence relation R on A partitions

A into disjoint subsets called equivalence classes.

The k-fold product of a relation R (on A), denoted R^k , is defined as follows:

$$(a) \quad aR^1b \text{ iff* } aRb$$

$$(b) \quad aR^ib \text{ iff there exists } c \in A$$

such that aRc and $cR^{i-1}b$ for $i > 1$.

The transitive closure of a relation R on a set A , denoted R^+ , is defined as follows:

$$aR^+b \text{ iff } aR^ib \text{ for } i > 1.$$

The reflexive and transitive closure of a relation R on a set A is denoted R^* and is defined as follows:

$$(a) \quad aR^*a \text{ for all } a \in A$$

$$(b) \quad aR^*b \text{ if } aR^+b$$

(c) nothing is in R^* unless its being there follows from (a) and (b)

* iff is the usual abbreviation for "if and only if".

2.1.1 Sets of strings

An alphabet (or vocabulary) is a finite set of symbols.

A string is a finite sequence of symbols from an alphabet.

The empty string is that string which has no symbols.

Conventions of representations:

(1) Symbols: $a, b, c, d, \dots$

(2) Alphabet: Σ (sometimes X)

(3) Strings: $u, v, w, x, y, z, \alpha, \beta, \gamma, \delta, \dots$

Empty string: λ or ϵ

(4) Σ^* denotes the set containing all strings over Σ , including λ

Σ^+ denotes $\Sigma^* - \lambda$

The length of a string x , denoted by $|x|$ or $l(x)$, is the number of symbols in the string.

If x and y are strings, then the string xy is the concatenation of x and y . A string of i a 's is represented by a^i ; a^0 is λ .

A language L over alphabet Σ is any subset of Σ^* .

2.1.2 Procedure and algorithms

A procedure consists of a finite set of instructions which can be mechanically executed in a fixed amount of time. A procedure halts on a given input if there is a finite number t such that after executing t elementary instructions of the procedure, either there is no instruction to be executed next or a "halt" instruction was last executed.

An algorithm is a procedure which halts on all inputs.

Two main criteria are used for evaluating the performance of an algorithm:

(1) time complexity - the number of instructions executed as a function of the size of the input

(2) space complexity - the amount of memory required to hold intermediate results as a function of the size of the input.

2.1.3 Languages, grammars and recognizers.

A language which can be defined by a procedure is called recursively enumerable. A language defined by an algorithm is called recursive.

A grammar is a generative mathematical system for defining a language in a recursive manner. Formally, a grammar is a quadruple

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

where: (1) N is a finite set of nonterminal symbols (nonterminals)

(2) Σ is a finite set of terminal symbols (terminals)

(3) P is a finite set of rewriting rules or productions of the form $\alpha \rightarrow \beta$

$$, \text{ where } \alpha \in (N \cup \Sigma)^+, \beta \in (N \cup \Sigma)^*$$

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

A grammar is decidable if it generates a recursive language.

A second method of providing a finite specification for a language is to define a recognizer for that language - an automaton which accepts the strings in the language and rejects the strings not in the language.

Conventions of representation:

(1) terminals: $a, b, c, \dots, 0, 1, 2, \dots$

(2) nonterminals: $A, B, C, \dots$

(3) start symbol: P

- (4) nonterminals or terminals: $U, V, \dots, Z$
- (5) strings of nonterminals and terminals: $\alpha, \beta, \dots$
- (6) strings of terminals only: $u, v, \dots, z$

If $\alpha \rightarrow \beta$ is a production in P and γ and δ are strings, then the production $\alpha \rightarrow \beta$ may be applied to the string $\gamma\alpha\delta$ to obtain the string $\gamma\beta\delta$; this process is denoted $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$

For any grammar G the language $L(G)$ generated by G is defined by

$$L(G) = \{ w \mid S \xRightarrow{*} w \text{ and } w \in \Sigma^* \}$$

where $\xRightarrow{*}$ is the transitive closure of $\Rightarrow$

Grammars can be classified according to the format of their productions as follows:

- (1) regular grammars, if each production in P is of the form:

$$A \rightarrow aB \text{ or } A \rightarrow a$$

- (2) context free, if each production in P is of the form:

$$A \rightarrow \alpha$$

- (3) context - sensitive, if each production in P is of the

form

$$\alpha \rightarrow \beta \quad , \quad |\alpha| \leq |\beta|$$

In the preceding definitions:

$$a \in \Sigma; \quad A, B \in N; \quad \alpha, \beta \in (N \cup \Sigma)^*$$

The languages generated by these grammars are called:

- (1) regular, denoted $L(R)$
- (2) context-free, denoted $L(CF)$
- (3) context-sensitive, denoted $L(CS)$ and satisfy the

following relations:

$$L(R) \subseteq L(CF) \subseteq L(CS)$$

Most syntactic pattern recognition techniques have been implemented using regular and context - free grammars, for which the theoretical foundations are best developed. The following discussion is restricted to regular grammars, which are used in the thesis.

2.1.1.4 Properties of regular languages

Regular expressions (RE) A convenient way to describe regular languages is to use regular expressions, which are defined recursively as follows:

- (1) $\emptyset$ is a RE denoting the set $\emptyset$
- (2) λ is a RE denoting the set λ
- (3) a in Σ is a RE denoting the set $\{a\}$
- (4) if p and q are RE denoting the sets P and Q , respectively,

then

- a $(p + q)$ is a RE denoting $P \cup Q$
- b (pq) is a RE denoting PQ
- c (p^*) is a RE denoting P^*
- (5) nothing else is a RE

Finite automata (FA) Recognizers for regular languages, called finite automata, are defined as follows:

A nondeterministic FA is a quintuple

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

where: (1) Q is a finite set of states;

(2) Q denotes the number of states

- (2) Σ is a finite set of input symbols;
 $|\Sigma|$ is the number of input symbols
- (3) δ is a mapping from $Q \times \Sigma$ to $P(Q)$ called the state transition function,
- (4) q_0 in Q is the initial state
- (5) $F \subseteq Q$ is the set of final states

M is deterministic if $\delta(q, a)$ has no more than one member for any q in Q and a in Σ . If $\delta(q, a)$ always has exactly one member, M is called completely specified.

In certain problems it is convenient to use a modified type of FA, defined as follows:

A Moore-type finite automaton is a six-tuple FA = (X, Y, S, f, g, q_0) where

X is the finite input alphabet

Y is the finite output alphabet

S is the finite set of states

$f : S \times X \rightarrow S$ is the state transition function

$g : S \rightarrow Y$ is the output function

q_0 in S is the initial state

Two fundamental theorems in language theory state:

- (1) a language is accepted by a FA iff it is a regular language
- (2) the following problems are decidable for regular languages:
 - a. the membership problem : given a specification and a string, is the string in the specified language ?

b. the equivalence problem : given two specifications of the same type, do they specify the same language ?

The process in which strings are examined to determine whether they belong to a language is called syntax analysis or parsing.

It can be shown that the algorithm for solving the membership problem has:

(1) time complexity : a linear function of the length of the string.

(2) space complexity : proportional to the size of matrix (q, a) , that is $|Q| \cdot |\Sigma|$

The hardware implementation of FA is economical and can be operated at very high speed. These properties make regular languages an attractive solution for on-line pattern recognition problems, provided the descriptive power of these languages is satisfactory for a specific application.

Since the class of 'context-free languages properly includes the regular languages, it can be expected to encounter more difficulties in their theoretical investigation. Indeed, the time and space complexity of the syntax analysis algorithms for context-free languages is in general considerable greater than for regular languages [7]. This was the major factor in taking the decision to restrict the research work on regular grammars and languages for real-time applications.

2.2 Formal languages for pattern recognition

In conventional formal language theory, strings are built up by concatenating symbols. By analogy, two-dimensional patterns could be specified as being built up out of simpler subpatterns called primitives using a set of composition descriptors defined for a two-dimensional space; the pattern syntax analysis starts with the identification of the primitives and composition descriptors used to build up the input patterns [8], [9], [10], [11].

The fundamental difficulty encountered in extending formal language theory to two-dimensional patterns is due to the fact that an one-dimensional line has a natural ordering, whereas the plane does not have this property. The problem can be solved by introducing a more general formalism, which can employ the primitives and composition descriptors to derive a representation of two-dimensional patterns by a string - structured expression. In this way, the conventional formal language theory becomes applicable to the description and classification of two-dimensional patterns.

The general ideas presented in the foregoing discussion were implemented in the present case in the following sequence:

- 1) Selection of a set of pattern primitives and composition descriptors.

This problem is similar in nature to the feature selection problem in the decision - theoretic approach and is solved by the designer using his knowledge of the application and by intuition. The set of pattern primitives, denoted Σ and

the set of composition descriptors, denoted D , which represent the solution in general satisfy two elementary conditions:

a) they should permit a structural description of a class of patterns

b) a simple extraction using non-syntactic methods should be possible.

2) Development of a convenient formalism for the syntactic description of the patterns.

There is no standard formalism for defining two-dimensional patterns in terms of primitives and composition descriptors. The obvious condition imposed is that the formalism chosen permits a string representation of the patterns after extracting the primitives and composition descriptors.

Using this formalism, the set of primitives and composition descriptors, a pattern grammar can be defined formally as the quintuple:

$$G_p = (\Sigma, D, N, P, S)$$

where Σ - finite set of pattern primitives

D - finite set of composition descriptors

N - finite set of subpatterns

P - finite set of productions, expressed using the formalism F :

$$A \rightarrow F(d, C_1, C_2); \quad A \in N, d \in D,$$

$$C_1 \text{ and } C_2 \in \Sigma \cup N$$

S - start primitive

Thus, the production $A \rightarrow F(d, C_1, C_2)$ defines a subpattern

whose constituents are the primitives or subpatterns C_1 and C_2 related by composition descriptor d .

3) Inference of the pattern grammar

Having specified the formalism F and the sets Σ and D , the next step is to derive the grammar G which describes the class of patterns using a training set of patterns,

$$T = T^+ \cup T^-; \quad T^+ \in L(G), \quad T^- \notin L(G)$$

4) Pattern recognition using syntax analysis

Consider a n -class pattern recognition problem and assume it was possible to construct a grammar G_i for each class of patterns. In an ideal situation, the languages $L(G_i)$ define classes of patterns which are disjoint:

$$\bigcap_{i=1}^n L(G_i) = \emptyset$$

Given an input pattern described by a string x , the recognition problem can be stated as follows:

Is $x \in L(G_i)$ for $i = 1, 2, \dots, n$?

The string x will be rejected if

$$x \notin \bigcup_{i=1}^n L(G_i)$$

In all the other cases, a solution of the following form can be found:

$$x \in L(G_k); \quad 1 \leq k \leq n$$

$$x \notin \bigcup_{i=1}^n L(G_i)$$

$$i = 1$$

$$i \neq k$$

hence the pattern represented by string x is in class k .

The following section will illustrate how the theoretical principles described can be applied in a practical pattern recognition system.

2.3 General structure of a syntactic pattern recognition system [8]

A syntactic pattern recognition system can be represented by a block diagram which consists of the following major parts (fig. 2.1):

(1) preprocessing : can be considered as an interface between the input pattern and the syntactic pattern recognition system (for ex: detection/delineation)

(2) pattern description : the syntactic primitives and the composition descriptors are extracted from the pattern.

(3) syntax analysis : the syntactic representation of the pattern is analyzed (parsed) in order to find the language to which it belongs, hence the class of patterns.

(4) grammatical inference : knowledge of the structure of sample patterns is applied to infer grammars for each class of patterns, in the learning stage, using a training set of patterns.

The inference of the pattern grammars is carried out off-line, by the designer or a programmed algorithm. Stages (1), (2) and (3) can be simulated (off-line) for designing the system, but finally must be implemented on-line and in real time in applications like defect classification.

Preprocessing is usually not considered a proper part of a syntactic pattern classifier. This applies also to the present

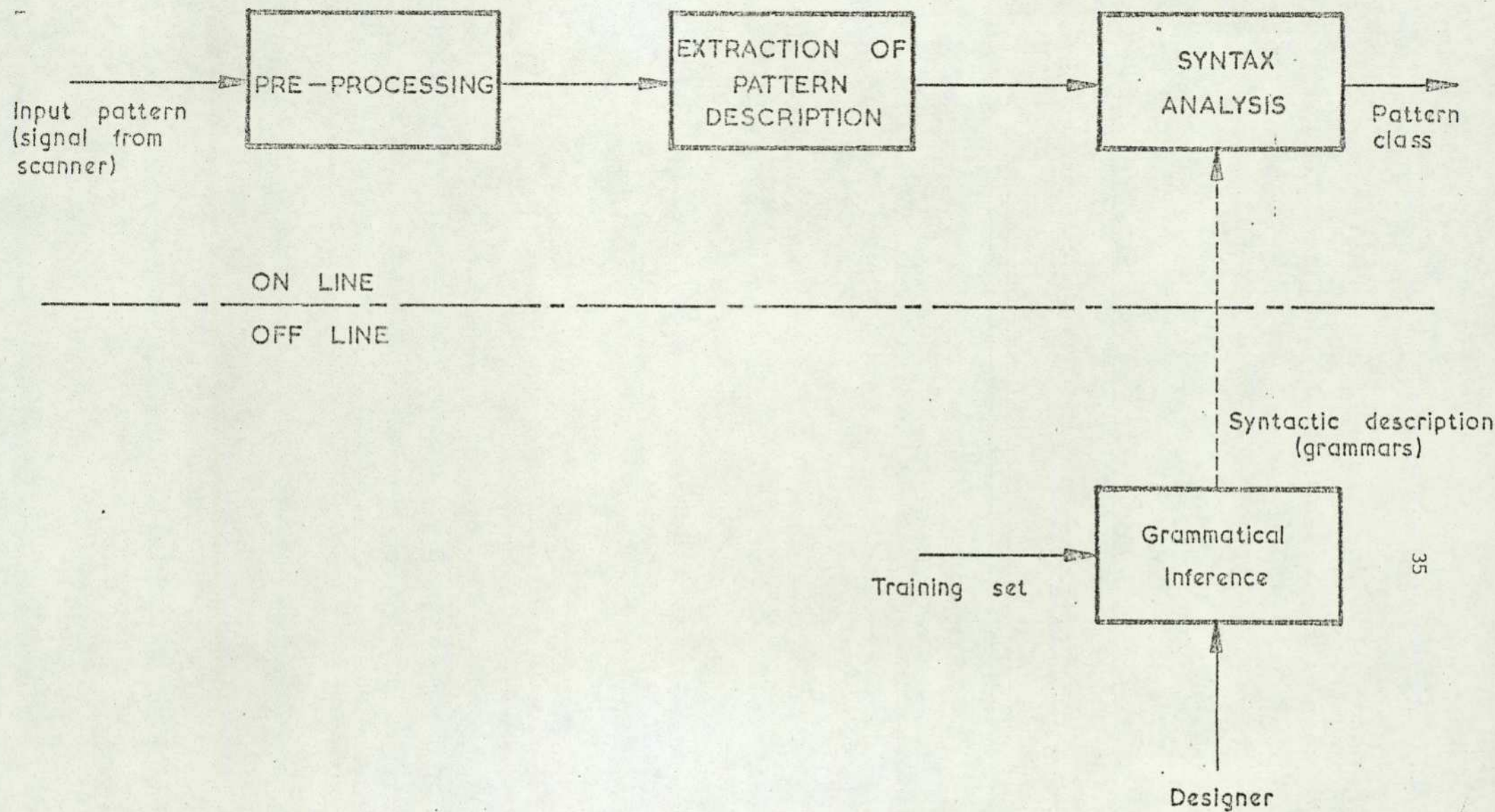


Fig. 2.1 Block diagram of a pattern recognition system for "on-line" applications

case of the automatic surface inspection system, in which detection and delineation are common functions, which must be implemented before any method of classification can be attempted.

A large variety of pattern description and classification techniques have been developed for performing stage 3 and stage 4 of a syntactic pattern recognition system. A critical review of these approaches, with the intention of selecting the most suitable techniques for the requirements of on-line defect classification, will be presented in the next chapter. The review has been oriented according to the conclusions of the preliminary experiment presented in para. 1.2.

3. REVIEW OF PREVIOUS WORK IN SYNTACTIC RECOGNITION OF
IMPERFECTLY STRUCTURED PATTERNS.

In the preceding chapter, the fundamental concepts of syntactic pattern recognition have been introduced and the main problems encountered in a practical application have been formulated. This chapter reviews the work in this area which could provide ideas or techniques applicable to an on-line defect classification problem.

Previous work in the selection of pattern primitives and composition descriptors and different formalisms proposed for representing two-dimensional patterns are not discussed because of their application - oriented character. The review concentrates on two major problems : syntax analysis of imperfectly formed patterns and pattern grammar inference. The main directions of development will be introduced as briefly as possible, in order to support the criteria used for rejecting or accepting a specific technique. Basically, this review is intended to justify the approach taken in this thesis : 1) approximate description of the defects using regular grammars and 2) classification based on the concept of syntactic distances, which reduces the errors due to language approximation and to pattern variability.

Formal language theory was originally developed to describe strings of symbols which were perfectly structured and until recently no provisions had been made to handle strings distorted or corrupted by noise. Consequently, syntactic pattern recognition methods based on concepts proposed for ideal situations offer a poor performance

when applied to imperfectly-formed patterns.

The effects of distortions and noise can be reduced using extensive preprocessing in order to present well-formed patterns at the input of the syntactic classifier. Although preprocessing does improve the recognition performance, it does not solve completely the problem, for two reasons:

- 1) preprocessing requires a considerable amount of computation, which could become prohibitive in an on-line application.
- 2) in complex situations the overall performance of the syntactic classifier is still unsatisfactory.

The only way left to improve the performance of a syntactic classifier is to develop more sophisticated techniques in formal language theory, to incorporate features which permit handling of imperfect patterns. The methods which have been recently proposed to tackle this type of applications can be divided into three classes:

- 1) stochastic grammars and languages
- 2) fuzzy grammars and languages
- 3) error - correcting techniques

These methods will be reviewed briefly in the following sections and evaluated according to the requirements mentioned in par. 1.3.2.

Finally, a brief discussion of the literature on grammatical inference will present the fundamental results obtained in this field and then select specific techniques applicable to the present problem.

3.1 Stochastic grammars and stochastic languages [12] - [14]

Stochastic languages have been proposed for modelling more realistically applications in which patterns must be characterized taking into account randomness introduced by measurement noise and ambiguities due to intersecting classes of languages.

The concepts of formal language theory can be extended to incorporate statistic considerations in two ways:

a) by assigning probabilities to the productions of the grammar, thus defining a stochastic grammar

b) by assigning probabilities to the state transitions of the acceptor, therefore introducing stochastic automata.

The following discussion will be restricted to stochastic regular languages and stochastic finite - state automata.

Definition 3.1 A stochastic grammar is a quadruple

$G_S = (N, \Sigma, P_S, S)$ where :

(1) N is a finite set of nonterminals

(2) Σ is a finite set of terminals

(3) $S \in \Sigma$ is the start symbol

(4) P_S is a finite set of stochastic productions, each of

which is of the form

$$\alpha_i \xrightarrow{p_{ij}} \beta_{ij}; \quad j = 1, 2, \dots, n \text{ and } i = 1, 2, \dots, k \quad (3.1)$$

where

$$\alpha_i \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$$

$$\beta_{ij} \in (N \cup \Sigma)^*$$

and p_{ij} is the probability associated with the production:

$$0 < p_{ij} < 1, \quad \sum_{j=1}^{n_i} p_{ij} = 1 \quad (3.2)$$

Definition 3.2 The probability p associated with a derivation $\xi \xRightarrow{*} \eta$

is equal to the product of the probabilities associated to the sequence of n productions used in the derivation:

$$p = \prod_{i=1}^n p_i \quad (3.3)$$

The stochastic language generated by G_S is

$$L(G_S) = \left\{ (x, p(x)) \mid x \in \Sigma^*, S \xRightarrow[p_*]{p_j} x ; \right. \\ \left. j = 1, 2, \dots, k \text{ and } p(x) = \sum_{j=1}^k p_j \right\} \quad (3.4)$$

where k is the number of all different derivations of x from S and p_j is the probability associated with the j -th distinct derivation of x .

Definition 3.3 A stochastic regular grammar

$G_S = (N, \Sigma, P_S, S)$ where

- (1) $\Sigma = \{ a_1, a_2, \dots, a_m \}$
- (2) $N = \{ A_1, A_2, \dots, A_k \}$
- (3) $S = \{ A_1 \}$

is a stochastic grammar in which the productions are of the form

$$\begin{array}{l} A_i \xrightarrow[p_{i1}]{1} a_1 A_1 \\ A_i \xrightarrow[p_{i2}]{2} a_2 A_2 \\ \vdots \\ A_i \xrightarrow[p_{ij}]{j} a_j A_j \\ \vdots \\ A_i \xrightarrow[p_{im}]{m} a_m A_m \\ A_i \xrightarrow[p_{ik}]{k} a_k A_k \end{array} \quad \begin{array}{l} A_i \xrightarrow[p_{i0}]{1} a_1 \\ \vdots \\ A_i \xrightarrow[p_{i0}]{1} a_\ell \\ \vdots \\ A_i \xrightarrow[p_{i0}]{m} a_m \\ A_i \xrightarrow[p_{i0}]{k} a_k \end{array} \quad (3.5)$$

(some probabilities may be equal to zero)

$$\sum_{j=1}^k \sum_{l=1}^m p_{ij}^l + \sum_{l=1}^m p_{i0}^l = 1 \quad (3.6)$$

Definition 3.4 A stochastic finite-state automaton is a five-tuple $A_s = (\Sigma, Q, M, \Pi_0, F)$ where:

- (1) Σ is a finite set of input symbols
- (2) Q is a finite set of internal states; $|Q| = n$
- (3) M is a mapping of Σ into the set of $n \times n$

stochastic transition matrices; for $a \in \Sigma$,

$$M(a) = [p_{ij}(a)]$$

where $p_{ij}(a)$ is the probability of the input a forcing a transition from state q_i to state q_j

$$\sum_{j=1}^n p_{ij} = 1, \quad i = 1, 2, \dots, n$$

(4) Π_0 is a n -dimensional row vector defining the initial state distribution

- (5) $F \subseteq Q$ is a finite set of final states.

The preceding basic definitions are sufficient to discuss some properties of stochastic languages and stochastic automata in connection with on-line defect classification.

1) The descriptive power of stochastic regular grammars is the same as that of their nonstochastic version, since the productions are similar in form, except for the assignment of probabilities.

2) For a given internal state and input symbol of a stochastic finite-state automaton, there is in general more than one possible transition defined in the matrix M . This fundamental property of a stochastic automaton extends considerably its recognition possibilities, but is also very difficult to implement directly in hardware.

3) The recognizer assigns nonzero probabilities to strings belonging to the language $L(G_s)$, but if only one symbol in these strings is corrupted and the new string does not belong to $L(G_s)$, it will be assigned probability zero; this is a major disadvantage in extrapolating from a training set to a new set of patterns.

The limitations of the stochastic language discussed above show that, although improving the performance of a syntactic recognizer in cases where ambiguity is due to overlapping classes, this method is not completely satisfactory in more complicated situations.

3.2 Fuzzy grammars and fuzzy languages. [15] - [17]

The concept of fuzzy sets was proposed by L.A. Zadeh in order to handle mathematically situations in which ambiguity is due to uncertainty about the structure of the sets rather than randomness. Results from the theory of fuzzy sets can be extended to formal language theory, which is also formulated in terms of sets. This also seems to apply well to syntactic pattern recognition, especially when the patterns are imperfectly formed from the beginning, though still retain a common underlying structure within a pattern class.

Let U denote the set of all elements in a class ("universe of discourse")

Definition 3.5 A fuzzy set A in U is identified by a membership function $\mu_A : U \rightarrow [0, 1]$

defined such that for any element $x \in U$, $\mu_A(x)$ is a number in the interval $[0, 1]$ indicating the degree of membership of x in A .

The membership function can be defined in a tabular form if U is a finite set. In practice, U is usually infinite or too large and therefore an algorithm for computing the membership function is required.

The definition of a grammar can be extended to fuzzy grammars by associating with each production a weight θ , $\theta \in [0, 1]$. Similarly, the definition of acceptors can be generalized by assigning a weight $\theta \in [0, 1]$ to each transition. These ideas will be illustrated for regular fuzzy grammars and finite-state fuzzy automata.

Definition 3.6 A regular fuzzy grammar is defined by a quadruple $G_F = (\Sigma, N, S, P)$ where:

- (1) Σ is a finite state of nonterminals
- (2) N is a finite state of terminals
- (3) S is the starting symbol
- (4) P is a finite set of productions of the form

$$\begin{array}{l} \theta \\ A \longrightarrow a B \\ \text{or} \quad \theta \\ A \longrightarrow a \end{array}$$

where $A, B \in N$; $a \in \Sigma$ and $\theta \in [0, 1]$

Definition 3.7 A finite-state fuzzy automaton is defined by a quintuple

$$A_F = (Q, \Sigma, \delta, q_0, F), \text{ where:}$$

- (1) Q is a finite set of internal states
- (2) Σ is a finite set of input symbols
- (3) δ is the fuzzy state transition map:

$$\delta: \Sigma \times Q \times [0, 1] \rightarrow Q$$

- (4) $q_0 \in Q$ is the initial state
- (5) $F \subseteq Q$ is the set of final states

Definition 3.8 Given a regular fuzzy grammar G_F , the membership grade of a string $x \in L(G_F)$ is the maximum value of any derivation of x , where the value of a specific derivation of x is equal to the minimum weight of the productions used.

The classification of strings belonging to regular fuzzy languages is simple to implement in hardware due to the following property [6]:

Theorem 3.1 Given a regular fuzzy grammar G_F , there exists a deterministic more finite-state automaton A_D with a set of final states F which computes the membership function $\mu : \Sigma^* \rightarrow [0,1]$ of the language $L(G_F)$

A direct analysis of the definitions and properties given for regular fuzzy languages and automata produces useful answers to the problem of applying this method to defects classification:

- 1) provided the regular fuzzy grammars are known, the corresponding classifier is simple to implement in hardware
- 2) there are no general methods for defining and computing membership functions; except for very simple or specially selected cases, this disadvantage drastically reduces the applicability of this technique to complex situations.

It will be shown in chapter 6 that some of the attractive features of fuzzy languages can be implemented using the concept of syntactic distances.

3.3 Error correcting techniques [18] - [22]

Error correcting techniques had been used originally in the field of compiler error recovery by applying edit operations in order to transform strings. Recently, a similar method has been proposed to classify imperfectly formed patterns for which stochastic or fuzzy languages are not adequate to apply because the distortions are too complex.

This method will be examined closer because it seems to be more powerful in handling imperfectly formed patterns and also because the technique finally developed in this thesis is related to the basic ideas of error correcting techniques. Although the method has been developed for both regular and context free languages, it will be introduced for regular languages.

Three types of corrupting errors in the syntactic description of a class of strings will be considered:

- 1) an α error, denotes deleting one symbol
- 2) a β error, denotes inserting one symbol
- 3) a δ error, denotes changing one symbol to any other symbol.

In this section, a method will be presented for deriving the regular expressions which define strings corrupted by these three types of errors, when knowing the regular expressions of the original strings; the discussion is based entirely on results obtained in [18].

Consider now a Moore type finite-state automaton $FSA = (S, Y, S, f, g, q_0)$. This FSA is defined by a set of regular expressions $\{R_j ; j \text{ in } A\}$, where R_j denotes the set of strings in X^* which cause the FSA to halt in a state with output j .

Given a string y in R_j , the following operators are introduced in order to deal with the types of errors defined before:

$$\alpha(y) = \{ x : x \text{ in } X^*, x \text{ obtained from } y \text{ corrupted by an } \alpha \text{ error} \} \quad (3.7)$$

$$\beta(y) = \{ x ; x \text{ in } X^*, x \text{ obtained from } y \text{ corrupted by an } \beta \text{ error} \} \quad (3.8)$$

$$\delta(y) = \{ x ; x \text{ in } X^*, x \text{ obtained from } y \text{ corrupted by a } \delta \text{ error} \} \quad (3.9)$$

These operators can be extended to R_j :

$$\alpha(R_j) = \{ x : x \text{ in } \alpha(y), y \text{ in } R_j \} \quad (3.10)$$

$$\beta(R_j) = \{ x : x \text{ in } \beta(y), y \text{ in } R_j \} \quad (3.11)$$

$$\delta(R_j) = \{ x ; x \text{ in } \delta(y), y \text{ in } R_j \} \quad (3.12)$$

Let $a \in X$ and let R be a regular expression defining all strings of length 1 in X^* ; then from (3.7) - (3.9) follows:

$$\alpha(a) = \lambda ; \quad \alpha(\lambda) = \emptyset ; \quad \alpha(\emptyset) = \emptyset \quad (3.13)$$

$$\beta(a) = Ra + aR ; \quad \beta(\lambda) = R ; \quad \beta(\emptyset) = \emptyset \quad (3.14)$$

$$\delta(a) = R \cap \bar{a} ; \quad \delta(\lambda) = \emptyset ; \quad \delta(\emptyset) = \emptyset \quad (3.15)$$

The results of applying the error operators to the union, concatenation and closure of regular expressions P and Q are the defined by the identities (proof given in [18]):

1) Union

$$\alpha(P + Q) = \alpha(P) + \alpha(Q) \quad (3.16)$$

$$\beta(P + Q) = \beta(P) + \beta(Q) \quad (3.17)$$

$$\delta(P + Q) = \delta(P) + \delta(Q) \quad (3.18)$$

2) Concatenation

$$\alpha(P.Q) = \alpha(P)Q + P.\alpha(Q) \quad (3.19)$$

$$\beta(P.Q) = \beta(P)Q + P.\beta(Q) \quad (3.20)$$

$$\delta(P.Q) = \delta(P).Q + P.\delta(Q) \quad (3.21)$$

3) Closure

$$\alpha(P_*) = P_* \alpha(P) P_* \quad (3.22)$$

$$\beta(P_*) = P_* \beta(P) P_* + R \quad (3.23)$$

$$\delta(P_*) = P_* \delta(P) P_* \quad (3.24)$$

Given the regular expression for a class of perfect strings, the regular expression defining the error - corrupted strings is determined by applying first one of the identities (3.16) - (3.24) and then of one of the identities (3.13) - (3.15).

If the probabilities of errors and the frequencies with which input strings occur are known, the regular expressions defining the error - corrupted strings can be used to design a stochastic finite - state automaton using Bayes' Rule. The transition matrix of the automaton will be deterministic, but its output function g will be stochastic.

The procedure can be extended to handle situations in which multiple errors can occur. Because the error operators are not commutative, the regular expressions defining strings corrupted by multiple errors must be derived separately, for each case. Application of the Bayes' Rule to determine the stochastic output function also becomes complicated in this case.

In principle, the method presented in this section solves completely, from a theoretical point of view, the problem of classifying error-corrupted regular strings. The resulting classifier can be implemented as a deterministic FSA with a stochastic output function. The final success of a practical application largely depends on developing efficient computation algorithms for solving

the following problems for the realistic situation of multiple errors:

- 1) estimation of the probabilities of each type of error using a training set
- 2) deriving regular expressions for strings corrupted by multiple errors
- 3) finding the stochastic output function of the FSA when the probabilities of multiple errors can not be ignored.

Although these problems seem to be theoretically solvable, they require considerable development effort. For this reason, the investigation was continued to find a simpler way to design a classifier based on similar principles. The resulting technique, which was finally adopted, is described in chapter 6.

3.4 Grammatical inference

The problem of learning decision functions from a training set of patterns has been studied extensively within the framework of the decision - theoretic method and efficient algorithms which provide solutions completely automatically are well known. By contrast, the grammatical inference (GI) problem has received attention only recently and most results refer to string grammars, little work is published in the area of high-dimensional GI.

Solving the GI problem represents an important prerequisite to the acceptance of syntactic pattern recognition as a general pattern recognition method. A satisfactory solution does not necessarily require a fully automatic procedure, because using the designer's knowledge of the problem to achieve dimensionality reduction is an attractive feature of the syntactic method. The emphasis in previous research on completely automatic procedures

explains the limited degree of success achieved in this area. An interactive procedure, in which the designer provides highly relevant information and the computer carries out routine operations would represent an ideal solution.

3.4.1 Problem statement [23], [24]

The GI problem can be stated as the process of choosing a "good" grammar for a language, as an extrapolation from a finite set of samples from the language. The problem is "well-defined" if it is stated in terms of:

- (1) a hypothesis space - the class of grammars to be inferred (ex: context free)
- (2) an observation space - the training set
- (3) an evaluation measure - an objective evaluation of the accuracy of the solution (ex: performance of the corresponding recognizer).
- (4) a complexity measure of the grammar (ex: the number of productions and the number of terminal and nonterminals in each production).

3.4.2 Basic definitions

Definition 3.9 An information sequence or informant presentation of a language L , denoted $I(L)$, is a sequence of strings from the set $\{+y \mid y \in L\} \cup \{-y \mid y \in \Sigma^* - L\}$. A positive information sequence or text presentation of a language L , denoted by $I^+(L)$, is a sequence containing only strings from L . A negative information sequence, denoted by $I^-(L)$, is an information sequence of L containing only strings from $\Sigma^* - L$.

Definition 3.10 An information sequence is complete if

- a. $I^+(L)$ contains all strings from L
- b. $I^-(L)$ contains all strings not in L

Definition 3.11 A sample is a subset of a complete information sequence. A positive sample of a language $L(G)$ is said to be structurally complete, denoted $S(L)$ if each production defined in G is used in the generation of at least one string in $I^+(L)$.

Definition 3.12 A grammar G_t is compatible with a sample $S_t(L)$ if $L(G_t)$ contains all strings in the positive sample and none of the strings in the negative sample.

Definition 3.13 A class of grammars C is said to be admissible if

- a) C is denumerable
- b) for $x \in \Sigma^*$, it is decidable whether or not $x \in L(G)$ for any $G \in C$.

Definition 3.14 A class of languages $L(G)$ is said to be identifiable in the limit if there is an inference algorithm D such that for any $G \in C$ and any complete information sequence $I(L(G))$ there exists a τ such that

$$a) G_t = G_\tau \quad \text{for } t > \tau$$

where $G_t = D(S_t, C)$ and $G_\tau = D(S_\tau, C)$

$$b) L(G_\tau) = L(G)$$

3.4.3 Fundamental results in grammatical inference [25], [28]

Theorem 3.2 Let C be an admissible class of grammars.

There is an algorithm such that for any $G \in C$, $L(G)$ is identifiable in the limit through an information presentation.

Theorem 3.3 Let C be an admissible class of grammars which generates all finite languages and any one infinite language L . Then L is not identifiable in the limit through a text presentation.

Theorem 3.4 Let C be a class of regular languages. C is not identifiable in the limit through a text presentation.

3.4.4 Approximate language identification [33]

Previous work in GI has been implicitly concerned with the problem of exact language identification; this is only possible if such a grammar exists in the solution space. In the more general case, where this assumption cannot be made, the best solution is a grammar whose language approximates the unidentified language with a satisfactory degree of accuracy.

The concept of approximate language identification represents a major progress in this area but will not be discussed formally because it was used implicitly rather than explicitly in this thesis. In principle, this concept supports the idea of approximating the description of defect patterns by regular languages since the emphasis is on a satisfactory recognition performance rather than on accurate description. The specific implementation of this approach is discussed in chapter 5.

3.4.5 Review of grammatical inference techniques

The GI algorithms described in the literature can be classified according to various criteria:

- 1) class of grammars
 - regular [32]
 - context free (restricted) [26], [27]

2) types of grammars

- deterministic [32], [27]
- stochastic [26], [29], [30]

3) dimensionality of patterns:

- strings : most of the work
- two-dimensional [24]

4) procedures employed:

- enumerative [25]
- inductive [32]
- heuristic [31], [24]
- interactive [23]

The inference of defect grammars requires an algorithm which can derive deterministic two-dimensional regular grammars; enumerative procedures should be avoided because of the astronomical complexity for a large training set; positive and negative samples are available and an approximate solution is acceptable.

Although none of the algorithms proposed in the literature completely satisfied these conditions, a suitable combination of techniques could lead to an acceptable solution. The method which has been developed is described in detail in chapter 5 and combines the algorithms described in [32] and [24].

4. SYNTACTIC REPRESENTATION OF DEFECTS

Selecting a pattern description formalism is the first major task in the development of a syntactic pattern recognition system. The formalism described in this chapter represents an original solution which has been developed by the author to satisfy the specific requirements of the present problem:

- a) provides a satisfactory description of the two-dimensional structure of the defects
- b) can be extracted at the high speed imposed by real-time conditions, using simple, standard digital integrated circuits.

4.1 Selection of pattern primitives and composition descriptors

Since the system which extracts the representation of the defects interacts with the delineation system, the primitives and descriptors have been defined taking into consideration that the delineation of the defects is performed on a scan by scan basis.

This method is well adapted to the operation of the inspection system and suggests the selection of the pattern primitives and composition descriptors in terms of the information contained in successive scan sections of the same defect. Although in this way the degrees of freedom in selecting the pattern primitives and composition descriptors are restricted, this solution was adopted because it simplified the extraction of the syntactic representation of defects and can be implemented at high speed, an essential requirement for an on-line application.

Comments included in the programs use the shorter terms "symbols" and "predicates" for pattern primitives and composition descriptors, respectively. In the text, the short-term "syntactic element" will refer to a pattern primitive / composition descriptor pair assigned to the same scan section.

4.1.1 Pattern primitives

Scan sections (SS) consist of one or more segments (SEG) of 1-trigger samples; both were defined using the following regular expressions:

$$\text{SEG} = 11_*00_* \quad (4.1)$$

$$\text{SS} = \text{SEG } 00_*(\text{SEG } 00_*)_* \quad (4.2)$$

After analysing the structure of the defects, it was decided to define the pattern primitives associated to scan sections in terms of two parameters:

- a) the number of segments included in the scan section (N_S)
- b) the length of these segments (L_S)

It is evident that the pattern primitives as defined describe the properties of the patterns in the transversal direction; further, the number of different pattern primitives which have to be defined is problem dependent. To a certain extent, it is possible to describe quantitatively some geometrical properties, by an appropriate selection of the pattern primitives, using two sets of thresholds, defined in the following sections.

Consider a first set of integers $N_1, N_2, \dots, N_m$, such that $N_i < N_{i+1}$ for $i = 1, 2, \dots, m-1$, and let N_S denote the actual number of segments in a scan section. The parameter

describing the number of segments in a scan section, $p_N(N_S)$, was defined as follows:

$$p_N(N_S) = \begin{cases} 1 & \text{if } N_S < N_1 \\ i + 1 & \text{if } N_i \leq N_S \leq N_{i+1} \text{ for } 1 \leq i \leq m-1 \\ m + 1 & \text{if } N_S > N_m \end{cases} \quad (4.3)$$

A similar method has been employed to define the second parameter of a pattern primitive, by selecting a set of integers $L_1, L_2, \dots, L_n$. If $N_S = 1$ and the length of that segment is L_S , the value of the parameter $p_L(L_S)$ is defined as follows:

$$p_L(L_S) = \begin{cases} 1 & \text{if } L_S < L_1 \\ i + 1 & \text{if } L_i \leq L_S < L_{i+1} \text{ for } 1 \leq i \leq n-1 \\ n + 1 & \text{if } L_S \geq L_n \end{cases} \quad (4.4)$$

If $N_S > 1$, there are several ways to characterise the length of these segments, the simplest being to select the minimum or maximum length and to assign an unique value to the parameter p_L by substituting in the preceding formulae

$$L_S = \min_{N_S} L_{S_i} \quad (4.5a)$$

$$\text{or } L_S = \max_{N_S} L_{S_i} \quad (4.5b)$$

Consequently, by selecting a number of m and n thresholds for the parameters p_N and p_L , respectively, one can define a number of $(m + 1)(n + 1)$ different pattern primitives which will characterise the scan sections. The specific values of these parameters were selected following the procedure discussed in para 4.1.3 and are given in table 2.

TABLE 2.

DEFINITION OF PATTERN PRIMITIVES1. Definition of Parameter $p_N(N_S)$

$$N_1 = 1, N_2 = 2, N_3 = 3$$

$$p_N(N_S) = \begin{cases} 1 & \text{if } N_S = 0 \\ 2 & \text{if } N_S = 1 \\ 3 & \text{if } N_S = 2 \\ 4 & \text{if } N_S \geq 3 \end{cases}$$

2. Definition of parameter $p_L(L_S)$

$$L_1 = 12, L_2 = 25$$

$$\text{if } N_S = 1:$$

$$p_N(L_S) = \begin{cases} 1 & \text{if } L_S < 12 \\ 2 & \text{if } 12 \leq L_S < 25 \\ 3 & \text{if } 25 \leq L_S \end{cases}$$

$$\text{if } N_S > 1, L_S = \min_{N_S} (L_{S_i})$$

3. Definitions of pattern primitives (P)

$p_N(N_S)$	$p_L(L_S)$	P
1	-	1
2	1	2
2	2	3
2	3	4
3	1	5
3	2	5
3	3	6
4	1	7
4	2	7
4	3	7

Note: P = 1 when the defect described is not encountered in the current scan.

4.1.2 Composition descriptors

After analysing the structure of the defects, the composition descriptors were selected to characterise the connectivity relationships which exist between the current scan section and the preceding scan section belonging to the same defect, in terms of two parameters:

- a) The degree of connectivity, defined as

$$C_S = \frac{N_C}{N_d} \quad (4.6)$$

where N_C - the number of common triggers

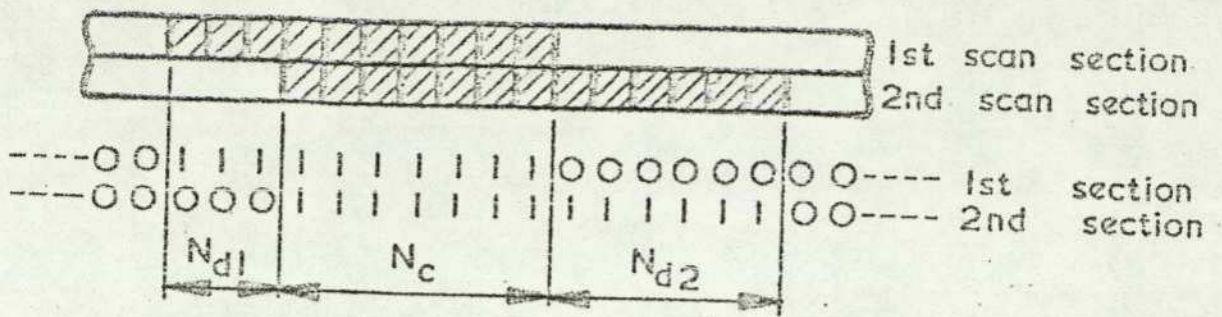
N_d - the number of disjoint triggers, as illustrated in fig. 4.1.

This parameter was then mapped to a set of integers by defining a set of thresholds $C_1, C_2, \dots, C_p$ as shown in the preceding paragraph:

$$d_C(C_S) = \begin{cases} 1 & \text{if } C_S < C_1 \\ i + 1 & \text{if } C_i \leq C_S < C_{i+1} \text{ for } 1 \leq i \leq p-1 \\ p + 1 & \text{if } C_S > C_p \end{cases} \quad (4.7)$$

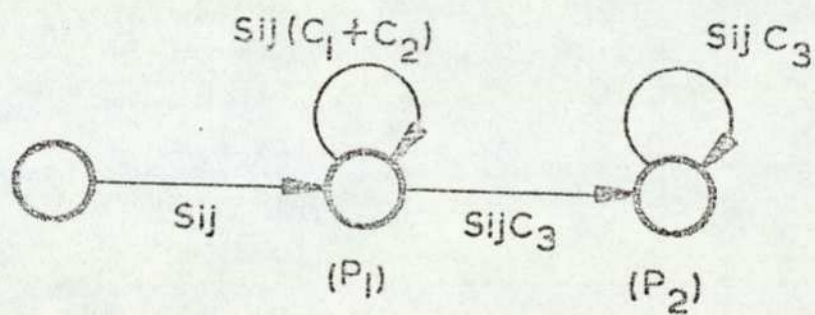
b) The type of connectivity (C_T), which specifies the position of the current scan section considering as reference the preceding scan section, belonging to the same defect. This parameter can be expressed in a simple manner if the number of segments in the two scan sections is less or equal to two, but becomes complicated and difficult to interpret for more than three segments in a scan section.

The composition descriptors as introduced characterise the properties of the patterns in the longitudinal direction; table 3 defines the composition descriptors used in the thesis.



$$C_s = \frac{N_c}{N_{d1} + N_{d2}}$$

Fig. 4.1 Definition of the degree of connectivity



$$C_1 = 1 \quad \text{iff } m < k$$

$$C_2 = 1 \quad \text{iff } m = k$$

$$C_3 = 1 \quad \text{iff } m > k$$

C_i - binary variables

Fig. 4.2 FSA Transition matrix

TABLE 3. DEFINITION OF COMPOSITION DESCRIPTORS1. Definition of connectivity degree

$$C_1 = 0.5, C_2 = 2$$

$$d_S(C_S) = \begin{cases} 1 & \text{if } C_S < C_1 \\ 2 & \text{if } C_1 \leq C_S < C_2 \\ 3 & \text{if } C_2 \leq C_S \end{cases}$$

2. Definition of connectivity type (T_C): refers to situations in which the defect described is not encountered in one or both consecutive scans ($i, i - 1$)

P_{i-1}	P_i	T_C
1	1	1
1	1	2
1	1	3
1	1	4

3. Definitions of composition descriptors (D)

T_C	$d_C(C_S)$	D
1	1	1
2	1	2
3	1	3
4	1	4
4	2	5
4	3	6

4.1.3 Evaluation of the separation capabilities of the pattern primitives and composition descriptors

The selection of pattern primitives and composition descriptors is in general based largely on intuition, as is the selection of features in the feature space method. This is an important and difficult problem; the pragmatic approach adopted in this case was to select a set of primitives and descriptors and to compute their distribution over the pattern classes, in order to evaluate their separation capabilities. However, it is essential to bear in mind that similar primitives and descriptors can define patterns with a completely different structure, if the productions of the grammars are different.

4.2 The formalism for describing two-dimensional patterns as strings of pattern primitives and composition descriptors.

In the present case, the general expression which was introduced in par. 2.2 has been rewritten in a simpler form, by taking into account two special properties of the selected pattern description language:

- 1) it is restricted to the class of regular languages
- 2) the composition descriptors are defined between two adjacent scan sections.

Consequently, it was decided to write the productions of the pattern grammars in the following form:

$$A \rightarrow (a, d) B \quad (4.8a)$$

$$\text{or} \quad A \rightarrow (a, d) \quad (4.8b)$$

where: A, B - (nonterminal) subpatterns

a - pattern primitive associated to a scan section

d - composition descriptor, specifying how the pattern primitive a is connected to the pattern primitive associated to the previous scan belonging to the same defect.

A two-dimensional pattern which extends over n scans will then be represented by a string of pattern primitives and composition descriptors of the form:

$$P = a_1 d_1, a_2 d_2, a_3 d_3, \dots, a_n d_n \quad (4.9)$$

In a strictly formal sense, both a_i and d_i can be regarded as symbols; by assigning a symbol s_i to each distinct pair $(a_i d_i)$, the representation of the pattern is formally encoded as a string of symbols:

$$P = s_1 s_2 \dots s_n \quad (4.10)$$

Hence all the syntactic classification methods developed for strings can be applied to two-dimensional patterns.

Quantitative geometric properties in the longitudinal direction can be described to some extent, if necessary, by adding a counter - comparator device to the finite - state automaton, thus enhancing the capabilities of a regular grammar. This necessity occurs when two patterns belonging to different classes can be described by similar pattern primitive / composition descriptor pairs, but the corresponding strings have different lengths.

A simple example illustrates this idea, in connection with the state transition graph shown in fig. 4.2:

$$P_1 = (a_i d_j)^m = (s_{ij})^m \text{ where } m \leq k \quad (4.11a)$$

$$P_2 = (a_i d_j)^n = (s_{ij})^n \text{ where } n > k \quad (4.11b)$$

As an illustration of the formalism introduced in this paragraph, the syntactic description of the defects encountered in the videoprint shown in fig. 4.3 is listed in table 4. The tails of 1/1 syntactic elements originate from the operation mode of the capture rectangle technique and are interpreted as "no scan sections".

4.3 Extraction of the syntactic representation of defects

The present paragraph describes the technique which has been developed for extracting on-line the representation of the defects; the structure of the designed block, denoted EXSYN, is outlined in fig. 4.4.

The main function of this block is to assign a pattern primitive and a composition descriptor to each scan section, by processing the trigger samples generated by the scanner. The operation of the block takes place under the control of the delineation block [3], which provides the following signals and data at its output:

1. beginning of a new scan section (BSS)
2. end of the current scan section (ESS)
3. the number of the defect with which the current scan section must be associated (ND)
4. a signal which indicates that two originally distinct defects have become so close that they have had to be merged (EQ)
5. an "end of scan" signal (EOS)

The pattern primitives are extracted by processing the trigger samples of the current scan only, according to rel. (4.1) - (4.5); the unit which implements this function (SYMEX) is initiated

by the signal BSS and presents the code of the pattern primitive associated to that scan section at its output, where it is sampled by the ESS signal.

The composition descriptors are extracted in the PREDEX unit, by processing the trigger samples of both the current and previous scan sections belonging to the same defect (which must therefore be stored in a shift register), according to rel. (4.6) - (4.7).

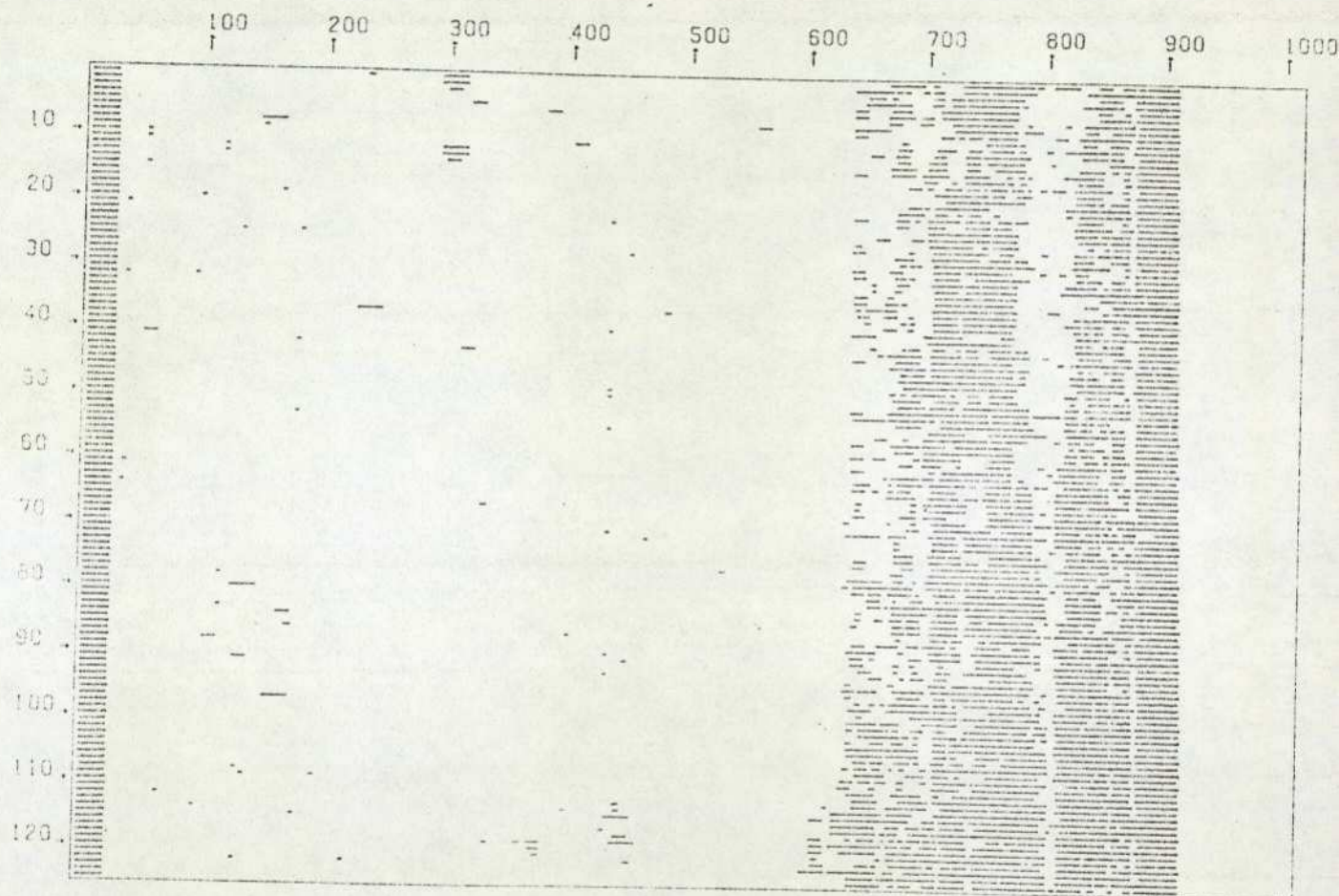
If the current scan section is the first one of a new defect, the "previous" scan section is assumed to consist of a sequence of 0's.

The hardware implementation of these two units is relatively simple and is discussed in chapter 9. The operation of these units is simulated by the subroutines SYMEX and PREDEX, respectively, which are called by the program CLUMPL, described in para. 4.4.2.

4.3.1 Multiple defects

The previous discussion was concerned with the simplest case, when defects occur at relatively long intervals, such that the situation in which the system must deal simultaneously with a certain number of separate defects, as shown in fig. 4.3, does not arise. For practical reasons it has to be assumed that there is a maximum number of active defects which must be considered simultaneously at any time, denoted N_{DM} , and consecutive defects are numbered in increasing order, modulo N_{DM} .

The delineation system associates to each scan section the number specifying the defect to which this scan section belongs.



SHEET EIGH. MATCHED FILTER VIDEOPRINT. 05-09-76. 4 FILTERS WITH THRESHOLDS 9.0/ 10.0/ 12.0/ 15.0/

Fig. 4.3 Matched filter videoprint.

Table 4. Extraction of syntactic
information units.

J=	1	1	1	2	2	2	4	2	3
J=	2	1	2	4	6	9	7	4	3
J=	3	1	2	3	6	3	7	6	3
J=	4	1	9	6	5	3	7	6	10
J=	5	1	2	3	2	9	6	5	3
J=	6	1	11	3	2	9	7	5	3
J=	7	1	9	6	6	3	6	6	10
J=	8	1	12	4	2	13	3	2	9
J=	9	1	12	2	4	9	7	5	10
J=	10	1	14	2	2	9	7	6	10
J=	11	1	14	2	6	15	3	2	9
J=	12	1	15	2	2	2	4	2	9
J=	13	1	15	2	6	2	4	6	9
J=	14	1	2	3	4	9	7	6	10
J=	15	1	14	2	2	9	7	5	10
J=	16	1	9	7	6	10	6	6	1
J=	17	1	9	7	4	10	6	6	1
J=	18	1	9	7	5	10	6	6	1
J=	19	1	12	2	2	9	7	6	10
J=	20	1	17	2	2	9	7	4	10
J=	21	1	15	2	2	9	7	5	10
J=	22	1	9	7	4	10	6	6	1
J=	23	1	19	2	2	9	7	6	10
J=	24	1	9	6	6	10	6	6	1
J=	25	1	20	2	2	21	2	2	9
J=	26	1	9	7	6	10	7	5	2
J=	27	1	9	7	6	10	7	4	2
J=	28	1	22	2	2	9	7	5	10
J=	29	1	9	7	6	10	7	4	2
J=	30	1	9	7	5	10	7	4	12
J=	31	1	9	7	6	10	6	4	12
J=	32	1	18	2	2	17	2	2	9
J=	33	1	9	7	6	10	4	6	12
J=	34	1	9	7	6	10	4	6	12
J=	35	1	9	7	6	10	6	5	17
J=	36	1	9	7	5	10	6	6	17
J=	37	1	23	4	2	24	2	2	9
J=	38	1	9	7	6	10	7	5	17
J=	39	1	9	7	6	10	7	5	17
J=	40	1	25	2	2	9	7	5	17
J=	41	1	26	3	2	9	7	5	17
J=	42	1	27	2	2	9	7	5	17
J=	43	1	28	3	2	9	7	5	17
J=	44	1	9	7	6	10	7	5	17
J=	45	1	9	7	6	10	7	5	17
J=	46	1	9	7	6	10	7	5	17
J=	47	1	9	7	6	10	7	5	17
J=	48	1	9	7	6	10	7	5	17
J=	49	1	25	2	2	9	7	5	17
J=	50	1	25	2	6	9	7	6	10
J=	51	1	9	7	6	10	7	5	17
J=	52	1	9	7	6	10	7	5	17
J=	53	1	27	2	2	9	7	6	10
J=	54	1	9	7	6	10	7	5	17
J=	55	1	25	2	2	9	7	6	10
J=	56	1	9	7	5	10	7	5	17
J=	57	1	9	7	5	10	7	5	17
J=	58	1	9	7	5	10	7	5	17

J# 50 : 9 7 5 25 1 1 27 1 1
 J# 51 : 9 7 5 25 1 1 27 1 1
 J# 52 : 9 7 5 25 1 1 27 1 1
 J# 53 : 9 7 5 25 1 1 27 1 1
 J# 54 : 9 7 5 25 1 1 27 1 1
 J# 55 : 9 7 5 25 1 1 27 1 1
 J# 56 : 9 7 5 25 1 1 27 1 1
 J# 57 : 9 7 5 25 1 1 27 1 1
 J# 58 : 9 7 5 25 1 1 27 1 1
 J# 59 : 9 7 5 25 1 1 27 1 1
 J# 60 : 9 7 5 25 1 1 27 1 1
 J# 61 : 9 7 5 25 1 1 27 1 1
 J# 62 : 9 7 5 25 1 1 27 1 1
 J# 63 : 9 7 5 25 1 1 27 1 1
 J# 64 : 9 7 5 25 1 1 27 1 1
 J# 65 : 9 7 5 25 1 1 27 1 1
 J# 66 : 9 7 5 25 1 1 27 1 1
 J# 67 : 9 7 5 25 1 1 27 1 1
 J# 68 : 9 7 5 25 1 1 27 1 1
 J# 69 : 9 7 5 25 1 1 27 1 1
 J# 70 : 9 7 5 25 1 1 27 1 1
 J# 71 : 9 7 5 25 1 1 27 1 1
 J# 72 : 9 7 5 25 1 1 27 1 1
 J# 73 : 9 7 5 25 1 1 27 1 1
 J# 74 : 9 7 5 25 1 1 27 1 1
 J# 75 : 9 7 5 25 1 1 27 1 1
 J# 76 : 9 7 5 25 1 1 27 1 1
 J# 77 : 9 7 5 25 1 1 27 1 1
 J# 78 : 9 7 5 25 1 1 27 1 1
 J# 79 : 9 7 5 25 1 1 27 1 1
 J# 80 : 9 7 5 25 1 1 27 1 1
 J# 81 : 9 7 5 25 1 1 27 1 1
 J# 82 : 9 7 5 25 1 1 27 1 1
 J# 83 : 9 7 5 25 1 1 27 1 1
 J# 84 : 9 7 5 25 1 1 27 1 1

J# 85 : 9 7 5 25 1 1 27 1 1
 J# 86 : 9 7 5 25 1 1 27 1 1
 J# 87 : 9 7 5 25 1 1 27 1 1
 J# 88 : 9 7 5 25 1 1 27 1 1
 J# 89 : 9 7 5 25 1 1 27 1 1
 J# 90 : 9 7 5 25 1 1 27 1 1
 J# 91 : 9 7 5 25 1 1 27 1 1
 J# 92 : 9 7 5 25 1 1 27 1 1
 J# 93 : 9 7 5 25 1 1 27 1 1
 J# 94 : 9 7 5 25 1 1 27 1 1
 J# 95 : 9 7 5 25 1 1 27 1 1
 J# 96 : 9 7 5 25 1 1 27 1 1
 J# 97 : 9 7 5 25 1 1 27 1 1
 J# 98 : 9 7 5 25 1 1 27 1 1
 J# 99 : 9 7 5 25 1 1 27 1 1
 J# 100 : 9 7 5 25 1 1 27 1 1
 J# 101 : 9 7 5 25 1 1 27 1 1
 J# 102 : 9 7 5 25 1 1 27 1 1
 J# 103 : 9 7 5 25 1 1 27 1 1
 J# 104 : 9 7 5 25 1 1 27 1 1
 J# 105 : 9 7 5 25 1 1 27 1 1
 J# 106 : 9 7 5 25 1 1 27 1 1
 J# 107 : 9 7 5 25 1 1 27 1 1
 J# 108 : 9 7 5 25 1 1 27 1 1
 J# 109 : 9 7 5 25 1 1 27 1 1
 J# 110 : 9 7 5 25 1 1 27 1 1
 J# 111 : 9 7 5 25 1 1 27 1 1
 J# 112 : 9 7 5 25 1 1 27 1 1
 J# 113 : 9 7 5 25 1 1 27 1 1
 J# 114 : 9 7 5 25 1 1 27 1 1
 J# 115 : 9 7 5 25 1 1 27 1 1
 J# 116 : 9 7 5 25 1 1 27 1 1
 J# 117 : 9 7 5 25 1 1 27 1 1
 J# 118 : 9 7 5 25 1 1 27 1 1
 J# 119 : 9 7 5 25 1 1 27 1 1
 J# 120 : 9 7 5 25 1 1 27 1 1
 J# 121 : 9 7 5 25 1 1 27 1 1
 J# 122 : 9 7 5 25 1 1 27 1 1
 J# 123 : 9 7 5 25 1 1 27 1 1
 J# 124 : 9 7 5 25 1 1 27 1 1
 J# 125 : 9 7 5 25 1 1 27 1 1

39 1 1
 40 1 1

Table 4 continued.

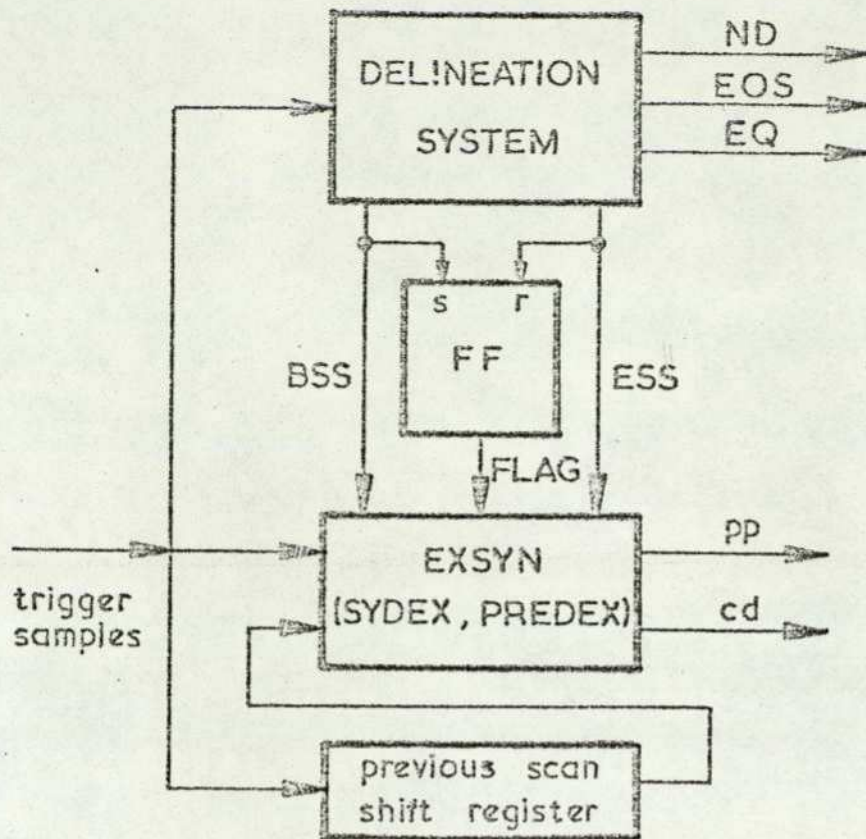


Fig.4.4 Structure of EXSYN block

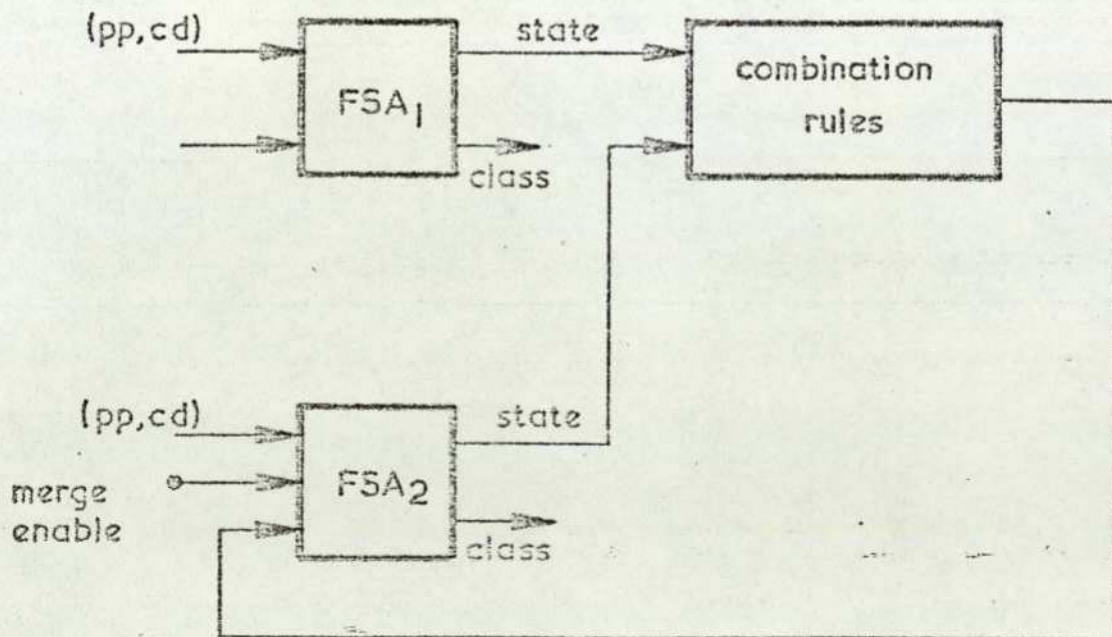


Fig.4.5 Merging defects

This number is then attached to the pattern primitive and composition descriptor associated with the same scan section and transmitted as a syntactic information unit (SIU) to the classification system.

The computer printout in table 4 shows the syntactic information units which were extracted for the sheet represented in fig. 4.3.

4.3.2 Merging separate defects

Certain complications are introduced by the situations in which two defects are initially assigned separate numbers, but merge later because the distance between them decreases and the capture rectangle covers both. In these cases the delineation block will assign the greater defect number to both defects and subsequent scan sections belonging to the merged defect will be assigned the common number.

The main difficulty is due to the scan sections belonging to these two defects, which were encountered in the scans preceding the scan in which the defects merged and therefore had already been transmitted to the classification system with different numbers assigned to them. In principle, this problem is solvable for the class of regular languages, because the only information a FSA retains about symbols already scanned is contained in its current state. Before merging the defects, each one had a FSA allocated, whose current state was updated as new scan sections were encountered. The current states of these two FSA's contain therefore a description of the two separate defects. When the two defects are merged, the

descriptions provided by the current states of the corresponding FSA's can be combined to obtain a unique description of the merged defects. In practical terms, this idea can be implemented by using the output of a FSA as a special input to the other FSA, which will execute a "forced" transition when this input is enabled by the signal EQ. This principle is illustrated in fig. 4.5.

The procedure discussed above is of interest for the actual on-line implementation of the syntactic classifier. At the present stage, a complete description of all defects is more important, in order to infer good grammars. Consequently, the program which simulates the extraction of the syntactic description is based on a different approach, which produces an accurate description of all defects, but can not be implemented on-line, because it requires two cycles of operation:

1. in the first phase, each scan section is processed separately, and the corresponding SIU (defect number, pattern primitive, composition descriptor) is stored; all merging operations are recorded in a table which indicates the defect numbers which became equivalent.

2. in the second phase, the merging table is rearranged to take into account multiple, successive merging operations; then the information obtained in the first phase is updated scan by scan, modifying the SIUs assigned to scan sections which have been merged, as follows:

- the last defect number becomes the defect number of the merged SIU

- the pattern primitives are combined in pairs, according to a merging table for pattern primitives
- the composition descriptors are combined in pairs, according to a merging table for composition descriptors.

Table 5 and table 6 show the two merging tables used in the thesis.

4.3.3 Non-contiguous defects

The vertical dimension of the capture rectangle, denoted K_y , is in general greater than 1, in order to permit the delineation of defects which consist of non-contiguous sections. A defect is rendered inactive if no scan sections are assigned to it over a number K_y of consecutive scans. The situations in which no scan sections are assigned to a certain active defect over a scan are not insignificant, because this characterises how "compact" a defect is. In order to deal with this type of situation, the EXSYN block carries out an additional operation when it receives the signal EOS : for all active defects not represented by a scan section over the current scan, it generates a special SIU, consisting of: the number of the defect, the NOS pattern primitive and NOS composition descriptor (NOS - "no scan section" is coded as no. 1 in tables 2 and 3).

4.3.4 Syntesis of the string representation of defects from the sequence of SIUs

The output of the block EXSYN is a continuous sequence of SIUs, which characterise scan sections, ordered from left to right within a scan and followed by a tail of special SIUs (NOS/ NOS) for the active defects not encountered during a scan.

TABLE 5

MERGING PATTERN PRIMITIVES

$P_1 \backslash P_2$	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	5	5	5	7	7	7
3	3	2	5	6	7	7	7
4	4	5	6	6	7	7	7
5	5	7	7	7	7	7	7
6	6	7	7	7	7	7	7
7	7	7	7	7	7	7	7

TABLE 6

MERGING COMPOSITION DESCRIPTORS

$D_1 \backslash D_2$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	2	4	4	5	5
3	3	4	3	4	5	5
4	4	4	4	4	5	5
5	5	5	5	5	5	5
6	6	5	5	5	5	6

The synthesis of the string representation of defects consists of concatenating the SIUs belonging to the same defect, in the order in which they were encountered during the scanning process. This operation is performed only in the design phase of the classifier, for inferring the pattern grammars; during the on-line operation of the classifier, each scan section is processed as it is encountered. Table 7 illustrated the method by showing the final syntactic representation of the defects encountered on the videoprint depicted in fig. 4.3, obtained by processing the scan sections listed in table 4.

4.4 Programs for extracting the syntactic representation of defects

The FORTRAN program for extracting the syntactic representation of defects is organised as shown in the flowchart in fig. 4.6, which outlines the main functions of the program:

1. Reads the problem parameters: the identifiers of the sheets to be analysed, the dimensions of the capture rectangle K_X and K_Y and the parameters of the pattern primitives and composition descriptors.
2. Reads the data which corresponds to the selected sheets from the main data base, rearranges this data and writes it in a convenient form on magnetic discs. This operation is necessary because the main data was acquired at SIRA using an automated test rig and the organization of the data on the recorded tapes is largely imposed by the operation mode of the test rig (for details see Chapter 7), but is not convenient for frequent access by the subsequent

Table 7. Syntactic representation of defects.

Sheet E 10 M

SITUATION BEFORE MERGING: ISPFIN= 924 NRDEF=53

SITUATION AFTER MERGING: NRDEF=42

DEF NR= 1 STARTED AT SCAN= 1, TRIGGER= 229 AND TERMINATED AT SCAN= 16
21111111111111111111
21111111111111111111
DEF NR= 2 STARTED AT SCAN= 1, TRIGGER= 293 AND TERMINATED AT SCAN= 29
44313111111144311111111111111111
26512111111126511111111111111111
DEF NR= 11 STARTED AT SCAN= 6, TRIGGER= 377 AND TERMINATED AT SCAN= 21
31111111111111111111
21111111111111111111
DEF NR= 12 STARTED AT SCAN= 6, TRIGGER= 141 AND TERMINATED AT SCAN= 34
42111111111121111111111111111111
24111111111121111111111111111111

```

DEF NR= 13  STARTED AT SCAN= 8, TRIGGER= 553 AND TERMINATED AT SCAN= 23
31111111111111111111
21111111111111111111
DEF NR= 14  STARTED AT SCAN= 10, TRIGGER= 48 AND TERMINATED AT SCAN= 30
221112111111111111111111
261112111111111111111111
DEF NR= 15  STARTED AT SCAN= 11, TRIGGER= 400 AND TERMINATED AT SCAN= 26
31111111111111111111
21111111111111111111
DEF NR= 16  STARTED AT SCAN= 12, TRIGGER= 112 AND TERMINATED AT SCAN= 28
22111111111111111111
26111111111111111111
DEF NR= 17  STARTED AT SCAN= 20, TRIGGER= 94 AND TERMINATED AT SCAN= 47
2111111111112111111111111111
2111111111111211111111111111
DEF NR= 18  STARTED AT SCAN= 21, TRIGGER= 32 AND TERMINATED AT SCAN= 47
2111111111121111111111111111
2111111111121111111111111111
DEF NR= 19  STARTED AT SCAN= 23, TRIGGER= 432 AND TERMINATED AT SCAN= 38
21111111111111111111
21111111111111111111
DEF NR= 20  STARTED AT SCAN= 25, TRIGGER= 128 AND TERMINATED AT SCAN= 40
21111111111111111111
21111111111111111111
DEF NR= 21  STARTED AT SCAN= 25, TRIGGER= 176 AND TERMINATED AT SCAN= 40
21111111111111111111
21111111111111111111
DEF NR= 22  STARTED AT SCAN= 28, TRIGGER= 448 AND TERMINATED AT SCAN= 43
21111111111111111111
21111111111111111111
DEF NR= 23  STARTED AT SCAN= 37, TRIGGER= 224 AND TERMINATED AT SCAN= 52
41111111111111111111
21111111111111111111
DEF NR= 24  STARTED AT SCAN= 37, TRIGGER= 478 AND TERMINATED AT SCAN= 52
21111111111111111111
21111111111111111111
DEF NR= 25  STARTED AT SCAN= 40, TRIGGER= 432 AND TERMINATED AT SCAN= 70
211111111221111211111111111111
211111111261111211111111111111
DEF NR= 26  STARTED AT SCAN= 41, TRIGGER= 48 AND TERMINATED AT SCAN= 56
31111111111111111111
21111111111111111111
DEF NR= 27  STARTED AT SCAN= 42, TRIGGER= 174 AND TERMINATED AT SCAN= 68
2111111111121111111111111111
2111111111121111111111111111
DEF NR= 28  STARTED AT SCAN= 43, TRIGGER= 310 AND TERMINATED AT SCAN= 58
31111111111111111111
21111111111111111111

```

Table 7 continued.

[illegible]

programs, which are run on the CDC 7600 computer.

3. Extracts the SIUs (sheet by sheet and scan by scan), carries out the modifications required by merging processes and writes the resultant array on a file denoted SYMPRED.

4. the SYMPRED file is processed in order to synthesise from separate SIUs the syntactic representation of the corresponding defects, which are then stored in two tables (IT and KST) and recorded on another file.

5. Finally, the string representations of the defects are analysed in order to evaluate the separation capabilities of the pattern primitives and composition descriptors and also the length distribution of the strings in different classes.

For programming and execution convenience, the flowchart represented in fig. 4.6 has been implemented in two separate main programs, called CLUMPL (for CLUMP-ing Linguistic) and LIPAN (for Linguistic Pattern Analysis). Since the execution of the CLUMPL program simulates the operation of the EXSYN block of the syntactic pattern recognition system (see fig. 4.4), it was considered that a detailed description of this program is useful for a better understanding of the hardware implementation which is discussed in Chapter 9. The complete programs are listed in [34].

4.4.1 Information Structure

This paragraph defines the main data structures used in the programs CLUMPL and LIPAN, their functions and dimensions:

4.4.1.1 Specular videoprint data: for the purpose of this thesis, the specular videoprint data, which is extracted from the data

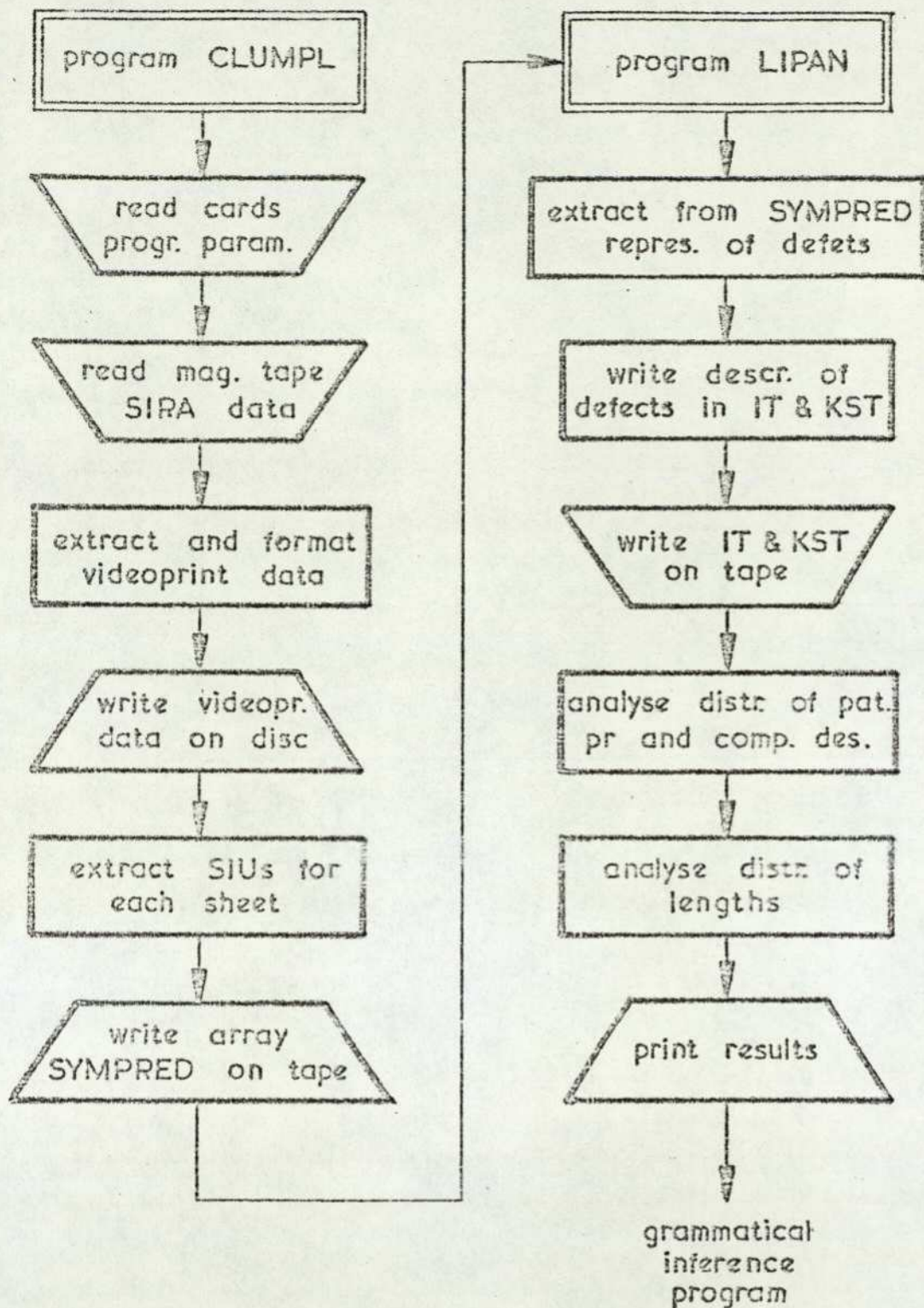


Fig.4.6 Flowchart of program for extracting the syntactic representation of defects

provided by the scanner using the matched filters technique, will be considered the main data base. This file is divided in subfiles corresponding to the sheets whose videoprints were recorded, each identified by a 10 character name; each subfile is organised in blocks of data describing complete scans, as shown in fig. 4.7.

4.4.1.2 Arrays TRIG and PTRIG: since the entire system operates on a scan by scan basis, it is necessary to store in core memory the trigger samples of the current scan (TRIG) and previous scan (PTRIG) - the later is required for the extraction of composition descriptors.

4.4.1.3 Array SYMPRED (2000,4): is a two dimensional table which stores the SIUs as they are generated and finally the SIUs which result after processing the merged defects.

The scan sections are characterised by 4 parameters, which are stored in successive rows as follows: column 1 - the defect number; column 2 - the pattern primitive; column 3 - the composition descriptor ; column 4 - the scan number to which the scan section belongs. (This information is required in order to identify scan sections which are in the same scan).

4.4.1.4 Array DEFST (50,4): describes the current state of the defects. Each row of the array describes the state of a defect by storing the following parameters:

Column 1 - is a flag (1/0) which indicates whether a certain defect is currently active or inactive; column 2 - specifies the number of scans over which an active defect was not seen; column 3 - stores defect numbers equivalences induced by merging operations; column 4 - is a flag (1/0) which indicates whether a

LMW—leading margin width
 TMW—trailing margin width

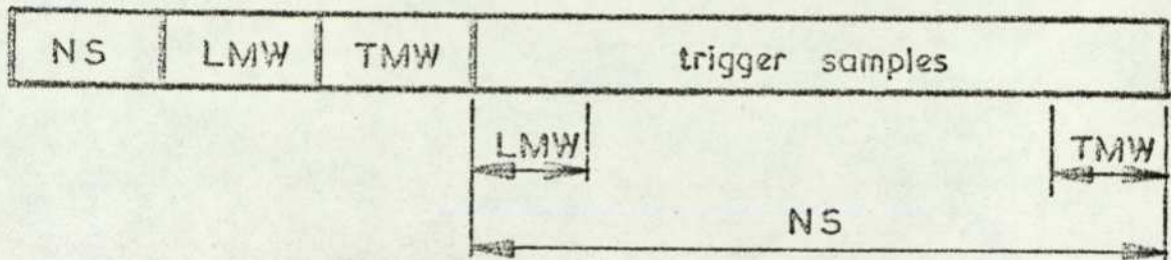


Fig.4.7 Structure of data block

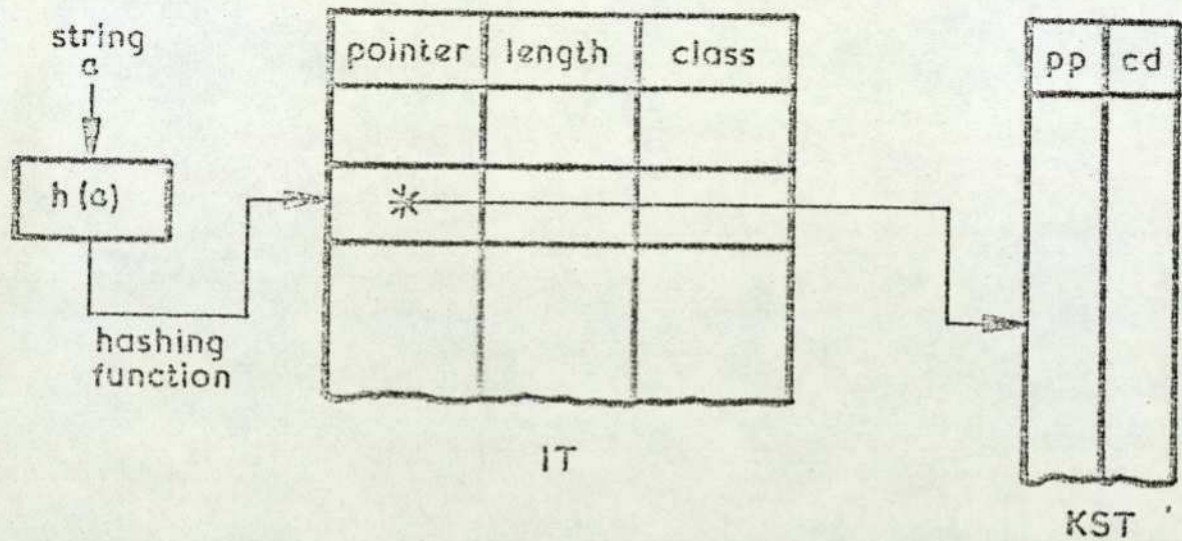


Fig.4.8 Hash coding procedure

certain defect is active/inactive over the current scan.

4.4.1.5 Arrays IT(1024,3) and KST(10000,2):

Together, these two tables are used for storing and retrieving all strings, using a fast hashcoding technique which is described in par. 4.4.4. Basically, this method assigns an unique integer number to each string, which represents the entry in the table IT associated to that string. The first column of that entry represents an address pointer to the first location in table KST where the string is stored; the second column stores the length of the string and the third column of table IT stores the class of the pattern described by that string. In the table KST, the first column stores the pattern primitive and the second column stores the composition descriptor. The procedure is illustrated in fig. 4.8 and has the merit of using the memory in an efficient manner and operates very fast.

4.4.2 The program CLUMPL

The general structure of the program is explained in the flowchart in fig. 4.6; the complicated functions of the program (processing of scan sections and reprocessing the merged scan sections) are detailed in separate flowcharts (fig. 4.9 and fig. 4.10), which need only a few comments.

4.4.2.1 Scan section processing

The beginning of the first scan section is defined by the first trigger sample which is 1; the following scan sections are defined by the first trigger sample which is 1 and follows a sequence of at least K_x 0's. The end of a scan section is defined by the K_x^{th}

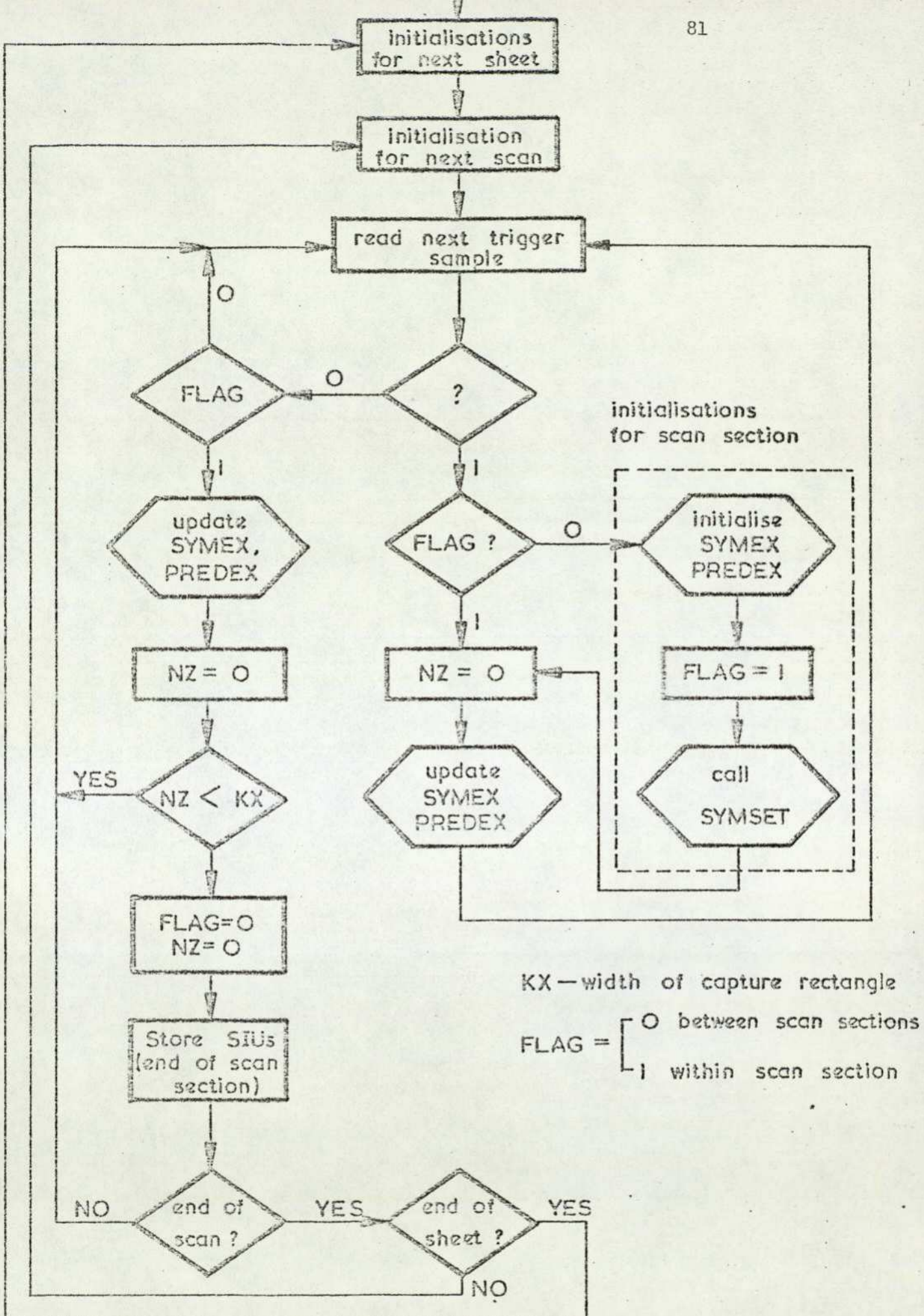


Fig.4.9 Processing of scan sections

O-trigger sample or, in case of the last scan section, by the last trigger sample. These functions of the program are implemented using

- a) a variable FLAG (0/1) to indicate whether the current trigger sample is between or within scan sections, respectively.
- b) a counter NZ which is reset by each 1-trigger sample and increased by each O-trigger sample.

This section of the main program calls the following subroutines:

- a) SYMSET: assigns a defect number to each scan section and indicates when a merging occurred.
- b) SYMEX: assigns a pattern primitive to the scan section
- c) PREDEX: assigns a composition descriptor to the scan section.

4.4.2.2 Reprocessing the merged scan sections

Merging operations are indicated by the 3rd column of the DEFST array, which is changed from its normal value (equal to the row number) to the number of the combined defect when a merging operation takes place. The reprocessing of the merged scan sections is described by the flowchart shown in fig. 4.10.

4.4.3 The program LIPAN

The general structure of this program is given in the flowchart in fig. 4.6, which is selfexplanatory. The following subroutines are called in the program LIPAN:

- 4.4.3.1 WRKST: writes a string and its parameters (length and the class of the pattern represented by the string) in the tables IT and KST; the row number of the Table IT assigned to the string is computed by the subroutine HASH, which is described separately in

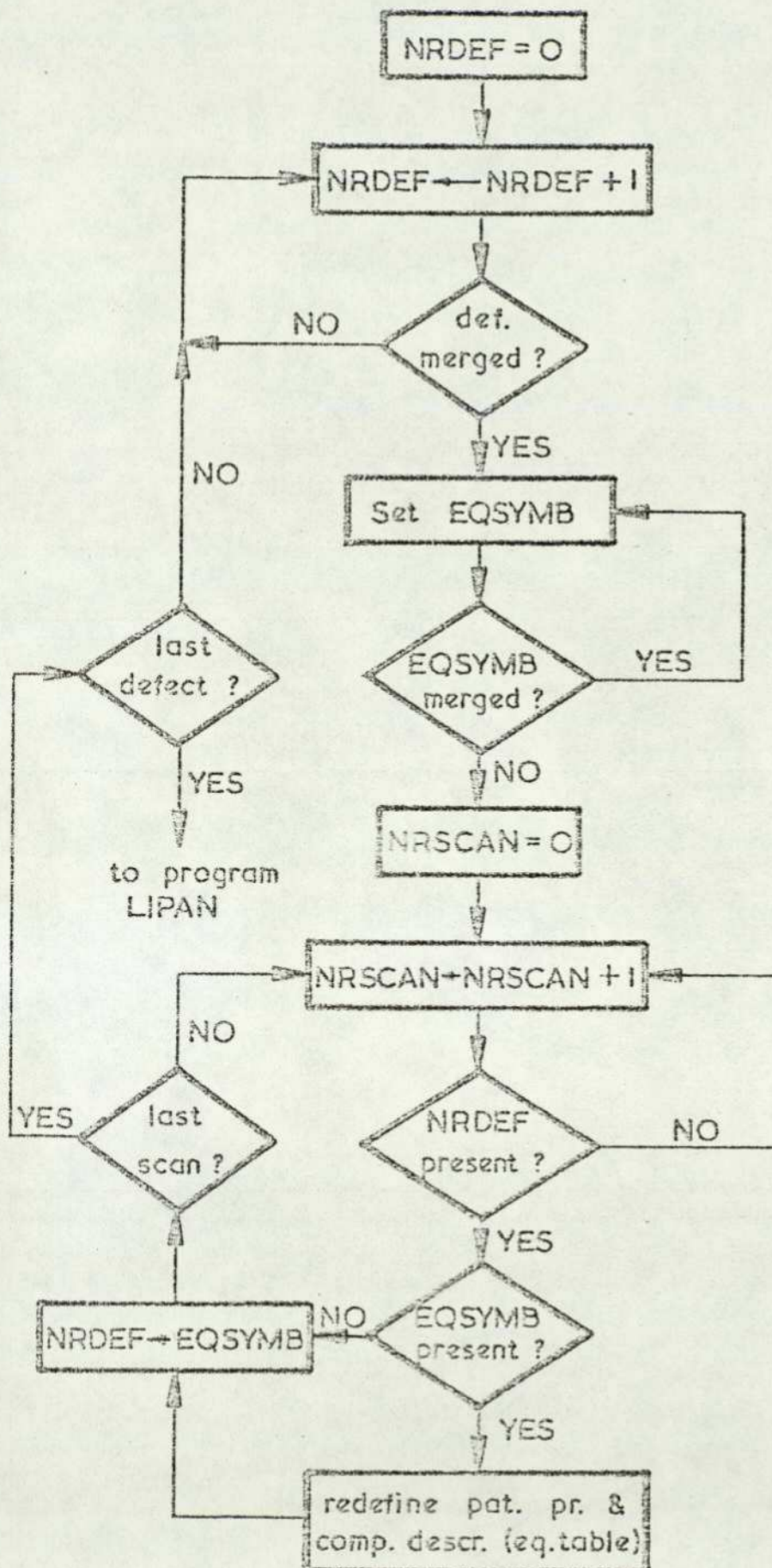


Fig. 4.10 Processing merged defects

the next paragraph, because of its importance.

4.4.3.2 SPAN: extracts all pattern primitives and composition descriptors used in the strings belonging to the same class and also computes the frequency of these elements (together and separately) per each class, in order to evaluate the separation capabilities of the selected syntactic elements.

4.4.3.3 LENAL: computes the length distribution of the strings within the classes, in a number of intervals which can be adjusted as a parameter.

4.4.4 The hash coding subroutine (HASH)

Computationally, syntactic pattern recognition is a nonnumerical application in which writing and retrieving strings of variable length are fundamental operations and must therefore be executed at maximum speed and with an efficient memory utilization; one of the best solutions to this problem is hash coding [6].

The hash storage mechanism uses a hasing function, h , a hash table, a data storage table and a name table. In the present case, the hash table and the data table are combined and denoted by HT ; the name table is denoted by KST and stores the strings (fig. 4.8).

The hashing function employed by the subroutine $HASH$ is defined as follows:

$$h_0 = \sum_{i=1}^{\ell} (10^i + d_i) \bmod n + 1 \quad (4.12)$$

$$h_i = (h_0 + r_i) \bmod n \quad (4.13)$$

where r_i are pseudorandom numbers generated by performing the sequence of operations:

$$R = 5R \quad (\text{initially } R = 1) \quad (4.14a)$$

$$R = R \bmod 4n \quad (4.15b)$$

$$r = \left[R/4 \right] \quad (4.16c)$$

It was found that this hashing function scatters the integers uniformly throughout the table IT. Relation (4.12) is valid only if the number of distinct pattern primitives and composition descriptors is less than 10; if this condition is not satisfied, the constant 10 must be increased accordingly.

After selecting a set of pattern primitives and composition descriptors, the next problem to be considered was developing a method to infer the pattern grammars. Experience gained during the initial stages of this study has shown that inferring good grammars is a complex process, even when the structure of the patterns is relatively simple, because the whole procedure has to be repeated: 1) for each set of intuitively designed pattern primitives and composition descriptors; 2) for each pattern class separately; 3) for successive solutions, with increasing degrees of accuracy and therefore increasing complexity; 4) when new patterns are added to the training set. Consequently, it was decided to develop and program an algorithm for grammatical inference, specially adapted to the requirements of the present application:

1) the solution space should be the class of regular grammars, capable of approximating the language with adjustable degrees of accuracy and complexity.

2) the algorithm should be capable of handling two-dimensional patterns, described by the formalism introduced in the preceding chapter.

3) the training set of patterns contains defects in n classes and the algorithm should produce a separate grammar for describing each pattern class (language).

Since no method found in the literature surveyed [23] - [32] satisfied completely these requirements, the approach taken was to

adopt and extend an algorithm proposed by Bierman and Feldman for inferring a regular string grammar from an informant presentation of a language [32]. This algorithm is reviewed briefly in the first paragraph of this chapter. The second paragraph describes the procedure which has been developed in order to generalize the algorithm to the case of multi-class, two-dimensional patterns, as required by the present application. Finally, the practical implementation of the algorithm and the corresponding FORTRAN which has been written and used are described in the last paragraph.

5.1 Algorithm for synthesizing a finite state automaton from a sample of a regular language.

This paragraph contains a brief presentation of the algorithm developed by Bierman and Feldman for designing FSAs from samples of their behaviour [32]. The formal proofs concerning the properties of the algorithm can be found in the original paper.

5.1.1 The Nerode relation

The inference algorithm is described best in terms of an equivalence relation introduced by Nerode and redefined in [32] as follows:

Let Σ be a set of terminal symbols and f a function which assigns to strings $x \in \Sigma^*$ values in $I = \{1, 2, 3, \dots, N\}$. The function f is not necessarily total; for some strings x , $f(x) = 0$ (undefined). For example, if we have a set of strings which are to be accepted and a set of strings which are to be rejected, the function f will assign to these strings the values 1 and 2, respectively.

Let x and Z be two strings $\in \Sigma^*$ and xZ their concatenation; the function $f(xZ)$ will be denoted $f_x(Z)$.

The Nerode relation R_f on the set Σ^* is defined as follows:

$$(x_1, x_2) \in R_f \text{ iff } f_{x_1}(z) = f_{x_2}(z) \text{ for all } z \in \Sigma^* \quad (5.1)$$

The equivalence class induced by R_f which contains a string x is

$$[x]_{R_f} = \{ x^1 \in \Sigma^* \mid (x, x^1) \in R_f \} \quad (5.2)$$

The function f is called a finite-state function if the number of equivalence classes is finite. The equivalence classes $[x]_{R_f}$ can be regarded as the states of a FSA which computes the function f . This machine, denoted $M(f)$, is defined as follows:

$$M(f) = (Q, \Sigma, \delta, q_0, F) \quad (5.3)$$

where Σ - the finite set of input symbols

$$Q = \{ [x]_{R_f} \mid x \in \Sigma^* \} - \text{the finite set of states}$$

$$\delta([x]_{R_f}, a) = [xa]_{R_f} \quad - \text{the state transition function} \\ (a \in \Sigma)$$

$$q_0 = [\lambda]_{R_f} - \text{the initial state}$$

$F \subseteq Q$ - the set of final states

where $F_i = \{ [x]_{R_f} \mid f(x) = i, i \in I \}$ - is the set of final states associated to output i .

5.1.2 The modified Nerode relation

Bierman and Feldman have suggested a modification to the Nerode relation, by introducing a parameter K to describe the maximum length of the suffixes which have to be considered when finding the equivalence classes.

The training set T is defined formally as a subset of $\Sigma^* \times I$, such that if $(x, i) \in T$, then $f(x) = i$. The strings in the training set are denoted $P(T)$ because they represent the projection

of T on the coordinate x . Let f_T be defined as the restriction of f to the pairs represented in T : $f_T(x) = 0$ if $x \notin P(T)$.
(K)

The modified Nerode relation, $E_T^{(K)}$, is defined as follows:
for a given $K \gg 0$,

$$(x_1, x_2) \in E_T^{(K)} \text{ iff for all } z \in \Sigma^* \text{ with length } |z| \leq k: \\ (1) \quad x_1 z \in P(T) \text{ iff } x_2 z \in P(T) \quad (5.4a)$$

$$(2) \quad f_T(x_1 z) = f_T(x_2 z) \text{ when defined} \quad (5.4b)$$

The equivalence classes induced by $E_T^{(K)}$ in the set Σ^* :

$$[x]_T^{(k)} = \{ x^1 \mid (x, x^1) \in E_T^{(k)} \} \quad (5.5)$$

will become the states of a nondeterministic FSA defined as follows:

$$M_T^{(k)} = (Q, \Sigma, \delta, q_0, F)^* \quad (5.6)$$

where: $Q = \{ [x]_E \}$ - the finite set of states

Σ - the finite set of input symbols

δ - the state transition function, defined as follows:

$$\delta([x], a) = \{ [x^1 a] \mid x^1 \in [x] \}, a \in \Sigma \text{ is constructed recursively:}$$

$$\delta([x], \lambda) = \{ [x] \}$$

$$\delta([x], za) = \bigcup_{x^1 \in \delta([x], z)} \delta([x^1], a)$$

$$q_0 = [\lambda] \text{ - the initial state}$$

$F \in Q$ - the set of final states.

The nondeterministic output function associated to $M^{(k)}$ and input string x is:

$$h(x) = \{ f(z) \mid [z] \in \delta([\lambda], x) \text{ and } z \in P(T) \} \quad (5.7)$$

The function $h_T^{(k)}$ represents an approximate realization

* The subscript T and the superscript (k) will be dropped in the subsequent definitions in order to simplify the notations; for example,

$E_T^{(k)}$ will be written simply as E .

of the function f_T ; the degree of accuracy can be adjusted by the designer by varying the value of the parameter k . For low values of k , $M_T^{(k)}$ is nondeterministic, has few states but a low degree of accuracy; for higher values of k , the number of states and the degree of accuracy increase. Finally, $M_T^{(k)}$ becomes deterministic and the function f_T is computed exactly, if k is equal or greater than the length of the longest string in $P(T)$.

These properties can be expressed formally as follows:

Property 1. If $f_T(x)$ is defined, then $f_T(x) \in h(x)$ (5.8)

Property 2. If k is greater than or equal to the length of the longest string in $P(T)$, then

$$h(x) = \{f_S(x)\} \quad \text{if } x \in P(T) \quad (5.9a)$$

$$h(x) = \emptyset \quad \text{otherwise} \quad (5.9b)$$

Property 3.

$$h_T^{(k+1)}(x) \subseteq h_T^{(k)}(x) \quad (5.10)$$

Property no. 2 reveals that $M_T^{(k)}$ is an acceptor of exactly the set $P(T)$, but does not accept any string $x \notin P(T)$. This important limitation implies that only the training set can be classified correctly and the recognizer has no extrapolation capabilities. Since in practice $P(T)$ is always a subset of the language which has to be accepted, it is necessary to find a set of conditions under which the set T contains enough information to permit an extrapolation from f_T to f .

5.1.3 Correct extrapolation from f_T to f

The restrictions on $P(T)$ which guarantee that $M_T^{(k)}$ will extrapolate from f_T to f can be expressed using two auxiliary sets of strings.

a) The set A_f is defined to be any set of strings in Σ^* such that:

1. every state q of $M(f)$ is reached by some string Z in A_f :

$$\delta_{R_f}([\lambda]_{R_f}, Z) = q$$

2. every prefix of a string in A_f is also in A_f : if

$$Z = z_1 z_2 \dots z_n \in A_f, \text{ then}$$

$$z_1 z_2 z_3 \dots z_i \in A_f, \text{ for } i = 1, 2, \dots, n-1$$

b) The set B_r is defined to be the set of strings in Σ^* of length r or less:

$$B_r = \{ w \mid w \in \Sigma^*, |w| \leq r \} \quad (5.11)$$

The sets A_f and B_r ** are used to express the restrictions on T which will enable $M_T^{(k)}$ to extrapolate from f_T to f as follows:

Property 4. If f is a finite-state function and the value of the parameter r is selected high enough, then for all T such that

$$P(T) = \{ x \in A_f \times B_r \mid f(x) \text{ defined} \}$$

and for all $x \in \Sigma^*$

$$h_T^{(m)}(x) = \begin{cases} f(x) & \text{if } f(x) \text{ is defined} \\ \phi & \text{otherwise} \end{cases} \quad (5.12a)$$

$$(5.12b)$$

There is a problem in that property no. 4 requires an a priori estimate of the value of r , which is difficult to find. In practice, this difficulty can be overcome by increasing the value of m until the performance of the recognizer $M^{(m)}$ becomes satisfactory.

** the strings in B_r are sometimes called "r-tails".

5.2 Generalised algorithm for constructing FSA acceptors of multi-class two-dimensional patterns from samples of languages

In order to become applicable to the present problem, the algorithm described in the preceding paragraph requires two major extensions:

- a) a convenient method of handling two-dimensional patterns has to be incorporated in the algorithm.
- b) the training set, which consists of a sample T of patterns in N classes, has to be interpreted as an informat presentation for the purpose of constructing individual class-recognizers.

The fact that the grammar inference problem is always solved off-line, in the design phase of the syntactic recognizer, has been exploited to derive simple solutions to these problems, based on convenient interpretations of the available information.

It has been shown in the preceding chapter that the formalism developed for the present application reduces the representation of two-dimensional patterns to strings of pattern primitives/composition descriptors, called syntactic elements. Consequently, it is possible to extend the original algorithm to handle two-dimensional patterns by replacing all symbols in the original formulation with syntactic elements, according to the convention that in a formal sense each distinct pattern primitive/composition descriptor pair is interpreted as a combined symbol, as in eq. (4.9) and (4.10).

The second problem was also solved by assigning convenient interpretations to the classes of the patterns in the training set.

If S_k denotes the set of patterns in class k , the following relations are valid for any sample T :

$$T = \bigcup_{k=1}^N S_k \quad (5.13)$$

$$S_i \cap S_j \neq \emptyset \quad \text{for } i \neq j \quad (5.14)$$

For any class k , the set T can be partitioned in two disjoint subsets:

$$T = S_k^+ \cup S_k^- \quad (5.15)$$

where $S_k^+ = S_k$ is a positive sample of L_k
 $S_k^- = \bigcup_{\substack{i=1 \\ i \neq k}}^N S_i$ is a negative sample of L_k

Consequently, the set T can be partitioned to represent a sample of the language L_k for each separate class k . The grammatical inference process can therefore be decomposed into N similar problems of deriving separate grammars G_i for each class k ($k = 1, 2, \dots, N$), using as samples of the languages L_k the partitions of the set T defined by the equations (5.13) - (5.15).

Since the final task is to design the finite-state acceptors for each class, the algorithm will be oriented towards designing directly the FSA rather than the regular grammars for each class. Assuming that the acceptor for a class k is

$$\text{FSA} = (Q, \Sigma, \delta, q_0, F)$$

the states Q satisfy the following class assignment:

q_0 no defect
 F class 1, patterns in class k
 $Q-F-q_0$ class 2, patterns not in class k

An alternative method would be to design a common acceptor for all defects and to assign classes from 1 - N to the corresponding states. Preliminary experiments indicated that this approach leads finally to an increased complexity of the syntactic classifier, which will be described in chapter 6.

One of the theoretical difficulties of the original algorithm is related to the estimation of the parameter k . Since the generalised algorithm discussed in this paragraph is intended to be implemented as a program executed on a powerful computer, it was considered that a practical and acceptable solution to this problem would be to increase the value of k from 1 to a maximum value (say 10) and to select the optimum solution as a trade-off between accuracy and complexity.

Experience has shown that the algorithm in the final form could be implemented simpler by using a construction technique which required additional modifications to the original version. Basically, the construction technique developed consists of generating automatically all strings in $A_F \times B_F$, and assigning class 0 to the strings which are not in $P(T)$. This class is interpreted as a don't care condition in the definition of equivalence classes, so that the equivalence conditions are relaxed:

$(x_1, x_2) \in E_T^{(k)}$ iff for all $z \in \Sigma^*$,

$|z| \leq k : f_T(x_1 z) = f_T(x_2 z)$, when both are defined.

This important modification determines new properties of $M_T^{(k)}$:

1. the speed with which the algorithm converges to a solution is increased.
2. the number of states is reduced artificially and this can lead to a degradation of the recognition performance due to:
 - false recursive properties (loops), hence excessive extrapolation.
 - increased amount of nondeterminism.
3. the state transition function $\delta(q, a)$ has at least one member for any $q \in Q$ and $a \in \Sigma$.

The main advantage obtained by the modifications mentioned above is that a simple and approximate solution is always derived while the performance can be improved by augmenting T in an interactive process or by increasing the value of r .

5.3 Practical implementation of the generalised algorithm

If the training set T contains all the information specified in par 5.1.3, the algorithm can be executed completely automatically, but experience has shown that this is rarely the case. In general, simple interactions with a trainer were necessary in order to improve the performance of the recognizer; it is the task of the algorithm to specify the additional information required, by printing the corresponding messages.

Practically, the algorithm is applied repeatedly for each pattern class and consists of the following major steps, represented also in the flow chart in fig. 5.1:

5.3.1 Selection of the initial set A_f : the patterns in the training set, represented as strings of syntactic elements.

5.3.2 Computation of the set B_r : all k-tails w_{1i} , $|w_{1i}| \leq k$, are extracted from the strings x_{1j} in the pattern class currently processed; the strings in other classes, x_{2j} , need not be considered, since their k-tails, w_2 , cannot introduce separations in the existing equivalence classes, because

$$f_{x_1}(w_2) = f_{x_2}(w_2) = 2 \text{ (or 0, if undefined)}$$

5.3.3 Computation of the equivalence classes

a) the strings $x \times B_r$ are constructed by concatenating all $p = |B_r|$ strings in B_r as suffixes:

$$x_{sr} = \{ x s_r \mid x \in A_f, s_r \in B_r \}$$

b) the classes C_i of the strings x_{sr} are retrieved from the training set of samples (T), according to the relations (5.13) - (5.15); if $x \notin P(T)$, it will be assigned class 0 and interpreted as a don't care condition.

c) the equivalence class defined by the string x is determined by comparing the vector $C = C_1 C_2 \dots C_p$ with the vectors defining the existing equivalence classes, as follows:

- the components $C_i = 0$ are discarded, in order to avoid the introduction of unnecessary new equivalence states.

- if all nonzero elements of vector C are identical with all nonzero elements of a vector defining an existing equivalence

class, the string x belongs to that class.

- if the previous condition is not met for any of the existing equivalence classes, the string x defines a new equivalence class.

As a result of introducing don't care conditions, situations can occur in which the same string x belongs to more than just one equivalence class Q , as in the following examples:

1. $C(x) = 3406$, $Q_2 = 3416$ and $Q_3 = 3426$;
 $x \in Q_2$ and $x \in Q_3$
2. $C(x) = 1430$, $Q_4 = 1430$ and $Q_5 = 1402$;
 $x \in Q_4$ and $x \in Q_5$

5.3.4 Construction of the final A_f set:

After detecting the equivalence classes, the set A_f is redefined to include:

- (a) at least one string for each equivalence class (for convenience, the shortest is selected)
- (b) all the prefixes of these strings (if not contained in the training set, the intervention of the designer is required)

5.3.5 Computation of the final equivalence classes:

all the operations discussed in step 4) are repeated for the new A_f set.

5.3.6 Computation of the state transition matrix of the acceptor:

the equivalence classes are interpreted as states and the computations defined by the eq. (5.6) and (5.7) are performed in the following sequence:

- a) for each equivalence class, a representative string A_f is selected
- b) the strings $x_s = \{ x.s_{1j} \mid j = 1, 2, \dots, |\Sigma|_i \}$ are constructed
- c) for each string x_s , the strings $z_s = \{ x_s.s_{rk} \mid k = 1, 2, \dots, p \}$ which determine the equivalence class of the string x_s , are constructed.
- d) the classes of the strings z_s specify a vector C which corresponds to one or more equivalence classes or next-states; if the next-state is not uniquely defined, the algorithm prints a message specifying the additional information required to remove the ambiguities.
- e) the class associated with a certain state is the class of the string which forces the acceptor into that state.

Steps 1-6 are repeated for each pattern class.

5.3.7 Preliminary performance estimation.

This step was introduced to give an indication about the accuracy with which the inferred grammars describe the pattern languages. These results are extremely useful for revealing the nature of the approximations, characterised by the confusion matrices computed separately for each recognizer. The global performance of the combined syntactic classifier is determined later, in connection with the classification technique developed in chapter 6.

5.4 Description of the grammatical inference program (GRINF)

The nature of the grammar inference algorithm indicates that an interactive program would be preferable, but the computing facilities available did not permit a program of this size to be executed interactively. Consequently, the program was written in FORTRAN and run in batch processing mode on the CDC 7600 computer; the organization of the program is outlined in the flowchart shown in fig. 5.1 and the complete listing is given in [34].

5.4.1 Program input

- the number of pattern classes (NCLASS)
- the training set of patterns (arrays IT and KST)
- the set of syntactic elements used in each class (array PSP)
- the maximum length of the k-tails (KLTM)

5.4.2 Program output

- transition matrices of finite-state acceptors for each pattern class (arrays TRANSM)
- preliminary performance estimation for each acceptor (confusion matrices CONFM and global recognition performance PERF).
- additional information required to eliminate ambiguities.

5.4.3 Data bases

5.4.3.1 Tables IT and KST: store originally the strings in the training set; during the execution of the program the strings selected by the designer are added.

5.4.3.2 Table TSP: stores the strings in B_r , generated at the beginning of the program.

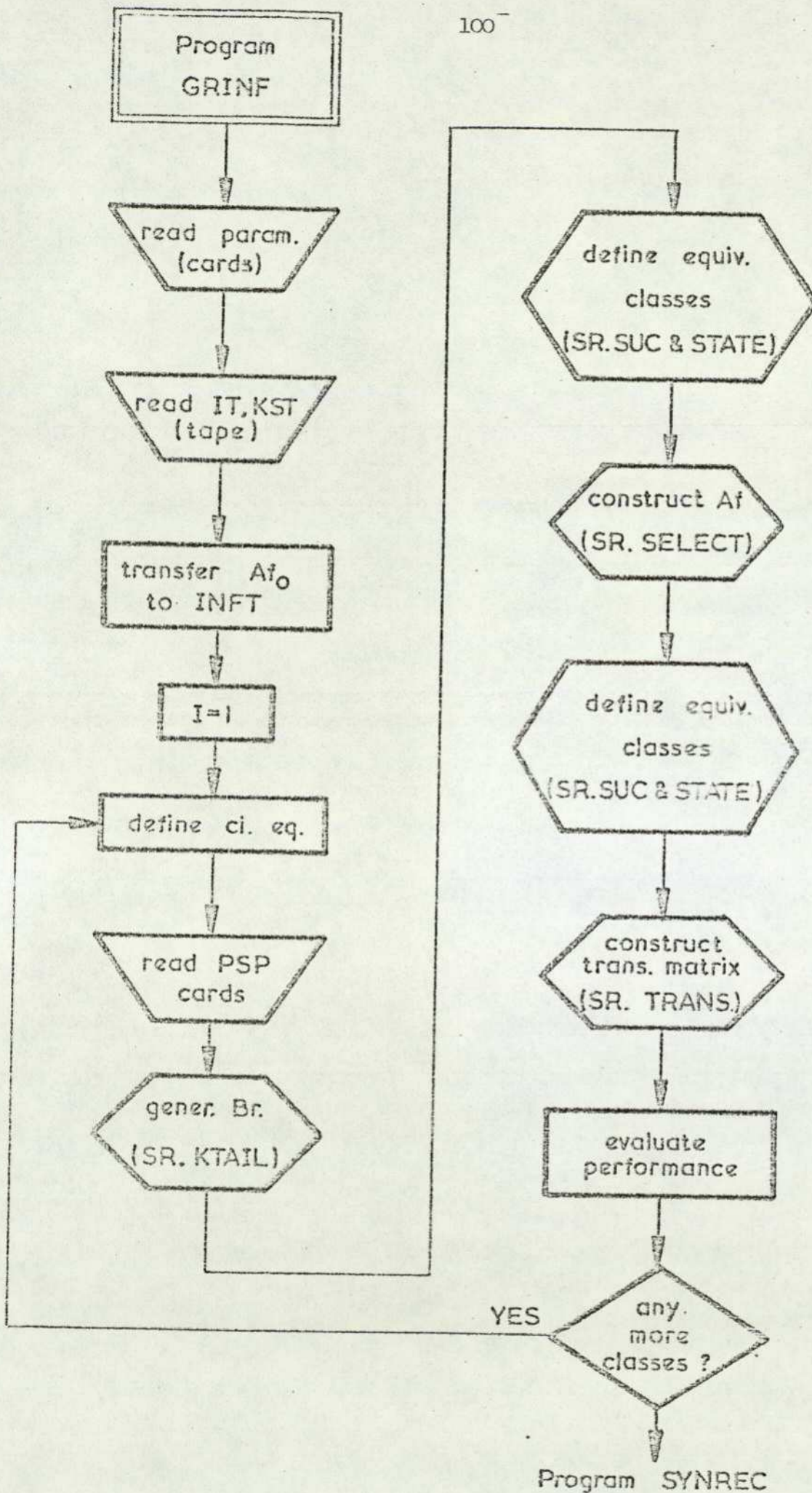


Fig. 5.1 Flowchart of program GRINF

5.4.3.3 Table INFT: is the main table used for discovering the equivalence classes.

5.4.3.4 Table STT: stores the class-vectors $C = C_1 C_2 \dots C_p$ which define the equivalence classes.

5.4.3.4 Table TRANSM: is used to construct and finally stores the transition matrix of the finite-state acceptor (columns $1 \div |\sum_i|$) and the class associated to each state (column $|\sum_i| + 1$).

5.4.4 The main program

The synthesis of a FSA from a training set is implemented by the program GRINF according to the algorithm described in par. 5.3; the structure of the program is outlined in the flowchart shown in fig. 5.2.

5.4.5 Subroutines called by the program GRINF

Since most of the subroutines called in the main program implement simple functions, the present paragraph is intended to provide a brief reference list of the simplest subroutines and to discuss in more detail the subroutines involved directly in the synthesis of the FSAs.

5.4.5.1 AFINF: is called at the beginning of the program to read the set A_{fo} of strings and to store them in the tables IT, KST and also in the first $IAF = |A_{fo}|$ entries of table INFT; the input parameter KSTCR specified the last entry in table KST occupied by the strings in the training set.

5.4.5.2 HASH: was described in par. 4.4.4.

5.4.5.3 ENTER: computes the hashing address HALFA and the CLASS of a STRING of length LS; if the string is not already in table INFT,

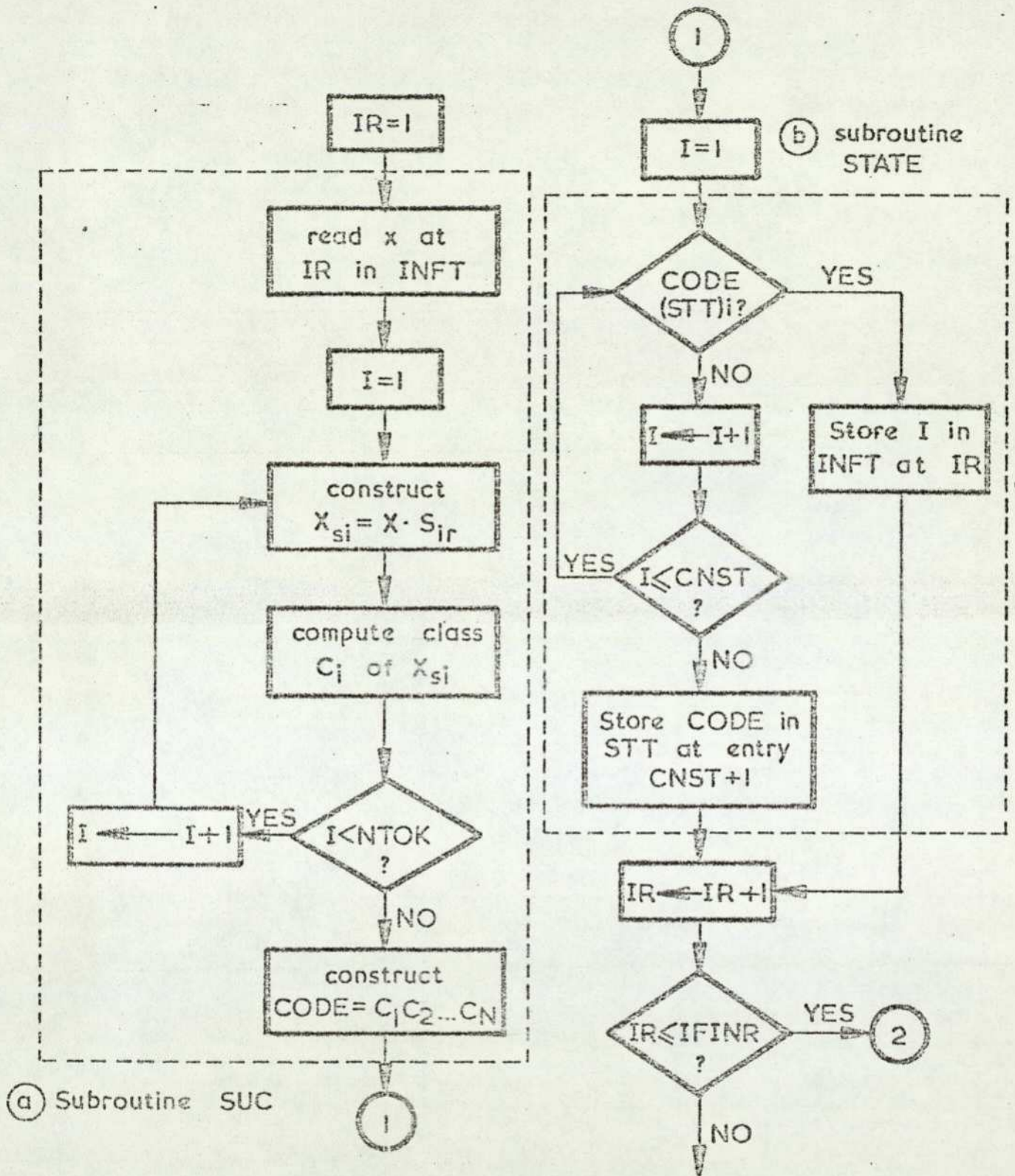


Fig.5.2 Determination of the equivalence classes

writes the two items in this table at entry CRINFT.

5.4.5.4 CONCAT: concatenates a SUFFIX of length LRS to a STRING of length LLS; if the length of the combined string exceeds a maximum M, prints a warning message.

5.4.5.5 REC: computes the CLASS of a STRING of length LS using a FSA with transition matrix TRANSM.

5.4.5.6 KTAIL: extracts all strings of length KLT or less in the pattern class currently processed and stores them in the table TSP.

5.4.5.7 STATE: computes the equivalence class (classes) defined by a class-vector CODE associated to a string x; output parameter is an array NST which stores a number NEXT of equivalence classes found (see flowchart in fig. 5.2.b)

5.4.5.8 SUC: performs the operations required to process the entry IR in the table INFT (see flowchart in fig. 5.2a)

- reads the string x specified by its hashing address.
- constructs and writes the strings x_{sr} in the table INFT at the next free entry, NRINFT
- computes the equivalence class to which the string x belongs and writes it in the table INFT.

5.4.5.9 TRANS: computes the transition matrix of the FSA whose states represent the equivalence classes determined in the main program; in the flow chart which illustrates the operation of the subroutine, shown in fig. 5.3, the syntactic elements are denoted by S_1 and the r-tails of syntactic elements are denoted by S_r .

Basically, the subroutine performs the computations corresponding to the relations (5.6) and (5.7); the main input and output parameters are

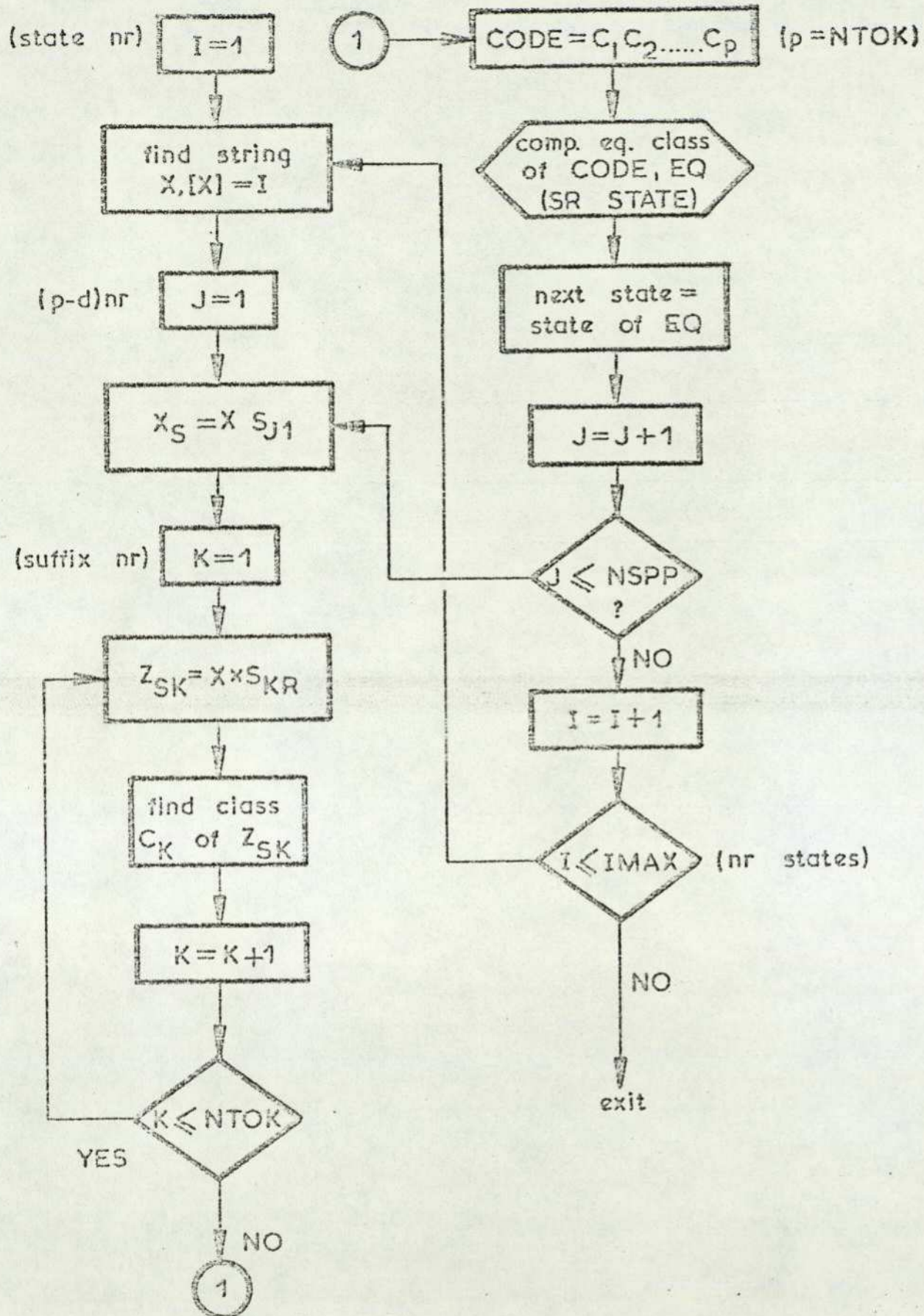


Fig.5.3 Subroutine TRANS

transmitted via common blocks.

5.4.5.10 SELECT: generates the final set A_f by selecting for each equivalence class the shortest representative string and all the prefixes of these strings; finally, stores the set A_f in the table INFT.

5.5 Conclusions

Finally, it must be recalled that the grammars represented by the FSAs constructed in this program are approximate solutions. In terms of set theory, the situation can be illustrated as in fig. 5.4. In this example, there are 4 pattern classes whose clusters in a "syntactic space" are denoted by $K_1 - K_4$. The grammars inferred define languages which can also be represented as clusters of strings, denoted $L_1 - L_4$. In general these clusters do not coincide and sometimes are simply incorrect, as in the case of L_3 , which overlaps K_2 .

The preliminary performance estimation of the recognizers constructed, which is carried out at the end of the program, can provide a better understanding of the syntactic structure of the patterns. If necessary, the designer can add strings to the training set, thus reducing the ambiguities which contribute to a poor performance. A method which can improve the performance of the recognizers, by "accepting" strings outside the clusters, is discussed in the next chapter.

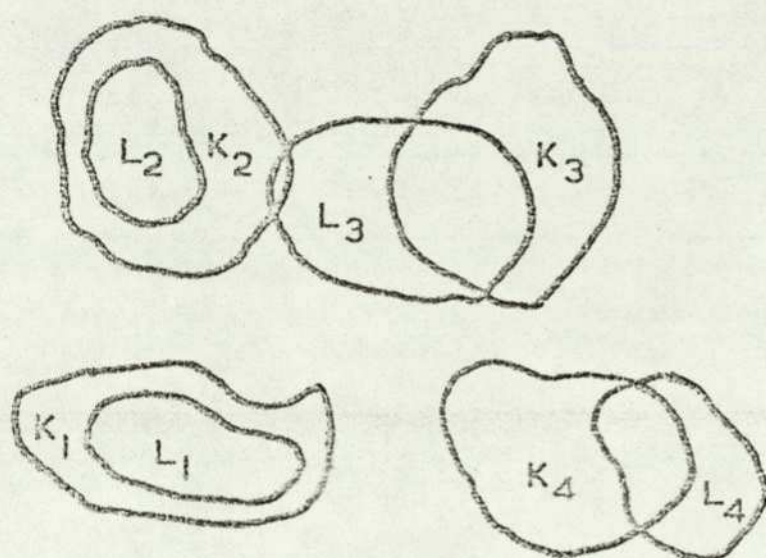


Fig. 5.4 Language approximation

6. CLASSIFICATION OF SYNTACTICALLY IMPERFECT PATTERNS

The grammatical inference method described in the preceding chapter provides only an approximate solution and therefore the performance of a syntactic classifier based directly on the FSAs constructed using this procedure is restricted. Preliminary experiments with real data have confirmed that the number of patterns rejected because of their imperfect structure is unacceptably high. It was therefore decided to investigate the possibility of incorporating in the classifier additional capabilities, to enable it to cope better with distorted patterns.

The literature survey presented in chapter 3 reviewed a few techniques which have been suggested recently for improving the performance of syntactic pattern recognition systems dealing with real life patterns; the concept of error operators, defined in para 3.3, emerged as most promising for the classification of defects in real time. However, the method proposed in the literature for applying this principle to pattern classification problems [18] proved during this investigation to become extremely complicated for a problem of this complexity. A simpler way to use this technique is described by R.A. Wagner [35] in connection with compiler error recovery problems and is based on the concept of edit distances between strings.

The basic contribution made in this chapter is to generalize the concept of edit distances to recursive grammars and to incorporate the formalism introduced in chapter 4 for the description of two-dimensional patterns.

The method can be introduced best by assuming initially that the inferred grammar (G_1) generates a subset of the unidentified language (L) , $L(G_1) \subseteq L$. This is often the case, since regular languages are properly contained in all other classes of languages and the grammar inference process can be continued until an acceptable accuracy is obtained. In a strictly formal sense, the situation can be considered as if the inferred grammar defined a set of perfect strings (patterns), $L(G_1)$, and some of the strings in $L - L(G_1)$ were obtained from perfect strings in $L(G_1)$ corrupted by error operators of the type defined in para. 3.3. Intuitively, this approach is based on the fact that the error corrupted strings maintain a certain resemblance to the strings in $L(G_1)$, which on the other hand are approximations for the unidentified language L .

The problem is to find which strings in Σ^* can be considered as resulting from strings in $L(G_1)$ corrupted by error operators and also to determine if, by accepting these strings, the language $L(G_1)$ is not extended far beyond the language L . These considerations can be expressed formally and quantitatively by introducing the term of "edit distance".

6.1 The concept of edit distance

The present paragraph introduces the idea of applying the concept of syntactic distance for the classification of imperfect patterns, as it emerged during the present study, from an initial attempt to employ directly the error operators technique.

Let R_0 denote a regular expression which defines a set of (error-free) strings. Each string in $L(R_0)$ can be viewed as a point in a linguistic space; all strings in $L(R_0)$ form a cluster in this space.

By applying the error operators, one can derive the regular expression which defines strings corrupted by one error:

$$R_1 = \alpha(R_0) + \beta(R_0) + \delta(R_0) \quad (6.1)$$

The sets R_0 and R_1 are not necessarily disjoint, because some error operators can transform a string in R_0 into another string in R_0 ; the regular expression which defines only imperfect strings is

$$R_{1e} = R_1 \cap \bar{R}_0 \quad (6.2)$$

In a similar manner, one can define the set of strings corrupted by two errors:

$$R_2 = \alpha(R_1) + \beta(R_1) + \delta(R_1) \quad (6.3)$$

As before, the regular expression which defines only the set of strings corrupted by two errors can be written as

$$R_{2e} = R_2 \cap \overline{(R_{1e} \cup R_0)} \quad (6.4)$$

This process can be continued for multiple error operators; the resulting sets of strings can be represented as clusters as shown in fig. 6.1. Using the techniques defined in para 3.3, it is possible to construct a grammar for the language

$$L = L(R_0) \cup L(R_{1e}) \cup L(R_{2e}) \cup \dots \cup L(R_{ke}) \quad (6.5)$$

The first attempt to apply this technique to the present problem revealed certain difficulties:

- a) the grammar of the language L tends to become complicated
- b) if the language $L(R_0)$ was a proper subset of the approximated language L_A , the language L could also include a number of strings

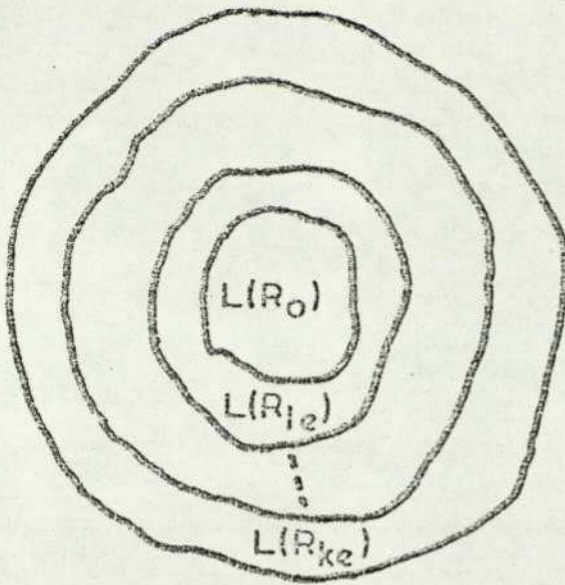


Fig. 6.1 Language defined by error operators

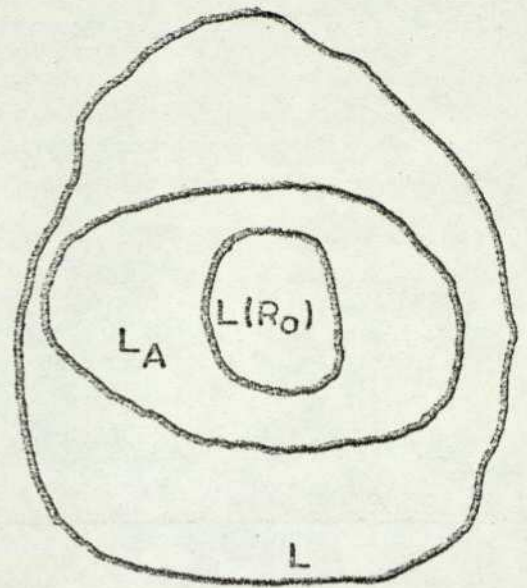


Fig. 6.2 Errors introduced by approximations

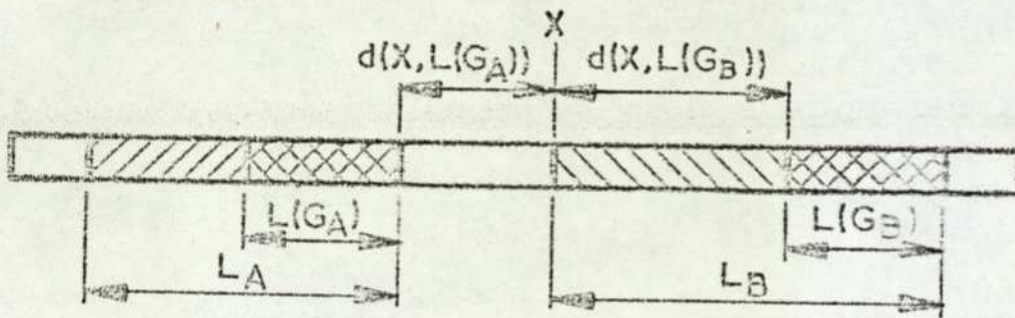


Fig. 6.3 Worst case situation

		1	2	3		M
1	pp					
	cd					
2	pp					
	cd					
NCL	pp					
	cd					

Fig. 6.4 Array IDSP

which do not belong to the language L_A : the overall recognition performance would then be degraded (fig. 6.2).

c) the situation becomes worse if the language L includes strings belonging to another pattern class, described by language L_B .

These difficulties were finally overcome simply by renouncing the idea of constructing explicitly the new language and using instead the concept of edit operations, which are more simple to compute. For each string in $L(R_{ke})$, there exists one correction which will transform that string into a string in $L(R_{k-le})$; this correction is called an edit operation and is the reverse of an error operator:

- a) an insertion corrects a deletion
- b) a deletion corrects an insertion
- c) a substitution corrects another substitution

A number K of successive edit operations is necessary in order to transform strings in $L(R_{ke})$ into error-free strings.

This concept was introduced in [35] to define the distance from a string to a language as the minimum number of edit operations (abbreviated MNEO) necessary to transform the string into a legal string of that language; the distance is 0 for error-free strings.

A simple example will illustrate the application of the MNEO concept to the classification of error-corrupted strings. Consider for the beginning two regular expressions, R_A and R_B , which define the languages L_A and L_B of error-free strings, both over the same set V_T of terminal symbols. Let x be an error-corrupted string:

$$x \in V_T^*, |x| = \ell; \quad x \notin L_A \text{ and } x \notin L_B$$

Assume that one can find two integers m and n such that

$$x \in L(R_{Ame}) \quad \text{and} \quad x \in L(R_{Bne})$$

$$m < n \leq 1$$

The MNEO distances from the string x to the languages L_A and L_B , respectively, are

$$d(x, L_A) = m$$

$$d(x, L_B) = n$$

If the string must be classified in one of the classes A or B, it will be considered in class A, because $m < n$. The solution is ambiguous when $m = n$ and the decision must be taken on the basis of the frequencies with which the strings occur.

In practice the problem can be more complicated, because the grammars which are inferred are usually approximate solutions: the type of situation which can create difficulties is illustrated graphically in fig. 6.3. The languages L_A and L_B are approximated by the languages $L(G_A)$ and $L(G_B)$, respectively. It may happen that for certain strings the following "worst-case situation" occurs:

$$x \in L_B, \text{ but}$$

$$d(x, L(G_A)) < d(x, L(G_B))$$

and therefore the string x will be assigned incorrectly to class A. The number of misclassifications can be reduced by introducing weighted distances ($w_A d_A$ and $w_B d_B$), a common procedure in statistical pattern recognition.

The application of the MNEO concept improves, in most cases significantly, the recognition performance of a FSA; its applicability depends on the existence of an efficient algorithm for computing this distance. The algorithm used in the thesis is an adaptation and

extension of an algorithm proposed in [35] for correcting errors in programming languages; the original formulation of this algorithm is presented briefly in the next paragraph.

6.2 Algorithm for the Computation of MNEO

Finite state automata have two special properties which can be used to develop efficient algorithms for computing the MNEO from corrupted strings to regular languages:

- 1) the current state contains all the information a FSA retains about the input strings.
- 2) the FSA can be in only a finite number of states, regardless of the length of the input string.

These two properties permit the computation of a "correction table" which depends only on the regular language and, once computed, can be used to calculate the MNEO for all possible input strings.

The algorithm for the computation of MNEO is based on a recursive function which is updated for each new symbol of the input string, using the correction table.

6.2.1 Computation of MNEO

Consider a FSA = $(Q, \Sigma, \delta, q_0, A)$ and a string $x \in \Sigma^*$. Let $x \langle j \rangle$ denote the j th symbol of string x and $F(j, S)$ equal the MNEO needed to change the first j symbols of x into some string y which will force fsa into state S .

It has been shown [35] that $F(j, S)$ can be computed recursively with the formula:

$$F(j, S) = \min_T \left\{ F(j-1, T) + V(T, S, x \langle j \rangle) \right\} \text{ if } j \geq 1$$

$$F(0, S) = \begin{cases} 0 & \text{if } S = q_0 \\ \text{otherwise} \end{cases} \quad (6.6)$$

where: $V(T, S, c)$ is equal to the MNEO which will change the single symbol c into a string $w(T, S)$ which will cause a transition from T to S

$$c \in \Sigma \cup d, \quad d \notin \Sigma$$

Finally, for the complete string x ,

$$\begin{aligned} \text{MNEO} &= \min_{S \in A} F(|x|, S) \end{aligned} \quad (6.7)$$

The table $V(T, S, c)$ depends only on the regular language accepted by FSA; it is computed once, stored and used for all possible input strings.

6.2.2 Computation of $V(T, S, c)$

It has been found that the table $V(T, S, c)$ can be conveniently expressed as a function of:

- 1) $P(T, S)$ - the length of the shortest path from state T to state S
- 2) $L(T, S, c) = \begin{cases} 1 & \text{if symbol } c \text{ appears in one of the shortest paths from state } T \text{ to } S \\ 0 & \text{otherwise} \end{cases}$

The matrix $V(T, S, c)$ can be computed as follows:

$$V(T, S, c) = \begin{cases} 1 & \text{if } P(T, S) = 0 \\ P(T, S) - L(T, S, c) & \text{otherwise} \end{cases} \quad (6.8)$$

6.2.3 Computation of $P(T, S)$

Let $P^k(T, S)$ denote the length of the shortest path from state T to state S , passing only through states numbered $1, 2, \dots, k$.

$P(T, S)$ is computed recursively as following:

$$P^0(T,S) = \begin{cases} 1 & \text{if there is an arc connecting directly } T \text{ to } S \\ 0 & \text{otherwise} \end{cases} \quad (6.9)$$

$$P^{k+1}(T,S) = \min \{ P^k(T,S); P^k(T,k+1) + P^k(k+1,S) \} \quad (6.10)$$

$$P(T,S) = P^t(T,S) \text{ where } t = |Q| \quad (6.11)$$

6.2.4 Computation of $L(T,S,c)$

$L(T,S,c)$ is also computed recursively:

$$L^k(T,S,c) = \begin{cases} 1 & \text{if symbol } c \text{ is accepted by some arc along path} \\ & P^k(T,S) \\ 0 & \text{otherwise} \end{cases} \quad (6.12)$$

$$L^{k+1}(T,S,c) = \begin{cases} L^k(T,k+1,c) \vee L^k(k+1,S,c) & \text{if } C > D + E \\ L^k(T,k+1,c) \vee L^k(k+1,S,c) \vee L^k(T,S,c) & \text{if } C = D + E \\ L^k(T,S,c) & \text{otherwise} \end{cases} \quad (6.13)$$

where: $C = P^k(T,S)$, $D = P^k(T,k+1)$, $E = P^k(k+1,S)$ and $\vee$ is defined as follows:

$a \vee b = 0$ if $a = b = 0$; $a \vee b = 1$ otherwise

Finally:

$$L(T,S,c) = L^t(T,S,c) \text{ where } t = |Q| \quad (6.14)$$

6.2.5 Time and space complexity of the algorithm

Relations (6.6) and (6.7) show that the time and space complexity of the MNEO algorithm are characterised as follows:

$$T_C \sim |x| \cdot |Q|^2 \quad (6.15)$$

$$S_C \sim \left| \sum \right| \cdot |Q|^2 + |Q| \quad (6.16)$$

where:

$|Q|$ - the number of states

$|\Sigma|$ - the number of input symbols

$|x|$ - the length of the input string

Relations (6.15) and (6.16) reveal that the improved recognition performance is obtained at the expense of a considerable increase in complexity, compared to the simple FSA recognizer, for which (see par. 2.1.4)

$$T'_C \sim |x| \quad (6.17)$$

$$S'_C \sim |\Sigma| \cdot |Q| \quad (6.18)$$

Nevertheless, it will be shown in chapter 9 that the hardware implementation of the MNEO classifier satisfies the requirements of the present problem.

The algorithm described can be applied to both deterministic and nondeterministic FSAs, therefore all solutions provided by the grammar inference algorithm are acceptable candidates for the MNEO classifiers.

6.3 Generalized MNEO algorithm for the classification of multi-class two-dimensional patterns

The application of the original MNEO algorithm to the classification of multi-class two-dimensional patterns required the following extensions to be incorporated in the algorithm employed in the present study:

1) recursive productions, of the form $A \rightarrow aA$, which were not considered in the original version, had to be included in the correction tables.

2) the algorithm had to be extended to enable the handling of strings of syntactic elements, consisting of pattern primitive/composition descriptor pairs.

3) a method had to be developed for the classification of patterns in multiple - classes, each class (language) described by a separate grammar and having its own recognizer.

6.3.1 Handling recursive productions

The first problem has been solved by taking into account the fact that recursive productions in a regular grammar lead to loops around the corresponding states of the FSA recognizer. If the recursive production is $A \rightarrow aA$ and the associated state of the FSA is A, the symbol "a" will determine a cyclic transition from A to A.

By applying the definitions of $P(T,S)$, $L(T,S,c)$ and $V(T,S,c)$ to this situation, the corresponding terms become

$$P(A,A) = 0$$

$$L(A,A,a) = 1$$

$$V(A, A,a) = 0$$

The last equality reveals that no edit operations are required in this case, since the symbol "a" itself determines a transition from state A to state A. Consequently, the generalized form of relation (6.8) will become

$$V(T,S,c) = \begin{cases} 0 & \text{if } P(T,S) = 0 \text{ and } L(T,S,c) = 1 \\ 1 & \text{if } P(T,S) = 0 \text{ and } L(T,S,c) = 0 \\ P(T,S) - L(T,S,c) & \text{otherwise} \end{cases} \quad (6.19)$$

6.3.2 Processing strings of syntactic elements

The second problem was solved, in principle, by interpreting the syntactic elements as symbols, in the same manner as for the grammatical inference algorithm: by assigning a separate symbol to each distinct syntactic element, the MNEO algorithm becomes directly applicable to the present problem. However, additional factors have to be considered in the present case, because this procedure implies simplifications which could reduce the classification performance of the generalized algorithm.

To illustrate the nature of the problem, consider a string $x = a_1a_2a_3a_4a_5$ and assume a substitution-type error has changed the symbol a_3 to b , so that $x' = a_1a_2ba_4a_5$. The MNEO distance from x' to x is 1, for any symbol $b \neq a_3$. While this assumption is justified for strings of symbols, it can introduce errors in the present case, because the pattern primitives and composition descriptors are in fact second-level symbols, computed from strings of 0's and 1's (the trigger samples). Looking at tables 2 and 3, it is evident that not all substitution errors imply the same degree of distortion of the corresponding scan sections. The problem is complicated even further by the fact that the grammar inference algorithm produces approximate solutions, therefore the transitions between the states of the inferred FSAs have not been determined with absolute precision. This suggests incorporation of probabilistic or fuzzy concepts in the structure of the syntactic classifier. At this point it is important to realize that these concepts can be incorporated directly into the operation of the MNEO algorithm, by extending the domain of the

elements $V(T,S,c)$ from the set of integers to the set of real numbers; this can be effectively implemented by assigning numbers in the range from 0 to 1 to the binary elements $L(T,S,c)$. Potentially, this is a powerful generalization of the MNEO algorithm, but has not been incorporated in the version used in this thesis, which is based on the simplified assumption that syntactic elements can be treated as simple symbols in strings.

6.3.3 Classification of multi-class patterns

The last question was concerned with the application of the MNEO algorithm to the classification of multi-class patterns, when each class has assigned its own recognizer. This problem has been solved by computing separately the MNEO distance from the pattern to be classified to each class (language). The distances are updated for each syntactic element and after encountering the last syntactic element, the defect is assigned to the class (language) for which the MNEO distance is minimum, according to rel. (6.7).

This procedure requires each syntactic element to be interpreted and assigned an identifier according to the language (class) currently considered, because the set of terminal syntactic elements (identifiers) is defined for each language separately. As a consequence, the same string of syntactic elements (pattern) is represented as different strings of identifiers in the process of computing the MNEO distances to different classes.

Patterns are rejected when they are equidistant to two or more languages (classes) and also when they are too distorted.

Experience has indicated that the classification becomes unreliable when the MNEO distance from a pattern to a language exceeds a certain ratio (ex. $1/3$) of the length of the string representing that pattern; in these cases, the class of the defect is declared unknown (reject).

6.4 Program SYNREC for syntactic recognition of imperfect patterns

The program SYNREC performs the syntactic classification of patterns represented by strings of pattern primitives and composition descriptors, according to the algorithm presented in par. 6.2, using as recognizers the FSAs constructed by inference from a training set of patterns.

The input to the program consists of:

1. the transition matrices of the FSAs (arrays TM)
2. the list of syntactic elements used in each pattern language (arrays IDSP and NSPMAX).
3. the representation of the patterns to be classified as strings of syntactic elements (arrays IT and KST).

The output of the program describes the performance of the syntactic pattern recognition system in terms of:

1. the global recognition performance (PERF) -the ratio between the number of patterns classified correctly (ICOR) and the number of patterns classified (IREAD).
2. the confusion matrix (CONFM), which specifies the nature of the misclassifications.

6.4.1 Data bases

This section describes the functions of the new data bases introduced in this program; data bases discussed previously are not mentioned again.

- 6.4.1.1 VTSC: three-dimensional arrays which store the correction tables $V(T,S,c)$ computed for each pattern language.
- 6.4.1.2 IDSP: a three-dimensional array which defines the syntactic elements used in each language and the codes (identifiers) assigned (fig. 6.4).
- 6.4.1.3 NSPMAX: a linear array which specifies the number of syntactic elements used in the definition of each language.
- 6.4.1.4 STRING: a two-dimensional array which stores the representation of the pattern currently classified as a string of syntactic elements
- 6.4.1.5 DIST: a linear array which stores the MNEO distances from the string (pattern) currently classified to each pattern language.
- 6.4.1.6 LINSTR: a linear array which stores the strings of identifiers resulted by applying the table look-up decoding procedure to STRING.

6.4.2 The main program

The organisation of the program SYNREC is represented in the flow chart depicted in fig. 6.5, which outlines the main functions of this program.

- 6.4.2.1 reads the definitions of the recognizers, as specified by the transition matrices TM.
- 6.4.2.2 computes the correction tables VTSC associated to the transition matrices, by calling the subroutine MINEDIT, discussed

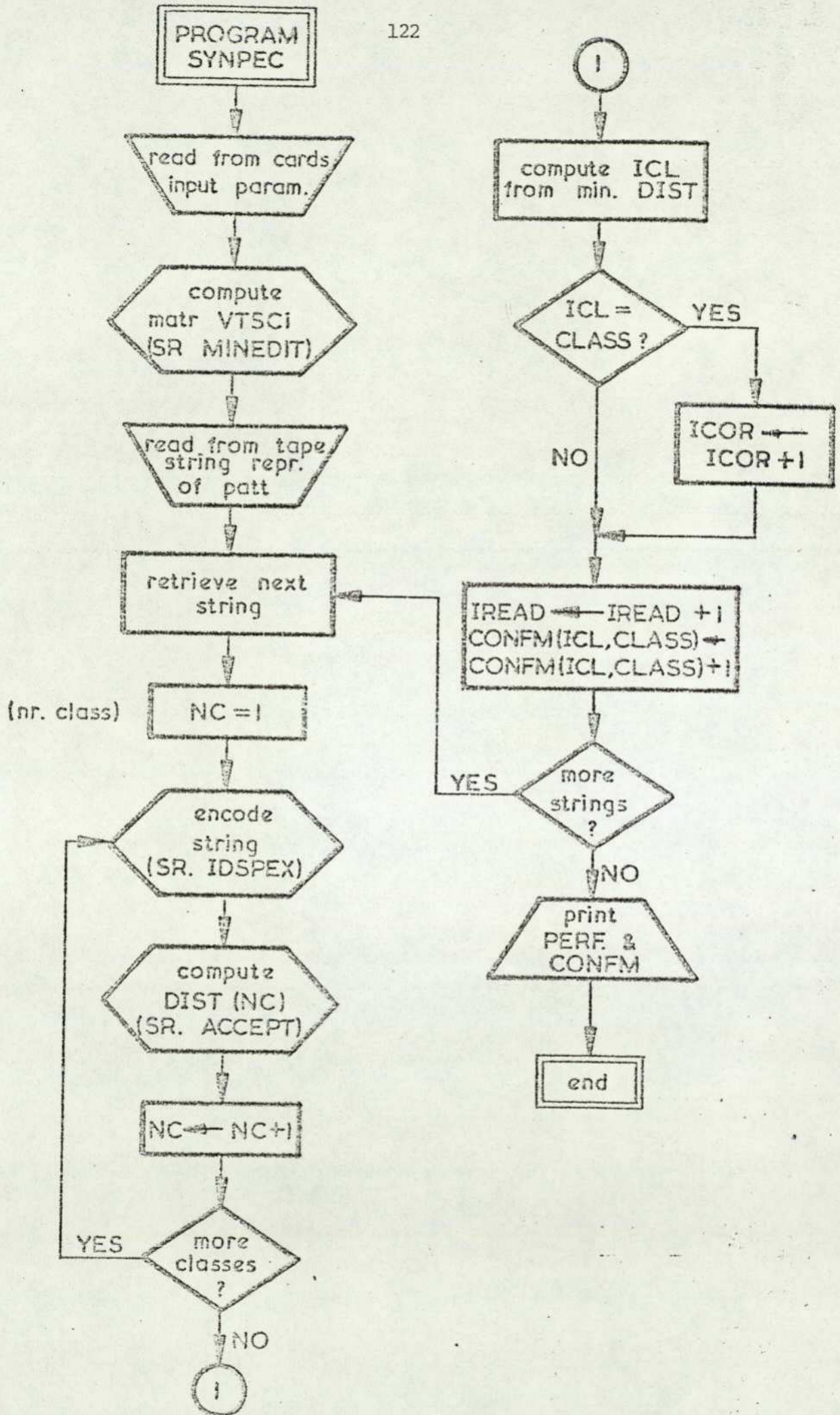


Fig. 6.5 Flowchart of program SYNREC

in the next section

6.4.2.3 reads from magnetic tape the representation of the defects to be classified as strings of pattern primitive/composition descriptor pairs and stores them in the tables IT and KST

6.4.2.4 retrieves sequentially each string and performs the following operations:

(a) decodes each STRING by calling the subroutine IDSPEX, discussed in the next section, which produces a string LINSTR, for each defect class separately.

(b) computes the MNEO distances from the string LINSTR to the corresponding defect language, for each class separately, and stores the computed values in the array DIST.

(c) finds the minimum value in the array DIST and assigns the string LINSTR and the corresponding pattern, to the class which defined that minimum distance.

(d) updates the variables which characterise the global recognition performance (ICOR, IREAD) and the confusion matrix (CONFM).

6.4.2.5 finally, prints the results (PERF, CONFM).

6.4.3 Subroutines

6.4.3.1 Subroutine IDSPEX: performs the decoding operations which are required in order to carry out the transformation from a string of syntactic elements to a string of identifiers, as specified by rel (4.9) and (4.10). Input parameters: arrays IDSP and NSPMAX, the pattern primitive (SYM)/composition descriptor (PRED) pair and the CLASS for which the decoding operation is required. Output parameter is the computed identifier (IDENT).

6.4.3.2 Subroutine MINEDIT: performs the computations required to construct the table $V(T,S,c)$ from the transition matrix TM, as defined by rel. (6.8) to (6.15).

Input parameters: the transition matrix TM, the number of states (NS) and the number of input symbols (identifiers, or pattern primitive/composition descriptor pairs).

Output parameter: the correction table VTSC associated to the transition matrix TM.

Arrays PNEW, POLD, LNEW and LOLD are auxiliary tables used to store intermediate results.

The computations involved are ordinary operations with matrices, defined by the equations mentioned above.

6.4.3.3 Subroutine ACCEPT: computes the MNEO distance from a string of identifiers, LINST, to a language described by its correction table, VTSC and the associated parameters.

Input parameters:

a) the matrix VTSC, the number of states of the recognizer (NS), the number of input symbols (MS), the number of final states (NFST) and the list of final states (FIST).

b) the string of identifiers (STRING) and its length (LS)

Output parameter: the MNEO distance (MIN)

The subroutine performs the computations defined by eq (6.6) and (6.7), which involve ordinary operations with matrices; a flowchart of the subroutine ACCEPT is shown in fig. 6.6.

The complete program is listed in [34].

7. ACQUISITION OF A DATA BASE

7.1 Introduction

The previous chapters of the thesis have presented the theoretical methods which have been developed in order to realize a syntactic pattern recognition system for real-time defect classification. Since the entire research has been directed to meet the severe requirements of a specific application, it was considered that the validity of the theoretical methods developed in the thesis can be effectively evaluated only by determining the performance of the system with data specific to that application.

Acquisition of a comprehensive data base is an essential requirement, which occurs in all pattern recognition applications related to automatic surface inspection which are being investigated at the Instrument Systems Centre. Considering the complexity of this problem and its general character, it was decided to dedicate a significant part of the work reported in this thesis to the task of building an automated data gathering system (or "test rig").

Basically, the test rig consists of a large table, driven by a stepping motor in what would be the strip direction in an on-line situation. The sheet of tinplate to be scanned is placed on the table, above which is mounted the scanner. The scanner output signal is digitized and stored in a minicomputer, where it is formatted and finally written on magnetic tape. The minicomputer controls the whole operation. The operator is required merely to place the sheet on the table and to introduce the sheet identifier conversationally into the teletype.

Since the operation of the test rig is not relevant to the main results of the present study, a detailed presentation of this topic has been deferred to the last sections of this chapter. The experience gained during the design, construction and exploitation of the test rig is valuable for the development of new test rigs, currently undertaken at the Instrument Systems Centre.

The data base of tin-plate defects which has been gathered using this test rig is described in the next section.

7.2 Description of the data base

The data base used to evaluate the performance of the syntactic pattern recognition system was acquired by scanning a number of 80 1m x 1m tinplate sheets with defects, sorted by visual inspection at BSC. The process of data gathering was completed, using the automated test rig, in 2 days and resulted in a batch of 8 magnetic tapes storing the SIRA data.

The defects have been identified and classified by visual examination, in cooperation with SIRA and BSC, in the following classes:

- 1) laminations: a whitish linear defect caused by lamination in the steel base being shown up after tinning; a steel defect.
- 2) tin stripping: longitudinal scratch like defects caused by build up on rolls in the E.T.L. line.
- 3) edge strains: a strain mark at the edge of the strip, perpendicular to the rolling direction, of very low contrast.
- 4) five stand ring: a linear defect parallel to the edge of the strip caused by a build up on the rolls.

- 5) scratches: bright scratches along the rolling direction.
- 6) oil/grease spots: oval shaped defects similar in appearance to a finger print; caused by tin being deposited on a grease which has come into contact with the steel base metal.
- 7) dirty surface: a defect giving a speckled appearance over large areas of the strip surface
- 8) dents: physical damage
- 9) bruise marks.

Videoprints of typical defects for each class are shown in Appendix I.

The distribution of defects over the classes represented in the data base is given in table 8. Considering the variability of the patterns within a class, the number of defects and classes, it may be concluded that the data base reflects realistically the nature of the problem investigated.

TABLE 8

nr. cr.	Defect class	nr. of samples
1	lamination	20
2	tin stripping	14
3	edge strains	20
4	five stand ring	18
5	scratches	36
6	oil/grease spots	32
7	dirty surface	6
8	dents	15
9	bruise marks	8

The magnetic tapes written at SIRA were processed at ULCC on the CDC 7600 computer and videoprints were generated on microfilm for subsequent comparison with the physical sheets in order to locate and define the classes of the defects. During this process, certain limitations of the scanner became evident:

1. the scanner is insensitive to low contrast defects which extend over a large area in the transversal direction (for ex: edge strains are not detected).
2. nonflatness of the sheet manifests itself as a large defect in the corresponding area; this is not an important shortcoming, since on-line the strip is stretched and nonflatness does not occur.
3. certain small defects are missed or appear only on the off-specular videoprint.

Some of these problems have been overcome by using signal processing techniques based on matched filters [3]. The videoprints used throughout the thesis were obtained by applying this procedure.

However, it must be stressed that while the present videoprints are satisfactory representations of the defects on the surface of the sheets, the performance of the pattern recognition system can be improved by enhancing the quality of defect representation in several ways:

- a) by considering the off-specular videoprints in combination with the specular videoprints
- b) by using the analog information, in addition, when different types of defects produce identical patterns on the video - prints (for ex: laminations and five stand rings)
- c) by employing more sophisticated delineation methods.

7.3 Functions and operation of the test rig

The remaining sections of the present chapter contain a description of the test rig, concentrating on the contribution of this author, which consists of the concept, design and implementation of the computer interfaces (to the magnetic tape and the electromechanical parts of the test rig) and the overall computer control, including development of the programs in assembler language. A complete description of the test rig is given in the report by L. Norton-Wayne, V. Popovici and W.J. Hill [36].

The operation of the test rig is controlled by a PDP-11 minicomputer which forms a communication centre to which all the other components of the system are connected (fig. 7.1). The human operator inserts a few commands via a teletype terminal, which is also used to display warnings of any error conditions, although most of these are then corrected automatically by the computer, without interruption to the data gathering process. The operator is required merely to place the sheet of tinplate in a starting position on the movable table and to introduce the sheet identifier conversationally into the teletype. The computer controls everything else required to complete the scan and record the data; the complete PAL assembler program which has been developed is given in [36].

In the initial phase, the operator switches on the test rig, loads and starts the program. If a complete cycle of operations is defined as the operations required to scan a sheet, one can distinguish two phases within a complete cycle:

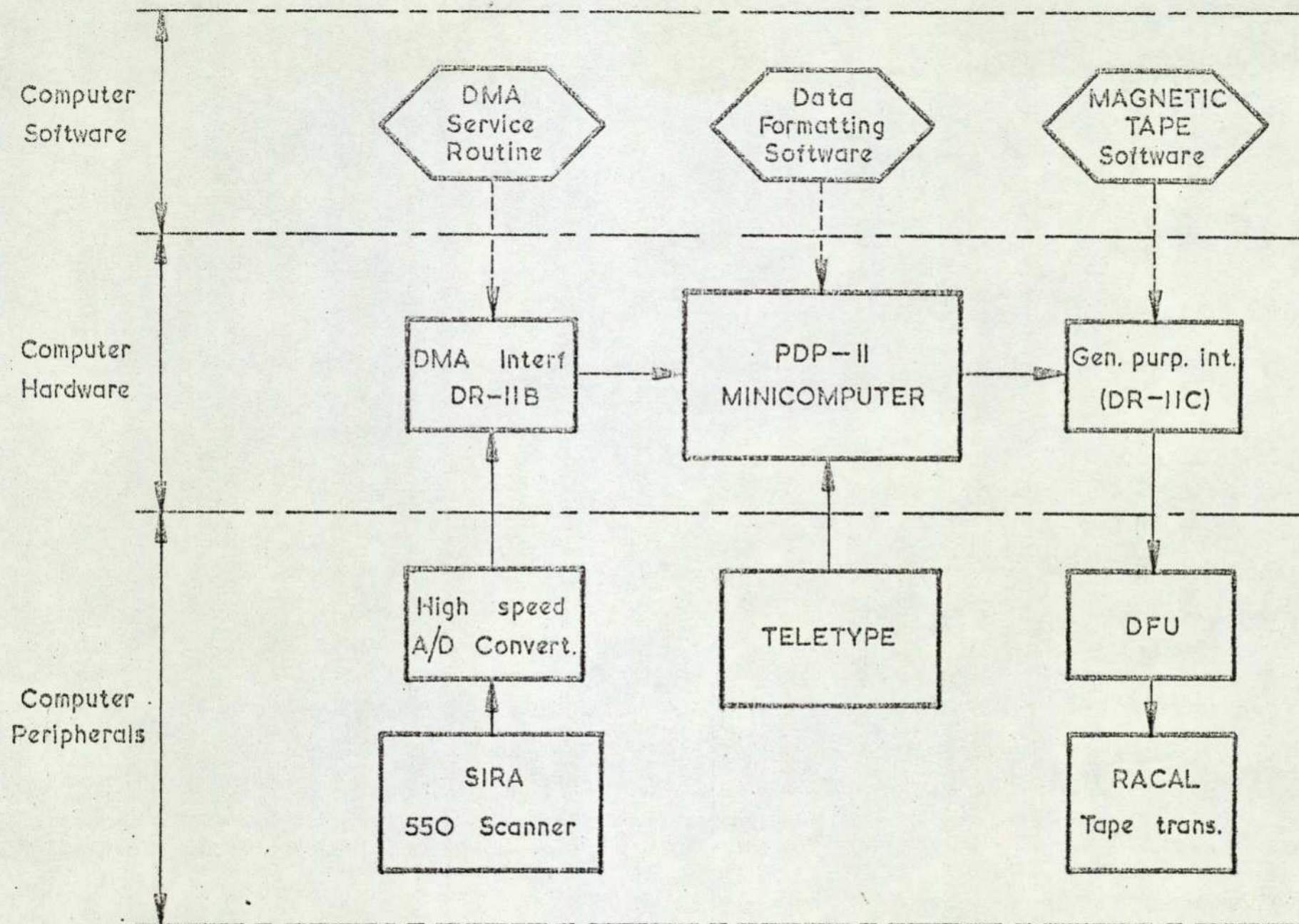


Fig.7.1 Structure of test rig

7.3.1 Phase no. 1 - operations performed by the operator (as a response to messages generated by the program):

- a) Placing a new sheet on the table.
- b) Providing the information required conversationally via the teletype; usually the operator specifies only the sheet identifier, but for debugging purposes, the following additional options can be selected from the teletype:

- the type of scans to be taken (specular, off-specular or both, in that order)
- whether single scans or automatic advances of the table are required.

7.3.2 Phase no. 2 - operations performed automatically

- a) The stepping motor which moves the table is advanced until the computer senses the "sheet present" signal produced by the scanner.
- b) At this stage the relay for selecting specular/off-specular scans is switched to the appropriate position and, after testing that the contact was made correctly, the operation of the DMA unit is initialized.
- c) The DMA interface takes one scan and transfers a block of digitized data to the computer memory, where the block of data is reformatted and converted in CDC-7600 compatible form.
- d) Finally, the sheet identifier, the number of samples, the scan type and the reformatted block of data are written as a constant length block (3000 bytes) on magnetic tape, thus completing the subcycle of operations for one scan.

Before a new scan is started a number of checks are carried out by the program:

a) if an End of Tape signal is detected after writing the data on tape:

- the tape is rewound to the end of the preceding block of data

- an end of information mark is written on the tape

- a message is typed at the teletype indicating a new tape to be loaded by the operator

- when resuming the operation, the last scan is taken again as the first scan of the new tape.

b) if both types of scans are required (the normal case), the off-specular scan is taken by switching the corresponding relay, before advancing the stepping motor.

c) if single-scan operation has been specified, a message indicating the end of operation is typed and the computer halts.

d) if automatic advance is required (the normal case), the stepping motor is advanced with the specified increment and the "sheet present" indicator is tested:

- if the beam is still on the sheet, a new cycle is started.

- if the beam is off the sheet, a reverse command is sent to the stepping motor and the table is moved in the reverse direction until the beam passes over the leading edge of the sheet; at this stage a message is typed, requesting a new sheet to be positioned; this is the end of the scanning process for one sheet. -

7.3.3 Error conditions

Three types of intermittent errors can occur in normal circumstances, without implying a hardware fault:

1. a parity error is detected after writing a block of data on magnetic tape; in this case a message is typed, the tape is rewound to the end of the preceding block and the block rewritten (edited) automatically; frequent parity errors indicate that the tape being used is bad or that the read/write heads must be cleaned.
2. the speed of data transfer from the ADC exceeds the speed of the computer memory; in this case a message is typed and the scan automatically retaken, after sending a reset signal from the DR 11 C interface to the DMA unit.
3. the number of samples transferred is too large and exceeds the capacity of the reserved memory area; a message is typed and the computer halts (clock frequency must be adjusted).

7.4 Test Rig Control

This paragraph describes the technical solutions which were developed for implementing the functions of the test rig presented in the preceding paragraph. At the time when the work reported in this chapter started, the data gathering system in existence at SIRA Institute consisted of the scanner, the high-speed A/D converter, the DMA interface and the PDP-11 minicomputer; the scanning process was performed manually and the data was punched on paper tape. The construction of the automated test rig required the following major facilities to be added to the system:

- 1) a tape unit for the output of data and the corresponding interface to the computer
- 2) a table advanced by stepping motor for scanning the sheets and the corresponding interface.
- 3) the minicomputer software required for controlling the automatic operation of the system.

These problems are discussed briefly in the following sections.

7.4.1 PDP-11 interface to the tape unit

Economic considerations dictated the purchase of a RACAL T7000 tape transport, together with the corresponding RACAL data formatting unit (DFU) [37], rather than a standard DEC-tape. This decision imposed the design and construction of a special interface to the PDP-11 computer. A detailed analysis of the speed requirements indicated that a DR-11C general purpose interface [38] and a user-designed board, containing a few integrated circuits, (flip-flops, monostables and line drivers) provided an economical solution. The interface was programmed in PAL for reading/writing 7 track, 556 bpi magnetic tapes, in an IBM compatible format.

A block diagram of the interface is shown in fig. 7.2. The DR-11C interface provides the logic for program - controlled parallel transfers of 16 bit data between the DR-11C and DFU, which provides data encoding/decoding, deskewing and error detection, together with the tape motion control.

The DR-11C is intended to control simpler devices and therefore its control and status register provides two control

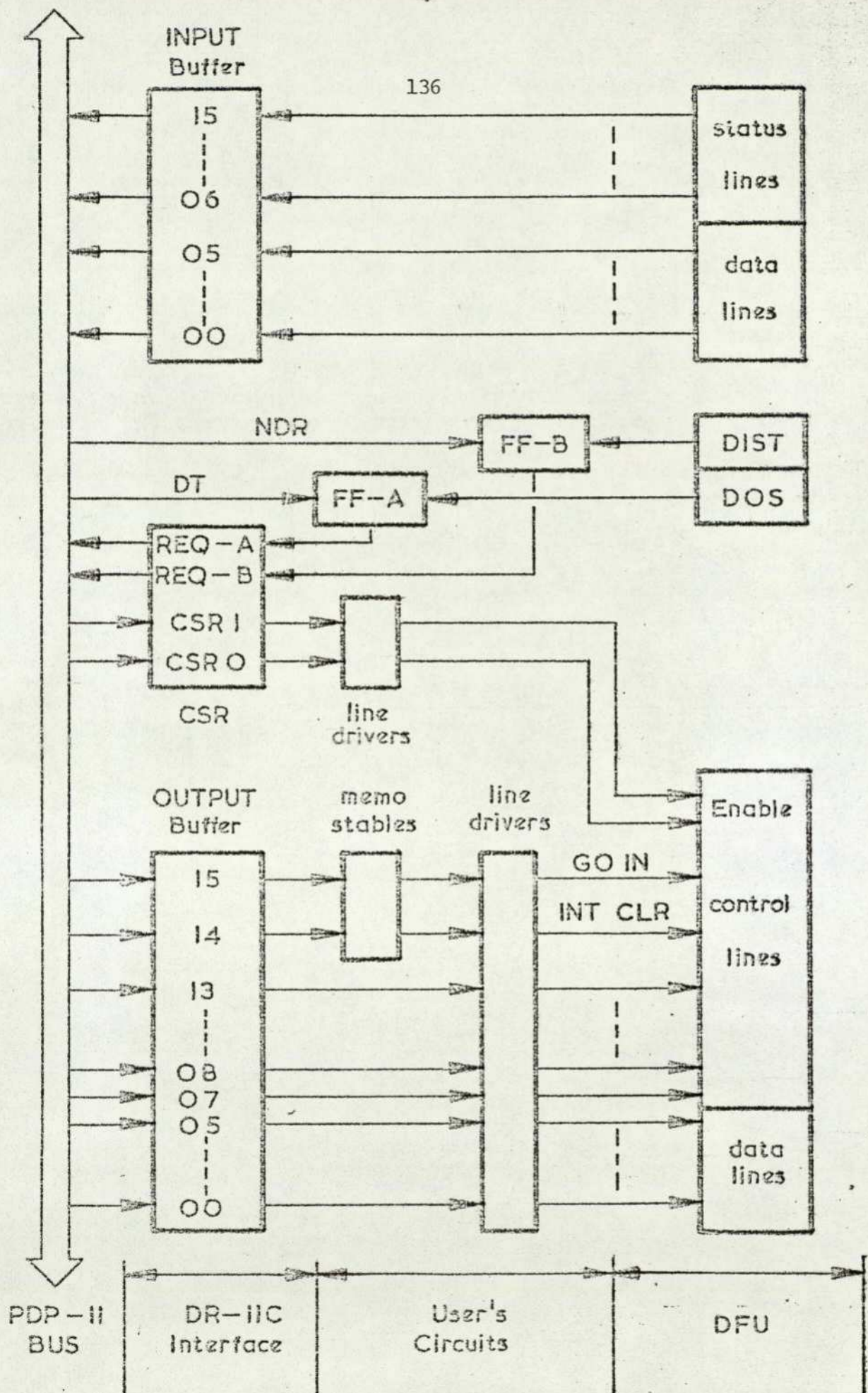


Fig.7.2 Block diagram of the interface between computer and tape unit

signals only, whereas the data paths (16 bits) are wider than the number of bits which have to be transferred (6 bits). This was exploited for assigning additional control and status signals to the unused bits of the output/input lines respectively; it was this unorthodox assignment of functions that made possible the use of a simple DR-11C interface for driving a complicated peripheral.

7.4.1.1 Signals Assignment of DR-11C Output Buffer

The lower byte of this register is used for data output: 6 bits (00-05) are transmitted to the corresponding inputs of the DFU via line drivers. Bit 06 is not used and bit 07 is normally zero, except for the last byte of the block of data to be written on tape, for which it is set to one. Thus bit 07 supplies the LW("last word") signal required by the DFU to generate the interblock gap on tape.

Bits 08 - 13 are used to set the control levels for the DFU and bits 14 - 15 are applied to the inputs of two monostables in order to generate the reset pulse (INIT CLR) and the stroke pulse (GO IN) for the DFU, via line drivers.

7.4.1.2 Signal Assignment of DR-11C Control and Status Register (CSR)

The control signals CSRO and CSRI are used to enable/disable the DFU. The INT EN A and INT EN B bits are set under program control to enable/disable the read and write interrupt flip-flops A and B, respectively. The read/write strobe pulses from the DFU (DOS/DIST) set the interrupt flip-flops A and B

included on the user-designed board, thus generating the interrupt requests REQ A and REQ B, respectively, if the corresponding enable bits are set.

The interrupt service routines transfer the next byte between the OUTPUT/INPUT buffers and memory; these transfers are accompanied by the signals NDR ("new data ready")/DT ("data transmitted"), respectively, which inform the device that the next byte can be transferred, and simultaneously reset the interrupt flip-flops A and B. The right speed of data transfer is therefore ensured by the DOS/DIST synchronizing signals, supplied by the DFU at a frequency dictated by the programmed recording density, in this case 556 bpi.

7.4.1.3 Signals Assignment of DR-11C Input Buffer

Bits 00-05 are the data input lines from the DFU and bits 06-15 carry status levels which are tested by the program as necessary. All input lines are terminated by resistive networks.

7.4.1.4 Principle of Operation of the Interface

Before a new command is issued, the DFU is reset by sending the INIT CLR pulse. The pattern of control bits required by the new command is then set in the corresponding bits of the OUTPUT buffer and strobed in by generating the GO IN pulse. This initiates the transfer of data between the computer and the tape unit, which is performed by the device service routines, called when the DFU generates an interrupt request.

After the execution of the command is completed, the status bits are checked for error conditions.

The operation of the automated test rig required the development of subroutines for implementing the following commands by the tape unit:

- a) write a block of data
- b) read a block of data
- c) read reverse a block of data
- d) edit a block of data
- e) write an end of information mark.

The corresponding subroutines have been included in [36].

7.4.2 PDP-11 interfaces to the test rig

The extremely wide range of the information transfer rate between the minicomputer and the test rig could be economically accommodated using two interfaces:

1. a DR-11B general purpose direct memory access interface, which has been implemented by the SIRA Institute to transfer data (analog samples and trigger bits) from the scanner to the memory at a high rate.
2. a DR-11C interface which controls the slow electromechanical parts of the test rig, designed and implemented as part of the work reported in this thesis.

A block diagram of this DR11-C interface is shown in fig. 7.3; the control signals are supplied from the OUTPUT buffer, the status bits are connected to the INPUT buffer. (the CSR is not used). The interface performs the following functions:

7.4.2.1 Driving the stepping motor

The stepping motor requires two signals to be supplied by the interface: a logical level for specifying the sense of

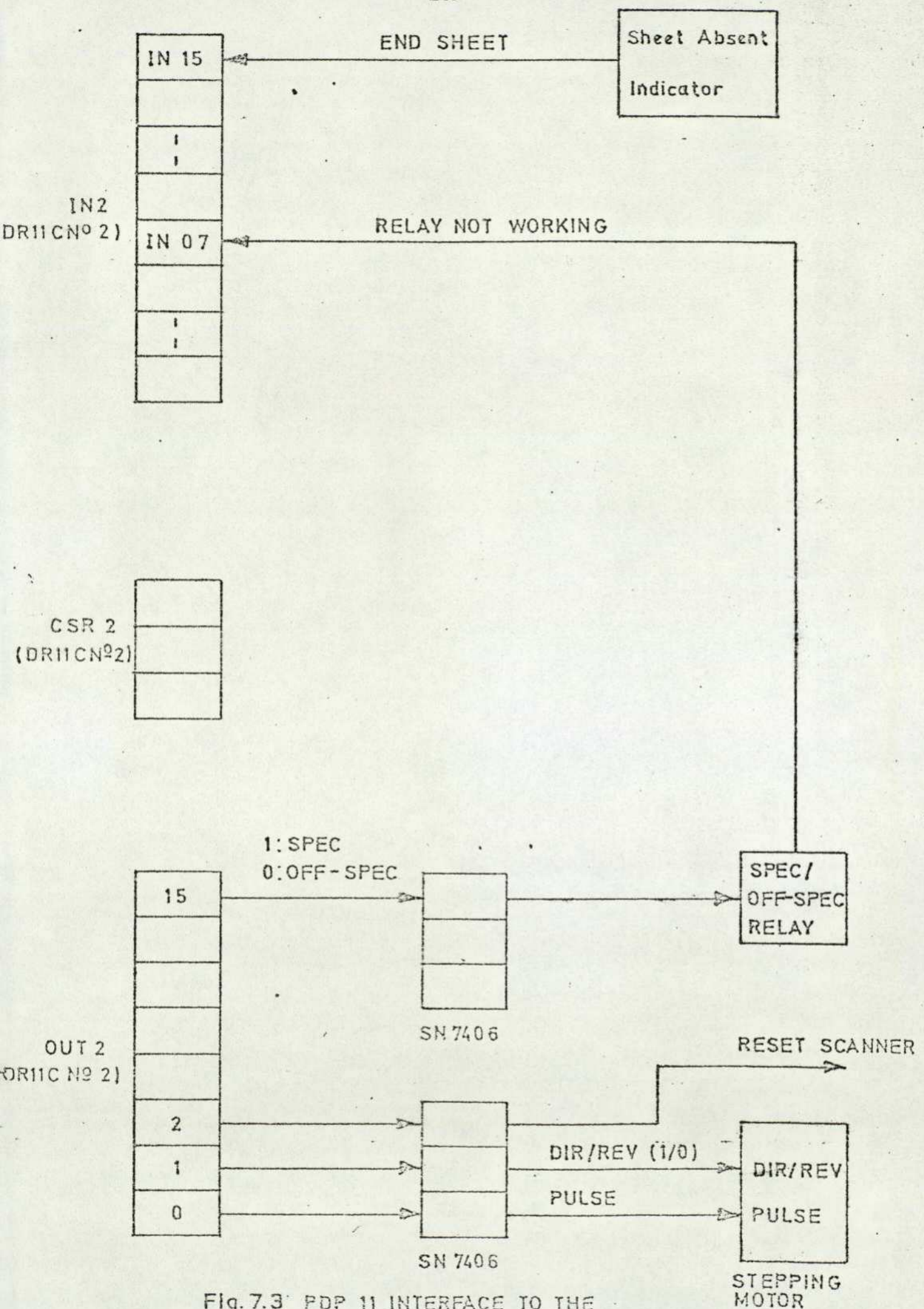


Fig. 7.3 PDP 11 INTERFACE TO THE
ELECTROMECHANICAL PARTS OF THE TEST RIG

rotation (direct/inverse) and a train of pulses, the number of which determines the angle of rotation. In order to keep a complete, flexible control over the rotation speed of the motor, the width (and repetition period) of the pulses are selected by programming means rather than by monostables.

The sense of rotation is controlled by bit 01 of the OUTPUT buffer, which is set to 1 for direct rotation and cleared to 0 for reverse direction. The pulses are generated by switching bit 00 of the OUTPUT buffer from 0 to 1 and 1 to 0; the state of this bit is maintained as long as desired by cycling an appropriate number of times through a group of instructions. For example, 183 cycles were required to give a pulse width of 1.5 msec and a repetition period of 3.0 msec, which ensured a reliable operation of the stepping motor (these values were derived from the execution times of the instructions belonging to the repeatedly executed group). The instructions providing one pulse and a period with a duty-factor of 0.5 were organized in the subroutine STEP which has only one parameter - the number of cycles required to give the selected pulse width (see [36]).

The number of pulses which determine the rotation angle is given by an outer loop, the counter of which is updated in the main program which calls repeatedly the subroutine STEP. In the present case, the length of the laser beam was 5mm; with the existing gear ratio, the linear increment of 5 mm required to cover completely the scanned sheet was obtained using a number of 185 cycles.

7.4.2.2 Driving the relay for selecting specular/off-specular scans

This relay is driven by a high-voltage open-collector TTL integrated circuit, connected to bit 15 of the OUTPUT buffer, which switches simultaneously 3 pairs of contacts:

- the first pair selects between specular/off-specular analog samples.
- the second pair selects between specular/off-specular trigger bits
- the last pair is used to sense the status of the relay, for detecting malfunctions and is connected to bit 07 of the INPUT buffer; the actual test instruction is executed after a waiting loop, which had to be introduced in the program to take into account the relatively long switching time of the relay as compared to the computer cycle.

7.4.2.3 Sensing the end of sheet

The electronics associated with the scanner supplies a logical level which is low while the laser beam is off the sheet and high while the laser beam is scanning the sheet. This signal is connected to bit 15 of the INPUT buffer and tested in the section of the main program which controls the table motion, as described in par 7.1.

7.4.2.4 Generating a reset signal for the DMA unit

During the transfer of data from the A/D converter, it may happen that the data transfer rate exceeds the speed of the computer memory. In this situation an error signal is generated by the DMA interface and the operation halts; to restart it, a RESET pulse must be supplied to the DMA interface. This function is performed automatically by clearing and setting bit 03 of the OUTPUT buffer.

7.5 Formatting of data

A major function of the automated data gathering system is to transfer the data acquired by the scanner to the computer memory, carry out the necessary formatting and conversion operations and write the resulting data on magnetic tape in a form suitable for direct processing by the pattern recognition software developed for the CDC 7600 computer. The main stages of the data transfer and processing operations, the equipment and PDP-11 software involved are represented in the block diagram in fig. 7.1. The information is transferred, processed and written on magnetic tape in blocks of constant length, each block corresponding to a complete scan.

The direct memory access (DMA) interface transfers the data produced by the scanner and associated electronics to the computer memory, where it is stored in the format represented in fig. 7.4. The data is structured in groups, each group comprising 16 analog samples (8 bit integers, unsigned) followed by a 16 bit word storing the corresponding trigger bits. The last group is followed by two words representing the leading margin width and the trailing margin width of the scan.

The first phase of data processing involves arranging the information provided by the scanner and the operator in an order suitable for the subsequent operations (fig. 7.5):

- a) the sheet identifier is always in the form SHEET ID₁ ID₂ ID₃ ID₄ where ID₁ - ID₄ are 4 characters specified by the operator.

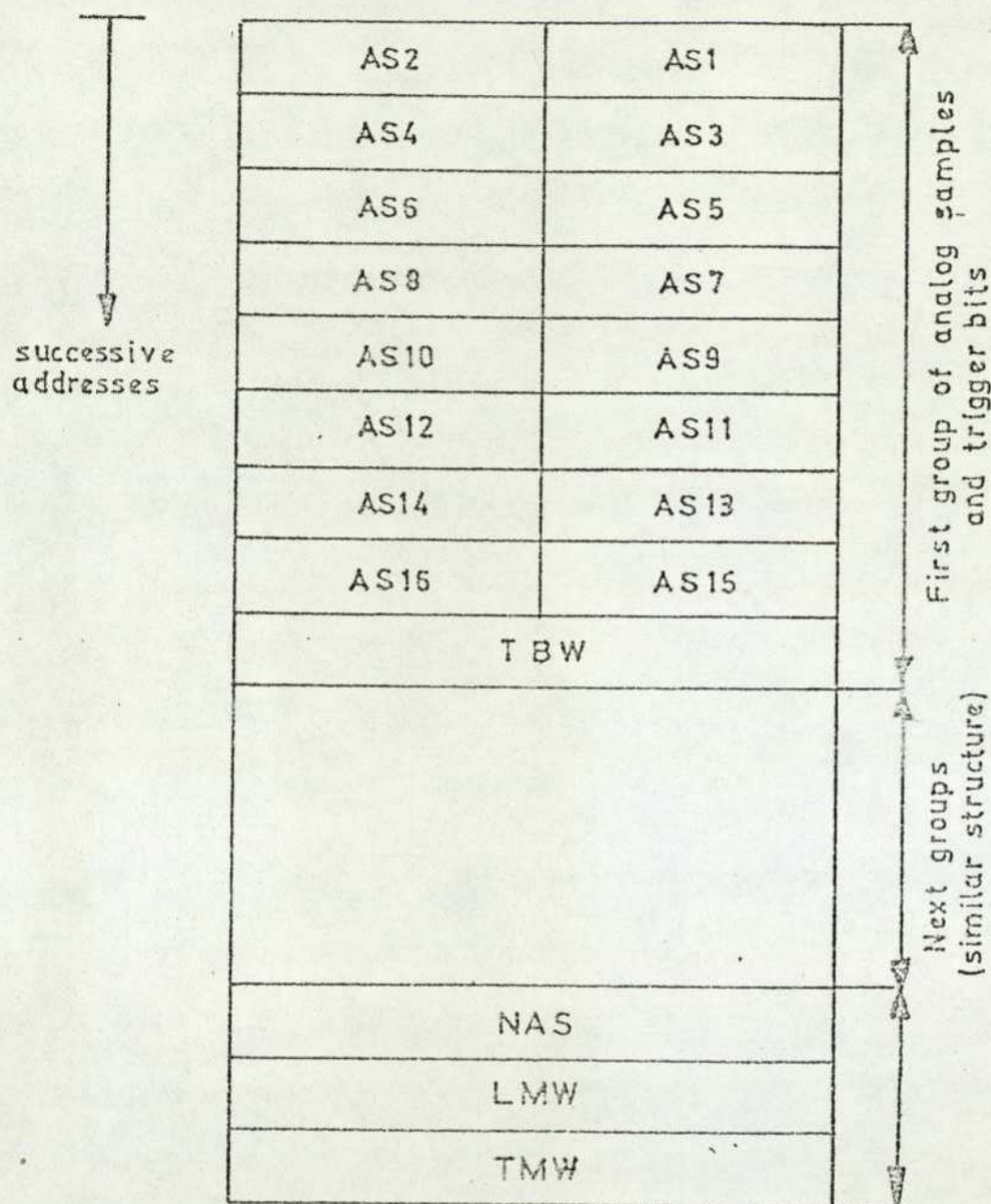
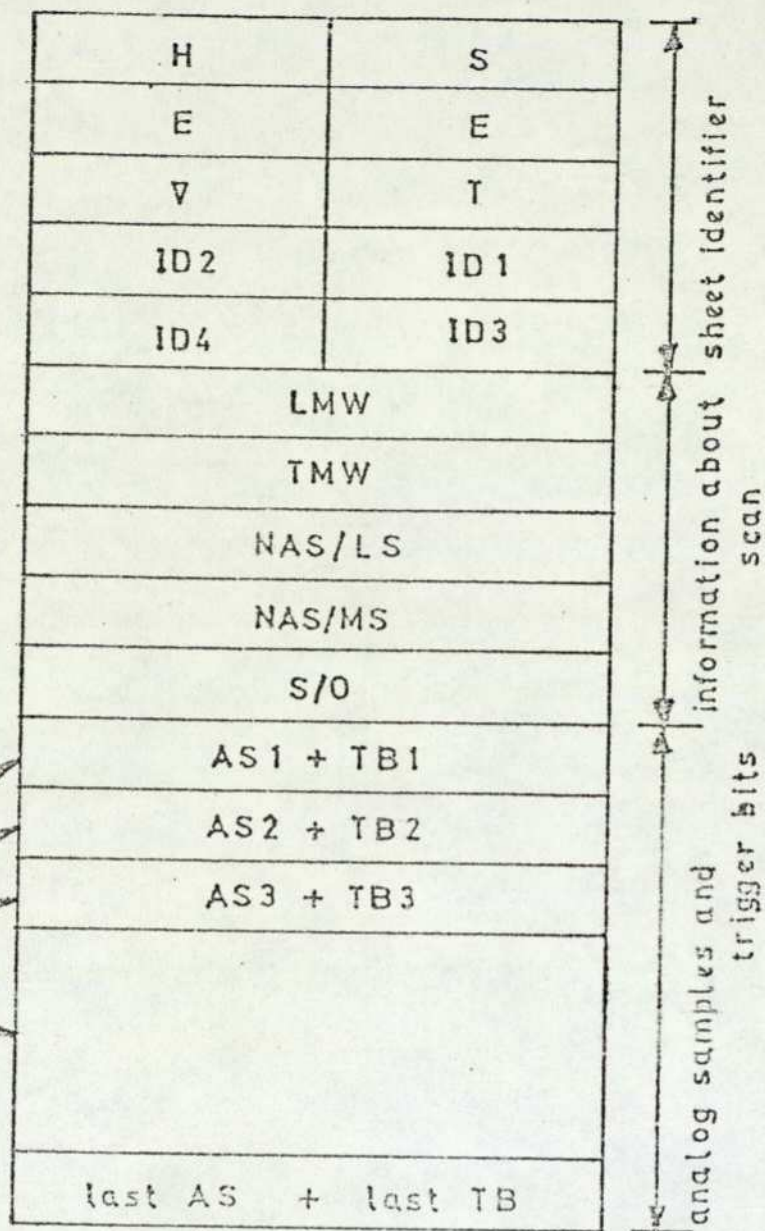
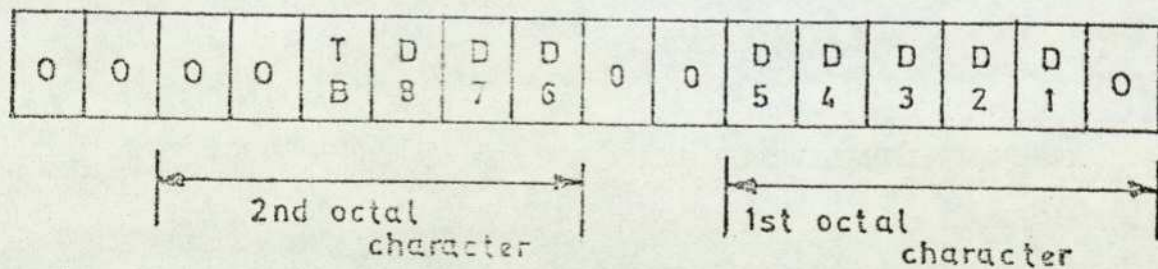


Fig. 7.4 Structure of data generated by scanner (as stored in BLOCK SCAN CORE MEMORY in the PDP-II)

- AS - analog sample
- TBW - trigger bits word
- LMW - leading margin width
- TMW - trailing margin width
- NAS - number of analog samples



Structure of a block of data after formatting in memory. (block TAPE)



Structure of a word in the block

D8, D7, D6, D5, D4, D3, D2, D1 - code of AS

TB - corresponding trigger bits

Fig.7.5 Data after formatting in PDP-II computer

b) the information about the scan comprises the following specifications: the leading margin width (LMW), the trailing margin width (TMW), the number of analog samples (NAS), and the scan type (S = specular/0 = off-specular).

c) successive analog scans together with the associated trigger bits, obtained by reformatting the words contained in the groups from fig. 7.4, as shown in fig. 7.5; this format was chosen in order to avoid writing null characters on magnetic tape, in even parity.

Before writing the data block on magnetic tape, a code conversion is necessary from the original code to the IBM external BCD code, which must appear on the magnetic tape in order to obtain the original code in the CDC 7600 core memory. The conversion is implemented simply using a table look-up procedure for each byte in the data block.

The blocks of data written on tape are of fixed length (3000 characters); if the number of bytes generated by the scanner is smaller, the remainder of the block is filled in with blanks.

The length of the block was chosen to be an integer multiple of 10, because 10 6-bit characters are packed in the 60 bit CDC 7600 word. Thus the first word in the CDC 7600 core memory will contain the sheet identifier, the second word the scan descriptor and the following words the analog samples and trigger bits (fig. 7.6).


 successive
addresses

S	H	E	E	T	V	ID1	ID2	ID3	ID4
LMW		TMW		NAS M/S		NAS L/S		S/O	
(AS+TB) ₁		(AS+TB) ₂		(AS+TB) ₃		(AS+TB) ₄		(AS+TB) ₅	

AS+TB		AS+TB		AS+TB		AS+TB		AS+TB	

Structure of data block in CDC 7600 core memory

$00b_7 \dots b_1 0$	$00b_7 \dots b_1 0$	$00b_7 \dots b_1 0$	$00b_7 \dots b_1 0$	$00b_7 \dots b_1 0$
---------------------	---------------------	---------------------	---------------------	---------------------

CDC 7600 data word (5×12 bit fields)

Fig.7.6 Data in CDC 7600 computer

8. COMPUTER SIMULATION OF THE SYNTACTIC PATTERN RECOGNITION SYSTEM

The object of this chapter is to present the results of the simulation experiments which have been performed to evaluate the performance of the syntactic pattern recognition system developed in the first part of this thesis, using the data base described in the previous chapter. The simulation program is based on the software packages written for the individual blocks of the syntactic pattern recognition system which have already been presented. These programs have been executed sequentially, in a batch processing mode, the designer having to examine the intermediate results and to provide the required information, so that the overall design process has an interactive character.

The areas in which the designer's intervention is necessary are the following:

- 1) specification of the class of each defect, after delineating and locating its position using the program CLUMPL; this information is introduced as data on cards;
- 2) evaluation of the separation capabilities of the selected pattern primitives and composition descriptors, by examining the results of the LENAL program; if the separation is not satisfactory, the syntactic elements are redefined and the design process has to be restarted;
- 3) specification of additional information required by the GRINF program, if the accuracy of the grammars inferred using only the training set is not satisfactory.

8.1 Syntactic representation of the defects

The procedure developed for extracting the syntactic representation of the defects from the information transmitted by the delineation system requires the following parameters to be specified:

- 1) the dimensions of the capture rectangle
- 2) the definitions of the pattern primitives and composition descriptors.

8.1.1 Selection of the capture rectangle dimensions

The dimensions of the capture rectangle have been decided by taking into account the following factors:

- 1) the scales in the two directions are different: K_x , the dimension in the transversal direction, is expressed in number of trigger samples and the unit corresponds to about 1 mm (1000 trigger samples for 1000 mm); K_y , the dimension in the longitudinal direction, is expressed in number of scans and the unit corresponds to about 5 mm (the beam length is 5 mm);

- 2) by selecting the capture rectangle dimensions too small, the clumping process becomes inefficient, because some sections belonging to a certain defect will be assigned incorrectly to a new defect;

- 3) by selecting the capture rectangle dimensions too large, the opposite situation may occur: defects which are physically distinct, sometimes even in different classes, could be combined and assigned the same number and class.

Different defect classes impose different requirements on the capture rectangle dimensions. Oil spots/grease marks, tin

stripping, dents and bruise marks are in general "compact" defects, with no gaps on the videoprint and would be delineated correctly by using a small capture rectangle. Scratches, laminations and five stand rings present relatively large gaps which can be bridged only by using a large capture rectangle. Since at the present stage a fixed capture rectangle technique is being employed, a compromise had to be found on an empirical basis; finally the solution $k_x = k_y = 15$ was adopted. The results are satisfactory, but certain incorrect delineations could not be avoided and refined clumping techniques are currently being investigated to overcome this problem.

8.1.2 Selection of the syntactic elements

The syntactic elements which have been introduced in ch. 4 were defined after a careful examination of the structure and geometrical properties of the defects, which revealed the existence of certain features that could be exploited to enhance the separation capabilities of the syntactic elements:

- 1) the length of the scan sections varies from class to class as follows: tin strippings and scratches consist of short scan sections (less than 6 trigger samples); the length of the scan sections is average for oil spots/grease marks (7-18 trigger samples); five stand rings and laminations consist in general of longer scan sections (more than 18 trigger samples);
- 2) most defects contain only one segment in a scan section, whereas the dirty surface defects consist of more than 3 segments in each scan section;
- 3) oil spots and tin strippings consist of scan sections which

are more closely connected (in successive scans) than in the defects belonging to the other classes;

4) scratches, laminations, five stand rings and dirty surface defects are in general much longer (extend over 15 successive scans or more) than the other defects.

An analysis of the structure and shape of laminations and five stand rings indicated that the information contained in the specular videoprint is not sufficient to discriminate between these two classes of defects which were therefore combined in a single class. Although many scratches are also inseparable from laminations or five stand rings, these two classes were not combined because they overlap only partially.

A complete list of the syntactic representations of the defects which have been used in the simulation experiment is shown in table 9.

A new class of defects (class 1), arbitrarily named "dots", has been introduced because they occur very frequently on the video-prints; most of these dots were initially not assigned to any of the other classes, but a few "dots" appear both as oil spots and as dents.

It is important to notice that the same string can be encountered many times, although the defects it describes are not necessarily identical so far as their geometrical properties are concerned. This situation is due to the fact that the syntactic elements characterise intervals in which certain geometrical features are included, rather than their absolute value. In general, this situation is

Table 9 Syntactic representation of the defects used in the simulation experiment.

CLASS NUMBER 1 - DOTS

1) LENGTH= 1 NR REP= 30 HALFA= 23
2
2

CLASS NUMBER 2 - OIL SPOTS

1) LENGTH= 2 NR REP= 1 HALFA= 58
32
25
2) LENGTH= 4 NR REP= 1 HALFA= 77
3112
2112
3) LENGTH= 2 NR REP= 6 HALFA= 89
44
26
4) LENGTH= 3 NR REP= 1 HALFA= 93
233
246
5) LENGTH= 4 NR REP= 1 HALFA= 101
2222
2666
6) LENGTH= 3 NR REP= 1 HALFA= 115
532
266
7) LENGTH= 4 NR REP= 1 HALFA= 130
3332
2665
8) LENGTH= 3 NR REP= 2 HALFA= 135
444
266
9) LENGTH= 5 NR REP= 1 HALFA= 157
31343
21265
10) LENGTH= 4 NR REP= 1 HALFA= 159
3443
2565
11) LENGTH= 4 NR REP= 1 HALFA= 166
3443
2466
12) LENGTH= 11 NR REP= 1 HALFA= 169
32111111112
26111111112

Table 9 continued.

CLASS NUMBER 3 - TIN STRIPP

1) LENGTH= 3 NR REP= 13 HALFA= 75
 222
 266
 2) LENGTH= 4 NR REP= 1 HALFA= 111
 3222
 2666

CLASS NUMBER 4 - DENTS

1) LENGTH= 1 NR REP= 4 HALFA= 33
 3
 2
 2) LENGTH= 2 NR REP= 1 HALFA= 56
 23
 23
 3) LENGTH= 2 NR REP= 1 HALFA= 59
 32
 26
 4) LENGTH= 2 NR REP= 1 HALFA= 78
 34
 25
 5) LENGTH= 4 NR REP= 1 HALFA= 102
 2133
 2126
 6) LENGTH= 4 NR REP= 1 HALFA= 121
 3413
 2512
 7) LENGTH= 5 NR REP= 1 HALFA= 164
 44412
 24412
 8) LENGTH= 4 NR REP= 1 HALFA= 180
 4444
 2566
 9) LENGTH= 13 NR REP= 1 HALFA= 239
 2111111113144
 2111111112123
 10) LENGTH= 10 NR REP= 1 HALFA= 255
 2111114444
 2111112456
 11) LENGTH= 14 NR REP= 1 HALFA= 699
 41466676444444
 21256555446544
 12) LENGTH= 34 NR REP= 1 HALFA= 773
 411111111144444464411111111112
 21111111112555446456111111111112

Table 9 continued.

13)	LENGTH= 4	NR REP= 1	HALFA= 170
	4443		
	2665		
14)	LENGTH= 9	NR REP= 1	HALFA= 176
	341111113		
	251111112		
15)	LENGTH= 8	NR REP= 1	HALFA= 177
	44111212		
	26111212		
16)	LENGTH= 5	NR REP= 1	HALFA= 179
	32533		
	24363		
17)	LENGTH= 5	NR REP= 1	HALFA= 185
	44332		
	26664		
18)	LENGTH= 5	NR REP= 1	HALFA= 195
	44432		
	26664		
19)	LENGTH= 5	NR REP= 1	HALFA= 203
	34434		
	25465		
20)	LENGTH= 14	NR REP= 1	HALFA= 221
	3211111111114		
	2511111111112		
21)	LENGTH= 15	NR REP= 1	HALFA= 223
	33111111111112		
	26111111111112		
22)	LENGTH= 11	NR REP= 1	HALFA= 245
	21111131443		
	21111121266		
23)	LENGTH= 15	NR REP= 1	HALFA= 288
	211111111111544		
	211111111111266		
24)	LENGTH= 15	NR REP= 1	HALFA= 324
	331111111131533		
	261111111121266		
25)	LENGTH= 14	NR REP= 1	HALFA= 357
	44313111111443		
	26612111111265		
26)	LENGTH= 21	NR REP= 1	HALFA= 376
	211131211111111444		
	2111212111111111266		
27)	LENGTH= 16	NR REP= 1	HALFA= 382
	344311111111443		
	256611111111265		
28)	LENGTH= 16	NR REP= 1	HALFA= 401
	443111111113463		
	266111111112456		
29)	LENGTH= 15	NR REP= 1	HALFA= 462
	221122315133556		
	221122212126656		
30)	LENGTH= 44	NR REP= 1	HALFA= 928
	2111112526511311113311111131131111134444		
	21111126544112111112311111121121111125666		

155

[illegible]

CLASS NUMBER 6 - SCRATCHES

```

1) LENGTH= 76      NR REP= 1      HALFA= 31
2111111121111111111132111111111121111111111212111112121111111111113111115222
21111111211111111111241111111111211111111112121111112121111111111112111112666
2) LENGTH= 12      NR REP= 1      HALFA= 196
221112111122
261112111126
3) LENGTH= 12      NR REP= 1      HALFA= 225
211122222112
211126665112
4) LENGTH= 14      NR REP= 1      HALFA= 231
21111111221222
21111111251265
5) LENGTH= 14      NR REP= 1      HALFA= 234
211111112211313
211111112611212
6) LENGTH= 15      NR REP= 1      HALFA= 242
211121231211112
211121221211112
7) LENGTH= 13      NR REP= 1      HALFA= 244
2121121222212

```


Table 9 continued

2121121266512		
8) LENGTH= 14	NR REP= 1	HALFA= 270
21122122122212		
21126126126512		
9) LENGTH= 18	NR REP= 1	HALFA= 292
212122111111121123		
212126111111121124		
10) LENGTH= 15	NR REP= 1	HALFA= 294
223111222211212		
254111266611212		
11) LENGTH= 72	NR REP= 1	HALFA= 330
32111111113111212251222142231222252522112111331112131111311112111112122		
23111111111211121222126512422122222423112111231112121111211112111112126		
12) LENGTH= 19	NR REP= 1	HALFA= 341
2121121212111222213		
2121121212111256612		
13) LENGTH= 23	NR REP= 1	HALFA= 352
21112121111211131112122		
21112121111211121112122		
14) LENGTH= 22	NR REP= 1	HALFA= 353
211111111112112123222		
211111111112112124565		
15) LENGTH= 23	NR REP= 1	HALFA= 379
2111111211211212122222		
2111111211211212126556		
16) LENGTH= 18	NR REP= 1	HALFA= 407
311111113431333312		
21111112551266612		
17) LENGTH= 29	NR REP= 1	HALFA= 408
2111111111121211121122112112		
2111111111121211121122112112		
18) LENGTH= 22	NR REP= 1	HALFA= 410
2112122111122222212122		
2112126111122666612124		
19) LENGTH= 29	NR REP= 1	HALFA= 415
211121211111111311111111222		
2111212111111111211111111266		
20) LENGTH= 28	NR REP= 1	HALFA= 422
211222211111311111121111212		
211262211111211111121111212		
21) LENGTH= 24	NR REP= 1	HALFA= 432
221211121112111222221222		
251211121112111256561265		
22) LENGTH= 30	NR REP= 1	HALFA= 472
31111311112221211111111212122		
21111211112651211111111212126		
23) LENGTH= 34	NR REP= 1	HALFA= 496
22112111113111111211311111311112		
25112111112111111211211111211112		
24) LENGTH= 34	NR REP= 1	HALFA= 512
31221112111111121111111211322112		
21221112111111121111111211256112		
25) LENGTH= 34	NR REP= 1	HALFA= 541
211111111111221111122111521112213		
211111111111261111126111261112612		
26) LENGTH= 35	NR REP= 1	HALFA= 568
211111122221121122211111111221122		
2111111255661121126611111111241126		

Table 9 continued

10) LENGTH= 19 NR REP= 1 HALFA= 289
 2111111132121121112
 2111111125121121112

11) LENGTH= 20 NR REP= 1 HALFA= 314
 2111111111121122222
 2111111111121126666

12) LENGTH= 17 NR REP= 1 HALFA= 315
 31112211211231132
 21112611211241125

13) LENGTH= 11 NR REP= 1 HALFA= 351
 22223333333
 26664666666

14) LENGTH=174 NR REP= 1 HALFA= 378
 3123441234323334443242443333223332433313433322343264233434333222222223222222
 322242223333233322223322223222322232222532232223332333444332233222222222
 22332321222222
 21245612455546656655444656666445466445661256665645554644656566656666664566666
 45664466465666546656666465465666456456454566456456645646654665665664656666666
 664654612656666

15) LENGTH= 76 NR REP= 1 HALFA= 383
 31331343323334333313332441311111413143313444334434344431333133434334432214
 21261256654665566612664461211111212126612566563655556651266126555656565612

16) LENGTH= 24 NR REP= 1 HALFA= 388
 211111111151111112222
 2111111111211111126666

17) LENGTH= 20 NR REP= 1 HALFA= 389
 21111133222221112122
 21111126566661112126

18) LENGTH= 13 NR REP= 1 HALFA= 426
 222256223332
 2546644664666

19) LENGTH= 24 NR REP= 1 HALFA= 451
 4252331111111111221112
 24454611111111111261112

20) LENGTH=175 NR REP= 1 HALFA= 464
 4333333233332323232223144322223232322243334433343232344324334433332223
 34333241433433334463434322232232221232223222442532223212212232322222232
 322362223222422
 25666664456644654546564566412655666645454564464566565665545456554565656666664
 6556644125555666666555555664564564566124566456446646454466645126126454566666645
 456446664566446

21) LENGTH=140 NR REP= 1 HALFA= 470
 52111121122212521112222212122111112222121121111122222111122211222221322111111
 1122123212223313111111111111211111111111122211122211222212
 2511112112661244111266661212611111266612112111112666561111256112666661256111111
 112612451266461211111111111121111111111111266111256611256512

22) LENGTH= 70 NR REP= 1 HALFA= 477
 333331121211111111111111111333331133332331133331111111111133233331113
 266661121211111111111111121266611266654611266661111111111126646661112

23) LENGTH=143 NR REP= 1 HALFA= 488
 311111111112123131233121113322132222332222123222222111111111111131311133333
 211323333333112131113333334444444322333323333311233222122
 21111111111212641212461211125561256666465666124666666511111111111212111266666
 611254665656611212111266665656666656466665466666112465666126

24) LENGTH= 89 NR REP= 1 HALFA= 497
 311333333343333333333333343333333311313333333333312312111111211111112321322
 334434433
 211256666656666666666656566666666112126666666666612412111111211111112451256
 465655566

[illegible]

CLASS NUMBER 8 - DIRTY SURF

[illegible]

CLASS NUMBER 9 - AMBIG STR

[illegible]

160

advantageous, because it leads to an information reduction which is reflected in a simplification of the structure of the recognizers. However, precautions must be taken in the special case when two or more defects have the same string representation, but belong to different classes. This situation is detected automatically by the program LENAL and the corresponding strings are assigned to a special class (class 9), also listed in table 9.

Finally, table 10 presents the results of the analysis of the separation capabilities of the syntactic elements defined in ch. 4 and the length distribution of the strings over the entire range (from 0 - 200 scans), separately for each defect class.

8.2 Synthesis of the syntactic pattern classifier

The syntactic description of the defects as strings of pattern primitives and composition descriptors, shown in table 9, reveals a complicated structure of the pattern languages and a high degree of structural variability within certain defect classes, such as oil spots/grease marks, dents and bruises. These properties indicate that an automatic method for grammar inference is imperatively required and therefore justify the effort dedicated to the development of the algorithm and program which solve this problem. Certain classes of defects, such as scratches, laminations/five stand rings and dirty surface, are described by strings of greater length, most of them having a relatively simple structure. Regular grammars which generate languages containing long strings are not necessarily complicated, because recursive productions provide a concise formalism for describing situations of this type.

CLASS= 1	NR OF DEFECTS= 30
CLASS= 2	NR OF DEFECTS= 36
CLASS= 3	NR OF DEFECTS= 14
CLASS= 4	NR OF DEFECTS= 15
CLASS= 5	NR OF DEFECTS= 8
CLASS= 6	NR OF DEFECTS= 33
CLASS= 7	NR OF DEFECTS= 38
CLASS= 8	NR OF DEFECTS= 6
CLASS= 9	NR OF DEFECTS= 17

NUMBERS OF OCCURENCES

CLASS:	1	2	3	4	5	6	7	8	9
SYMBOLS									
SYMBOL= 1	0	47.49	0	40.82	58.94	63.54	31.67	0	0
SYMBOL= 2	100.00	9.72	97.67	7.14	7.95	31.27	33.49	.13	20.00
SYMBOL= 3	0	18.73	2.33	12.24	17.22	3.92	22.36	.13	73.33
SYMBOL= 4	0	20.07	0	33.67	9.93	.27	7.59	.25	6.67
SYMBOL= 5	0	3.31	0	0	1.32	.82	3.68	.51	0
SYMBOL= 6	0	1.00	0	5.10	4.64	.18	.95	2.16	0
SYMBOL= 7	0	0	0	1.22	0	0	.26	96.82	0

PREDICATES

PREDICAT= 1	0	47.49	0	40.82	58.94	63.54	31.67	0	0
PREDICAT= 2	100.00	22.74	32.56	24.49	20.53	23.61	12.00	1.14	53.33
PREDICAT= 3	0	1.20	0	2.04	.66	.36	.40	.25	0
PREDICAT= 4	0	3.01	0	10.20	3.97	1.54	11.27	5.84	0
PREDICAT= 5	0	5.69	0	13.27	3.31	2.55	10.84	40.66	0
PREDICAT= 6	0	20.27	67.44	9.18	12.58	8.30	33.82	52.10	46.6

SYN-PRED PAIRS

SYM-PRED= 0- 0	0	0	0	0	0	0	0	0	0
SYM-PRED= 1- 1	0	47.49	0	40.82	58.94	63.54	31.67	0	0
SYM-PRED= 2- 2	100.00	5.69	30.23	6.12	6.62	10.96	7.77	.13	10.00
SYM-PRED= 2- 3	0	0	0	0	0	.27	.33	0	0
SYM-PRED= 2- 4	0	1.02	0	0	.66	.82	2.26	0	0
SYM-PRED= 2- 5	0	1.34	0	0	0	2.28	5.69	0	0
SYM-PRED= 2- 6	0	1.67	67.44	1.02	.66	7.93	17.44	0	10.00
SYM-PRED= 3- 2	0	9.36	2.33	10.20	11.28	2.83	3.06	.13	36.67
SYM-PRED= 3- 3	0	.67	0	1.02	.66	.09	.04	0	0
SYM-PRED= 3- 4	0	.33	0	0	.66	.55	6.20	0	0
SYM-PRED= 3- 5	0	1.67	0	0	1.32	.09	2.23	0	0
SYM-PRED= 3- 6	0	6.69	0	1.02	3.31	.36	10.84	0	36.67
SYM-PRED= 4- 2	0	6.35	0	8.16	.66	.18	.62	0	6.67
SYM-PRED= 4- 3	0	0	0	1.02	0	0	0	0	0
SYM-PRED= 4- 4	0	1.02	0	9.18	.66	0	1.57	0	0
SYM-PRED= 4- 5	0	2.01	0	9.18	1.32	.09	1.90	0	0
SYM-PRED= 4- 6	0	10.70	0	6.12	7.28	0	3.50	.25	0
SYM-PRED= 5- 2	0	1.34	0	0	1.32	.55	.44	.13	0
SYM-PRED= 5- 3	0	.33	0	0	0	0	.04	.13	0
SYM-PRED= 5- 4	0	.33	0	0	0	.18	.73	.13	0
SYM-PRED= 5- 5	0	.33	0	0	0	.09	.73	.13	0
SYM-PRED= 5- 6	0	.67	0	0	0	0	1.75	0	0
SYM-PRED= 6- 2	0	0	0	0	.66	.09	.07	0	0
SYM-PRED= 6- 4	0	.33	0	1.02	1.99	.09	.40	.13	0
SYM-PRED= 6- 5	0	.33	0	3.06	.66	0	.26	.64	0
SYM-PRED= 6- 6	0	.33	0	1.02	1.32	0	.22	1.40	0
SYM-PRED= 7- 2	0	0	0	0	0	0	.04	.76	0
SYM-PRED= 7- 3	0	0	0	0	0	0	0	.13	0
SYM-PRED= 7- 4	0	0	0	0	0	0	.11	5.59	0
SYM-PRED= 7- 5	0	0	0	1.02	0	0	.04	39.90	0
SYM-PRED= 7- 6	0	0	0	0	0	0	.07	50.44	0

LENGTH DISTRIBUTION WITHIN CLASSES

[illegible]

However, precautions must be taken when, as in the present case, the language contains only strings longer than ℓ ($\ell \gg 1$) and their prefixes which are shorter than ℓ belong to other languages. To prevent the grammar from assigning these prefixes to the language, additional productions have to be introduced, but this complicates the structure of the recognizer, especially when $\ell \gg 1$.

Since the classification of the strings is performed on the basis of their MNEO distances, this technique was extended to simplify the recognition of the defects described by long strings with a simple structure. The MNEO distances associated with these classes of defects are computed simply by counting the number of syntactic elements which are not of the corresponding type and finally by adding a penalty when the length of the string is not in the characteristic interval.

The simplest example of this type is represented by the dirty surface defects, in which the great majority of pattern primitives is of type 7 and their length extends between 120 and 141 scans. The MNEO distance from a string of length ℓ to the language associated to this class can be computed simply by counting the number of pattern primitives which are not of type 7 and finally adding $(120 - \ell)$ if $\ell < 120$.

Consequently, the synthesis of the syntactic pattern classifier has been accomplished in three phases:

- 1) inference of the grammars associated with the classes of defects described by short strings with a variable structure;

- 2) synthesis of the MNEO distance processors based on the inferred grammars;
- 3) direct synthesis of the MNEO distance processors for the classes of defects described by long strings with a simple structure.

8.2.1 Grammar inference for the classes of short defects with a variable structure

The classes of defects for which the grammar inference procedure has been applied are the following: 1) dots; 2) oil spots/grease marks; 3) tin stripping; 4) dents; 5) bruise marks. To evaluate the extrapolation capabilities of the grammars and the corresponding MNEO classifier, the set of defects belonging to the data base was divided in two disjoint subsets: the first was used for training in the grammar inference process and the second for testing the performance of the classifier.

The interactive facilities of the grammar inference algorithm have not been exploited in this experiment because the additional information required could be provided reliably only in co-operation with an inspector specialized in tinplate defect recognition and this was impractical in the present batch processing mode. Instead, all strings which would have been specified by the designer in an interactive manner had been automatically considered "not defined", but this solution represents a basic limitation of the maximum accuracy which can be achieved.

To reduce the complexity of the grammars, only the most representative syntactic elements have been selected for each grammar, by inspecting the frequency with which they occur in the corresponding

class (see table 10). The grammar inference process was carried out for the following values of the parameter k : $k = 1, 2, 3, 4$. The transition matrices associated with the nondeterministic recognizers constructed by the GRINF program are shown in table 11, in which the nature of each state is specified according to the following convention:

CLASS 1 - final state (string accepted)

CLASS 2 - reject state (string not accepted)

CLASS 0 - undefined state (string not classified).

The precision of the inferred grammars improves slightly when the length of the k -tails is increased from 1 to 3, but further increases do not lead to better language approximations. This saturation effect is due to two factors:

1) the number of terminal syntactic elements included in the definition of the grammars being only a subset of the syntactic elements which actually occur in the strings belonging to the corresponding class, all the strings which contain syntactic elements not specified in the definition of the grammar are "imperfect" strings and are automatically rejected;

2) certain strings which are not "imperfect" are misclassified because some of their prefixes are not defined in the training set; these strings are listed at the beginning of the inference process (see table 11) and automatically interpreted as "don't care" because there is no interaction with the designer.

None of these difficulties is an inherent limitation of the grammar inference algorithm and could be overcome by augmenting the

Table 11. Transition matrices of the recognizers.

MAIN PROBLEM PARAMETERS

UNDEFINED STRING
 23
 24
 UNDEFINED STRING
 5
 2
 UNDEFINED STRING
 53
 26
 UNDEFINED STRING
 213
 212
 UNDEFINED STRING
 322
 266
 UNDEFINED STRING
 344
 256
 UNDEFINED STRING
 34
 24
 UNDEFINED STRING
 344
 246
 UNDEFINED STRING
 44
 25
 UNDEFINED STRING
 444
 256
 UNDEFINED STRING
 313
 212
 UNDEFINED STRING
 3134
 2126
 UNDEFINED STRING
 44
 24
 UNDEFINED STRING
 444
 244
 UNDEFINED STRING
 325
 243
 UNDEFINED STRING
 3253
 2436
 UNDEFINED STRING
 443
 266
 UNDEFINED STRING
 4433
 2666
 UNDEFINED STRING
 4443
 2666
 UNDEFINED STRING
 344
 254
 UNDEFINED STRING
 3443
 2546
 UNDEFINED STRING
 4444
 2666
 UNDEFINED STRING
 44444
 26666
 UNDEFINED STRING
 3444
 2566
 UNDEFINED STRING
 34446
 25666
 UNDEFINED STRING
 344464
 256666
 UNDEFINED STRING
 441112
 261112

Table 11 continued

3-TAILS OF SYM-PRED PAIRS

INFERENCE PROCESS FOR CLASS= 1

TRANSITION MATRIX

2
2

I= 1 CLASS= 0 2 0
I= 2 CLASS= 1 1 0
I= 3 CLASS= 2 1 0
I= 4 CLASS= 2 4 0

PERF=100.00%

CONFUSION MATRIX

1 0
0 37

INFERENCE PROCESS FOR CLASS= 2

TRANSITION MATRIX

2 3 4 5 2 2 3 3 4 4 4 2
2 2 2 2 4 5 5 6 4 5 6 6

I= 1 CLASS= 0 2 0 3 0 2 3 1 6 4 0 1 6 4 0 4 0 1 6 4 0 1 6 4 0 1 6 4 0
I= 2 CLASS= 2 1 6 1 6 1 6 1 6 1 6 4 0 1 6 1 6 1 6 1 6 4 0 3 0
I= 3 CLASS= 2 1 6 1 6 1 6 1 6 2 3 4 0 1 6 2 3 1 6 2 3 1 6 2 5
I= 4 CLASS= 1 1 6 1 6 1 6 1 6 1 6 1 6 4 0 1 6 1 6 1 6 4 0 1 6
I= 5 CLASS= 2 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6 4 0
I= 6 CLASS= 0 1 6 1 6 1 6 1 6 1 6 1 6 2 3 1 6 1 6 1 6 1 6
I= 7 CLASS= 2 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0

PERF= 87.14%

CONFUSION MATRIX

INFERENCE PROCESS FOR CLASS= 3

TRANSITION MATRIX

2	3	2
2	2	6

I= 1	CLASS= 0	2	0	3	0	2	3
I= 2	CLASS= 2	1	6	1	6	4	0
I= 3	CLASS= 2	1	6	1	6	2	3
I= 4	CLASS= 2	1	6	1	6	5	0
I= 5	CLASS= 1	1	6	1	6	2	3
I= 6	CLASS= 0	1	6	1	6	5	0
I= 7	CLASS= 2	7	0	7	0	7	0

PERF=100.00%

CONFUSION MATRIX

2	0
---	---

2	36
---	----

INFERENCE PROCESS FOR CLASS= 4

TRANSITION MATRIX

2	3	4	2	3	3	4	4	4	4
2	2	2	6	3	6	3	4	5	6

I= 1	CLASS= 0	2	0	3	0	2	0	2	0	2	0	2	0	1	4	1	4	1	4	1	4
I= 2	CLASS= 2	1	4	1	4	1	4	2	0	3	0	1	4	1	4	1	4	1	4	2	0
I= 3	CLASS= 1	1	4	1	4	1	4	3	0	1	4	2	0	1	4	1	4	3	0	1	4
I= 4	CLASS= 0	1	4	1	4	1	4	1	4	1	4	3	0	1	4	1	4	1	4	1	4
I= 5	CLASS= 0	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	5	0
I= 6	CLASS= 2	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

PERF= 88.57%

CONFUSION MATRIX

4	5
---	---

Table 11 continued.

INFERENCE PROCESS FOR CLASS= 5

TRANSITION MATRIX

2	3	4	2	3	3	4	4
2	2	2	4	3	6	5	6

I= 1	CLASS# 0	2	4	2	0	2	4	1	5	1	5	2	4	1	5	1	5
I= 2	CLASS# 2	1	5	1	5	1	5	3	0	2	4	4	0	2	4	2	4
I= 3	CLASS# 1	1	5	1	5	1	5	1	5	1	5	1	5	1	5	1	5
I= 4	CLASS# 2	1	5	1	5	1	5	1	5	1	5	3	0	1	5	1	5
I= 5	CLASS# 0	1	5	1	5	1	5	1	5	1	5	1	5	1	5	5	0
I= 6	CLASS# 2	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

PERF= 94.29%

CONFUSION MATRIX

2	2
2	32

set of terminal syntactic elements (thus increasing the complexity of the grammars) and by running the GRINF program in an interactive manner, respectively.

The confusion matrices and the recognition performance specified separately for each recognizer have been evaluated on the training set, by attempting to separate the strings in the corresponding class from all the other strings. This experiment is useful as an indication about the precision of the inferred grammar and the nature of the errors, but has no direct relevance to the global classification of the defects which will be discussed in par. 8.3.

8.2.2 Synthesis of the MNEO distance processors for the classes of defects described by short strings with a variable structure

The transition matrices which describe the acceptors constructed by the GRINF program represent the input data for the program SYNREC which computes for each defect class the associated correction table $V(T, S, c)$ used to calculate the MNEO distance from a string to the corresponding language.

Since the grammar inference procedure has been carried out for 4 values of the parameter k ($k = 1, 2, 3, 4$), the program SYNREC will produce 4 sets of correction tables. Table 12 shows only the correction tables obtained for the value $k = 3$, which provides an optimum performance - complexity trade-off of the syntactic pattern classifier. Although the correction tables are 3-dimensional arrays, the computer print-out is organized to produce 2-dimensional tables, to permit a simpler interpretation: for each defect class, the associated correction table is printed separately to indicate the MNEO

Table 12. Correction tables

DESCRIPTION OF CLASS= 1

NS= 4 MS= 3 NFST= 1

FINAL STATES:

2

NR OF SYM-PRED PAIRS= 2

2 1
2 1

MATRIX VTSC

I= 1

1	0	1
0	1	1
9999	9999	9999
1	1	0

I= 2

0	0	1
1	0	1
9999	9999	9999
1	1	0

I= 3

0	0	1
1	1	2
1	1	1
1	1	0

I= 4

9999	9999	9999
9999	9999	9999
9999	9999	9999
0	0	0

[illegible]

I= 6

Table 12 continued.

173

0	0	0	0	0	0	0	1	0	0	0	0	0	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0

I= 7

```

9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

DESCRIPTION OF CLASS= 3

NS= 7 MS= 5 NFST= 1

FINAL STATES:

5

NR OF SYM-PRED PAIRS= 4

```

2 3 2 1
2 2 6 1

```

MATRIX VTSC

I= 1

1	1	1	0	1
0	1	0	1	1
1	0	0	1	1
1	1	0	1	1
2	2	1	2	2
1	1	1	2	2
1	1	1	1	0

I= 2

Table 12 continued.

174

0	0	1	0	1
1	1	1	1	1
1	1	1	1	2
1	1	0	1	1
1	1	1	2	2
0	0	1	1	1
1	1	1	1	0

I= 3

0	0	1	0	1
1	1	0	1	1
1	1	0	1	1
1	1	1	1	2
1	1	1	2	2
0	0	1	1	1
1	1	1	1	0

I= 4

0	0	1	0	1
1	1	1	1	2
1	1	1	1	2
1	1	1	1	1
1	1	0	1	1
0	0	1	1	1
1	1	1	1	0

I= 5

0	0	1	0	1
1	1	0	1	1
1	1	0	1	1
1	1	0	1	1
1	1	1	0	1
0	0	1	1	1
1	1	1	1	0

I= 6

0	0	1	0	1
1	1	1	1	2
1	1	1	1	2
1	1	1	1	2
1	1	0	1	1
0	0	1	1	1
1	1	1	1	0

I= 7

9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
0	0	0	0	0

[illegible]

3

9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
1	2	3	4	5	6	7	8	9	0	1	2	3

distances from a given state i to all states of the acceptor, for all terminal syntactic elements specified in the definition of the language.

8.2.3 Direct synthesis of the MNEO distance processors for the classes of defects described by long strings with a simple structure

The classes of defects included in this group are the following: 6) scratches; 7) laminations and five stand rings; 8) dirty surface. The basis for the design of the MNEO distance processors associated with these classes is represented by the analysis of the strings shown in table 10, which reveals the following properties which can be used to simplify the recognition of the defects:

1) scratches: 72% of the defects are shorter than 40 scans, but longer than 12 scans; 73.5% of the syntactic elements are of type 1-1 and 2-2;

2) laminations and five stand rings: their length covers the entire range from 0 - 180 scans, but 75% of the string lengths are longer than 25 scans; the most frequent syntactic elements are of type 1-1, 2-6, 3-6, 4-6 and 5-6;

3) dirty surface: all defects are longer than 120 scans and 96.03% of the syntactic elements contain the pattern primitive of type 7 (3 or more scan sections), which does appear only accidentally in other classes of defects.

One of the difficulties expected to be encountered in separating scratches from laminations is due to the fact that some defects in these classes are described by similar strings. An analysis of

the corresponding specular video-prints indicates that some of these defects have indeed a similar pattern and therefore a higher degree of misclassifications between these classes will have to be accepted. Additional information, such as off-specular videoprints or analog samples, is necessary to obtain a better separation between these classes. The frequencies with which common syntactic elements (type 1-1 and 2-2) occur in these two classes of defects are different and this property could be exploited to enhance the separation capabilities of the classifier, but it has been considered that at the present stage the complications introduced by assigning non-integer weights is not justified.

The formulae finally adopted for computing the MNEO distances associated with the classes of defects, 6, 7 and 8 are the following (ℓ denotes the length of the strings, P_j denotes the primitive j and $\bar{P}_j$ all the other primitives):

- 1) scratches:

$$\text{MNEO (6)} = \sum_{i=1}^{\ell} \frac{1}{(P_1 + P_2)^i} \quad (8.1)$$

$$\text{if } \ell > 40, \text{ MNEO (6)} = \text{MNEO (6)} + (\ell - 40) \quad (8.2)$$

- 2) laminations and five stand rings:

$$\text{MNEO (7)} = \sum_{i=1}^{\ell} \frac{1}{(P_1 + P_2 + P_3 + P_4 + P_5)^i} \quad (8.3)$$

$$\text{if } \ell < 30, \text{ MNEO (7)} = \text{MNEO (7)} + (30 - \ell) \quad (8.4)$$

3) dirty surface:

$$\text{MNEO (8)} = \sum_{i=1}^{\ell} \bar{P}_7^i \quad (8.5)$$

$$\text{if } \ell < 120, \text{ MNEO (8)} = \text{MNEO (8)} + (120 - \ell) \quad (8.6)$$

The classification performance of the syntactic pattern recognition system was also evaluated for a second situation, in which no attempt has been made to separate scratches from laminations and five stand rings. In this case, the combined class, denoted by 6, and the dirty surface, denoted by 7, are classified using the following formulae for the MNEO distances:

$$1) \quad \text{MNEO (6)} = \sum_{i=1}^{\ell} \overline{(P_1 + P_2 + P_3 + P_4 + P_5)}^i \quad (8.7)$$

$$\text{if } \ell < 16, \text{ MNEO (6)} = \text{MNEO (6)} + (16 - \ell) \quad (8.8)$$

$$2) \quad \text{MNEO (7)} = \sum_{i=1}^{\ell} \bar{P}_7^i \quad (8.9)$$

$$\text{if } \ell < 120, \text{ MNEO (7)} = \text{MNEO (7)} + (120 - \ell) \quad (8.10)$$

Although this technique is very simple and does not take into account the statistical properties of the defects, it is well adapted to the general concept of edit distances and leads to a reliable recognition of these classes of defects.

8.3 Recognition performance of the syntactic defect classification system

The performance of the syntactic classifier has been evaluated on a testing set of defects, obtained from the complete data base after eliminating the defects used in the grammar inference process. Since a

number of different defects may have the same syntactic representation, the sets of strings used in designing and testing the classifier are not necessarily disjoint. This situation is likely to bias slightly favourably the performance of the classifier, but since it reflects a specific feature of the syntactic method, precautions to eliminate this effect by artificial corrections are not justified.

The experiment has been performed for the values of $k = 1, 2, 3, 4$, in two situations: 1) when scratches are considered a separate class; 2) when scratches are combined with laminations and five stand rings in a common class.

The final results are summarized in table 13. The computer print-out shown in table 14 describes in detail the operation of the syntactic MNEO classifier as a support for the interpretation of the results. The results refer to the classifier designed with $k = 3$, tested in the case in which scratches are treated as a separate class; only strings which are not classified correctly are included in the table.

Considering the high variability of the defect properties within a class, the final results may be interpreted as a proof of the validity of the theoretical methods and the feasibility of the proposed syntactic pattern recognition system. The general framework offers a sound basis on which improvements can be incorporated to ensure a better performance. The shortcomings of the present system are revealed by the nature of the classification errors, which is discussed in the following section, with reference to case 1, when scratches are treated as a separate class.

Table 13 Simulation results

Recognition performance

case \ k	1	2	3	4
1. Scratches treated as a separate class	66.54	68.88	70.92	70.92
2. Scratches combined with laminations and VSR	78.97	79.24	80.10	80.10

Confusion matrix (Case 1, k = 3)

	1	2	3	4	5	6	7	8	9
1	30	0	0	0	0	0	0	0	0
2	0	18	0	2	0	3	1	0	12
3	0	0	13	0	0	0	0	0	1
4	0	2	0	8	0	0	1	0	4
5	0	1	0	1	2	1	1	0	2
6	0	0	0	0	0	20	10	0	3
7	0	2	0	0	0	8	26	0	2
8	0	0	0	0	0	0	0	6	0

Case 1

1 - dots

2 - oil spots/grease marks

3 - tin stripping

4 - dents

5 - bruises

6 - scratches

7 - laminations and five stand rings

8 - dirty surface

9 - reject class

Confusion matrix (Case 2, $k = 3$)

	1	2	3	4	5	6	7	8
1	30	0	0	0	0	0	0	0
2	0	15	0	2	0	9	0	10
3	0	0	13	0	0	0	0	1
4	0	2	0	8	0	3	0	2
5	0	1	0	1	2	3	0	1
6	0	1	0	0	0	67	0	3
7	0	0	0	0	0	0	6	0

Case 2

1 - 5 - as in case 1

6 - scratches, laminations and five stand rings

7 - dirty surface

8 - reject class

[illegible]

Table 14. Analysis of the classification errors.

2256
25654

107) CLASS= 2 CLASS ASS= 4 NOTDEF=0
DISTANCES= 5 1 4 0 2 207 25 120
34434
25465

108) CLASS= 5 CLASS ASS= 6 NOTDEF=0
DISTANCES= 1 1 2 1 1 0 19 120
2111111111126
2111111111124

109) CLASS= 2 CLASS ASS= 6 NOTDEF=0
DISTANCES= 3 1 3 2 2 0 17 120
3211111111114
2511111111112

110) CLASS= 2 CLASS ASS= 6 NOTDEF=0
DISTANCES= 2 1 3 1 1 0 16 120
3311111111112
2511111111112

113) CLASS= 6 CLASS ASS= 9 NOTDEF=1
DISTANCES= 3 1 2 0 2 0 17 120
21111112211313
21111112611212

114) CLASS= 4 CLASS ASS= 9 NOTDEF=1
DISTANCES= 3 1 2 1 1 200 17 120
2111111113144
2111111112123

118) CLASS= 2 CLASS ASS= 9 NOTDEF=1
DISTANCES= 4 0 3 0 0 201 19 120
21111131443
21111121266

119) CLASS= 7 CLASS ASS= 6 NOTDEF=0
DISTANCES= 200 200 200 200 200 0 15 120
321111112211112
2511111112611112

120) CLASS= 4 CLASS ASS= 2 NOTDEF=0
DISTANCES= 4 2 4 1 2 202 20 120
2111114444
2111112456

122) CLASS= 5 CLASS ASS= 2 NOTDEF=0
DISTANCES= 6 0 6 1 1 206 24 120
444444
256666

126) CLASS= 7 CLASS ASS= 6 NOTDEF=0
DISTANCES= 200 200 200 200 200 0 12 120
211111113212112112
211111112512112112

130) CLASS= 7 CLASS ASS= 6 NOTDEF=0
DISTANCES= 200 200 200 200 200 0 11 120
2111111111121122222
2111111111121126666

131) CLASS= 7 CLASS ASS= 9 NOTDEF=0
DISTANCES= 200 200 200 200 200 200 13 120
31112211211231132
21112611211241125

132) CLASS= 5 CLASS ASS= 4 NOTDEF=0
DISTANCES= 7 2 6 1 2 205 24 120
3444643
2566663

133) CLASS= 5 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 236 2 120
2111111113111311515366411231311111111113111311113111121331111111113112
21111111112111211212646611221211111111112111121111212411111111112112

134) CLASS= 2 CLASS ASS= 9 NOTDEF=1
DISTANCES= 6 1 4 1 1 200 15 120
331111111131533
251111111121266

Table 14 continued.

137) CLASS= 7 CLASS ASS= 9 NOTDEF=1
 DISTANCES= 10 1 7 1 4 201 19 120
 22223333333
 25664666666
 141) CLASS= 2 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 200 9 120
 2111312111111111444
 21112121111111111266
 145) CLASS= 2 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 200 14 120
 3443111111111443
 2566111111111265
 147) CLASS= 7 CLASS ASS= 6 NOTDEF=0
 DISTANCES= 200 200 200 200 0 7 120
 21111111111511111122222
 21111111112111111126666
 148) CLASS= 7 CLASS ASS= 6 NOTDEF=0
 DISTANCES= 200 200 200 200 0 11 120
 2111113322221112122
 2111112666661112126
 149) CLASS= 2 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 15 120
 443111111113463
 2661111111112456
 150) CLASS= 6 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 12 120
 31111113431333312
 211111112561266612
 155) CLASS= 7 CLASS ASS= 2 NOTDEF=0
 DISTANCES= 12 3 7 5 9 200 18 120
 222256223332
 2546644664666
 164) CLASS= 5 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 200 9 120
 211123116334431111112
 2111221126646511111112
 165) CLASS= 6 CLASS ASS= 7 NOTDEF=0
 DISTANCES= 200 200 200 200 200 0 120
 221121111131111112113111113111112
 251121111121111112112111112111112
 169) CLASS= 7 CLASS ASS= 6 NOTDEF=0
 DISTANCES= 200 200 200 200 0 1 120
 4111111112211111211112232111111122
 2111111112611111211112646111111126
 171) CLASS= 7 CLASS ASS= 6 NOTDEF=0
 DISTANCES= 200 200 200 200 0 1 120
 211111111111222211121232222122
 211111111111266611121246666126
 172) CLASS= 5 CLASS ASS= 9 NOTDEF=0
 DISTANCES= 200 200 200 200 200 10 120
 2211344636111311113113
 2611256554111211112112
 174) CLASS= 7 CLASS ASS= 6 NOTDEF=0
 DISTANCES= 200 200 200 200 0 4 120
 22215522222221111111111122
 256125466666441111111111126
 179) CLASS= 4 CLASS ASS= 9 NOTDEF=1
 DISTANCES= 13 6 13 5 9 200 21 119
 41466676444444
 21256555446544
 181) CLASS= 6 CLASS ASS= 7 NOTDEF=0
 DISTANCES= 200 200 200 200 200 4 0 120

Table 14 continued.


```

182) 21111112122212121111112212212121112112
211111121264612121111111265126121261112112
CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 266 1 120
2111211111121111121111211112211121221222322111221221111111113111111111122121212122265112211
112112
2111211111121111121111212411111251112126126565441112412611111111121111111111261212121256645112611
112112
183) CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 10 0 120
3111111211122211111111211121211121222221231112
2111111211126611111111211121211121266661221112
184) CLASS= 4 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 200 1 120
411111111114444444544111111111112
211111111112555446456111111111112
186) CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 207 1 120
2112221122212111221116111112111122311115111112
21125511226121112611121111211126411112111112
187) CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 23 0 120
211111212211111112111111211111111121211111222111111111112
211111212611111112111111211111111121211111223111111111112
189) CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 214 0 120
231123122211211111112121231211121111132111111111112
24112412661121111112121221211121111125111111111112
193) CLASS= 2 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 204 1 120
211111125265113111113311111131131111134444
211111126544112111112311111121121111125666
194) CLASS= 6 CLASS ASS= 7 NOTDEF=0
DISTANCES= 200 200 200 200 200 18 0 120
222211112112111115125211111122122211222112111121111122
26661111211211111212461111112612661126611211121111126

```

PERFORMANCE= 70.92

CONFUSION MATRIX

30	0	0	0	0	0	0	0	0
0	18	0	2	0	3	1	0	12
0	0	13	0	0	0	0	0	0
0	2	0	8	0	0	1	0	4
0	1	0	1	1	1	1	0	3
0	0	0	0	0	21	10	0	2
0	2	0	0	0	26	26	0	2
0	0	0	0	0	0	0	6	0

* Table 14 continued.

8.3.1 Analysis of defect rejections

Defects are rejected (not classified) whenever there exists uncertainty about their class, expressed mathematically by equal MNEO distances from the corresponding string to two (or more) languages.

Case 1: Zero MNEO distances to two (or more) languages

This situation occurs when two grammars generate intersecting languages, hence the cause lies in the inaccuracies of the grammar inference process, discussed in the preceding paragraph. Errors of this type produce, for example, the rejection of the strings numbered 53, 66, 74, 87, etc.

Case 2: Equal non-zero MNEO distances to two (or more) languages

Imperfect strings for which the MNEO distance is not uniquely determined are rejected because the class probabilities are assumed equal. This assumption is not reflected in the data base, but in an on-line situation defects tend to repeat during a short time and average values taken over a long period are not meaningful.

Rejections of this type occur relatively frequently; for instance, the strings numbered 45, 49, 51, 65, 67, etc. This problem can be overcome only partially by improving the accuracy of the inferred grammars, because it is impractical to extend the grammars to incorporate more and more imperfect strings. The best solution is to extend the VTSC correction tables from integer to real values, following the ideas presented in par. 6.3 and thus reducing the number of situations in which imperfect strings are assigned equal MNEO distances to two languages.

8.3.2 Analysis of misclassifications

A careful analysis of the factors which determine incorrect classifications reveal the existence of a certain number of unrelated causes:

1) Similar structure on the specular videoprint: the best example of this type is presented by the scratches and laminations, which often cannot be distinguished on the specular videoprint.

2) Incorrect defect delineations produced by the clumping algorithm. Defect nr. 114, for example, consists of two oil spots which have been merged because the capture rectangle was too long, therefore look like a five stand ring and are classified accordingly. The opposite effect isolates the scratch represented by the string nr. 64 because the capture rectangle is too short, and the resulting fragment is classified as an oil spot.

3) Unsatisfactory separation capabilities of the selected syntactic elements, combined with imprecisions in the grammar inference process produce a small number of misclassifications.

8.4 Conclusions

The observations listed above confirm certain limitations of the techniques which had to be expected; possible solutions to these problems have already been suggested in the preceding sections of the thesis.

A parallel investigation on the application of feature space techniques to the classification of tinplate defects is currently being undertaken at the Instruments Systems Centre, but comparative

results are not available at this time. The recognition performance of a feature space classifier can be expected to be better, for two reasons:

- 1) incorporation of features extracted from analog data is straightforward and enables a better separation between classes of defects which are similar on the videoprint;
- 2) some defects on the specular videoprint are not well structured and any syntactic pattern recognition technique is likely to give a lower performance in these cases.

The syntactic pattern recognition system developed in the thesis is intended to have general applicability to the classification of dynamically changing two-dimensional patterns. Its performance has been tested on tinplate defects because an industrial need and a potential data base existed, but the final decision on the feasibility of this approach will depend on the nature of a specific problem.

9. LOGIC DESIGN OF A REAL-TIME SYNTACTIC PATTERN RECOGNITION SYSTEM.

In the previous chapter it has been shown that the algorithms developed for grammar inference and defect classification provide a good performance with real data, in an off-line simulation experiment. The final questions are related to the practical aspects of the problem:

- 1) is it possible to implement this method at the very high speed required on-line ?
- 2) if so, how complex is the hardware implementation of a syntactic pattern classifier ?

It was considered that the most convincing answer to these problems is to design the logic structure of a syntactic pattern classifier, based on the methods developed, and then to evaluate it in terms of computation time and hardware complexity. The structure of the system designed in this chapter is intended to be general, but the quantitative parameters (number of classes, number of states, number of symbols, etc) refer to the present problem. Since the main objective is to show the feasibility of the hardware implementation in terms of speed and complexity, the design is not carried out in detail.

9.1 General description of the proposed syntactic pattern recognition system

The general structure of the syntactic pattern recognition system, introduced in par 2.3, consists of two basic

blocks which participate in the real-time classification process: the pattern description block and the pattern classifier. The theoretical principles underlying the operation of these blocks have been developed in chapters 4 and 6, respectively, which also contain a detailed specification of the functions which must be realized.

The allocation of hardware resources for implementing these functions must take into account the different speed at which they have to be performed. The solution developed is outlined in the block diagram in fig. 9.1 and presents the following features:

1. the pattern description block is relatively simple but has to operate at the high sampling rate (1-5 MHz), hence TTL-MSI circuits are an optimum solution.
2. the SIUs have to be stored in a buffer memory, to enable the subsequent operations to be performed at the average rate, rather than the maximum rate.
3. the pattern classifier was designed to use a dedicated microcomputer, which interacts with the SIU buffer while it is not being accessed by the high speed pattern description block; the microcomputer offers an economic solution and will be shown to satisfy the speed parameters of the system.

The signals which control the operation of the system were specified according to the block to/from which they are transmitted as follows:

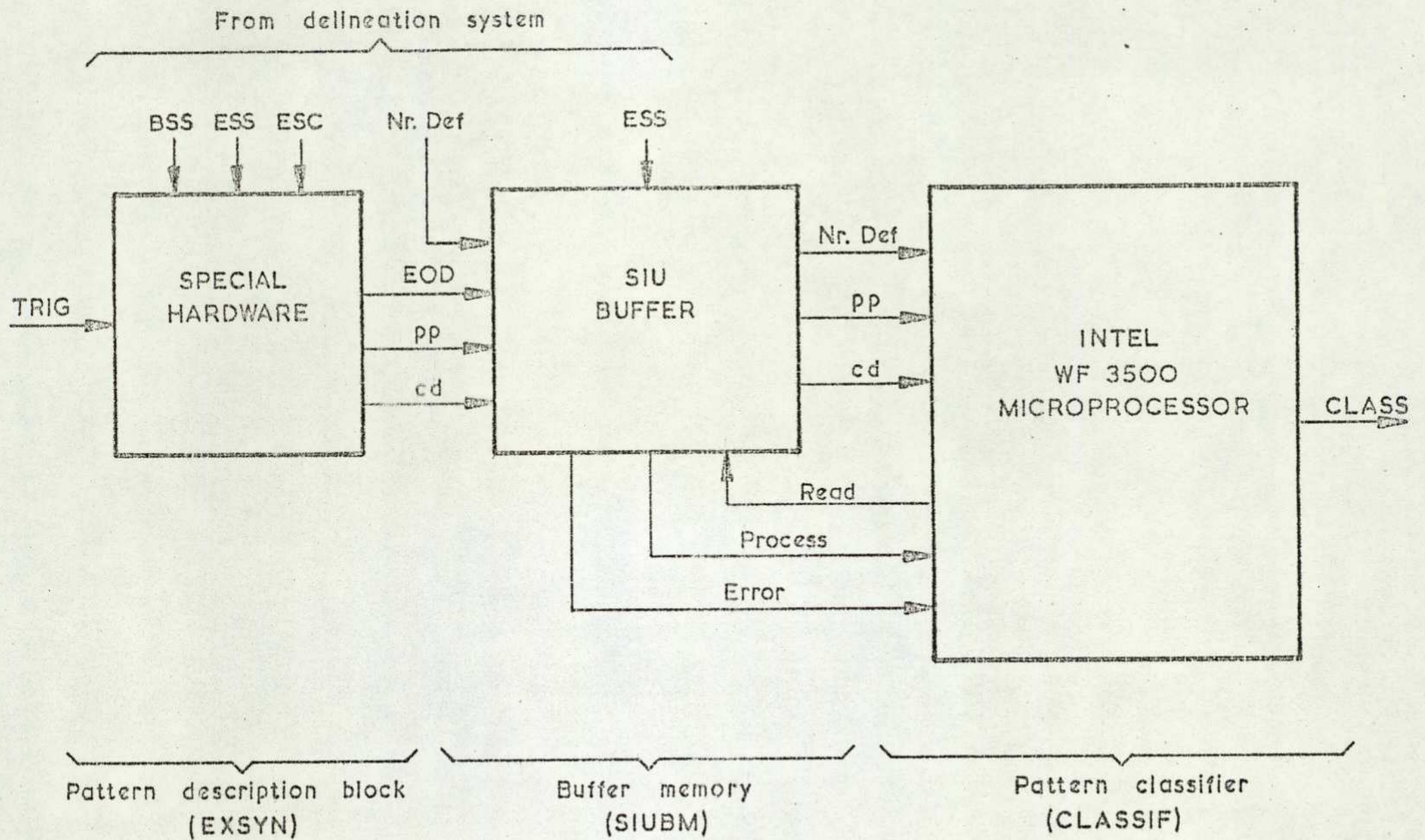


Fig. 9.1 Block diagram of syntactic pattern recognition system

1. Pattern description block
 - a) input signals
 - from the delineation system:
 - BSS - beginning of scan section
 - ESS - end of scan section
 - ESC - end of scan
 - from the scanner:
 - TRIG - trigger samples
 - b) output signals - all to the SIU buffer:
 - pp - pattern primitive
 - cd - composition descriptor
 - EOD - end of defect
2. SIU buffer
 - a) input signals (not previously specified)
 - from the delineation system: Nr. Def - defect number
 - from the classifier (microcomputer):
 - Read - Read next SIU, if not busy
 - b) output signals (all to the microcomputer)
 - Nr. Def, pp, cd - SIU to be processed
 - Process - indicates that not all SIUs stored in the buffer have been processed
 - Error - buffer overflow
3. Pattern classifier (microcomputer)
 - a) input signals: already specified
 - b) output signals (not previously specified):
 - CLASS - the class of the defects whose end had been detected.

The functions performed by the blocks shown in fig. 9.1 and their input - output signals represent the basis on which the logic design of these blocks was carried out. Since the design methodology is of secondary importance, the following sections concentrate on the description of the resulting system.

9.2 Logic structure of the pattern description block

It has been shown in chapter 4 that the pattern description block performs the following functions:

1. extraction of pattern primitives
2. extraction of composition descriptors
3. generating and counting the NOS syntactic elements

(for active defects not encountered during a scan)

9.2.1 Extraction of pattern primitives (fig. 9.2)

This function requires the computation of two parameters, as described in par 4.1.1:

- 1) the number of segments
- 2) the length of the segments

The logical structure of this block is represented in fig. 9.2.

It has been found that the first parameter can be computed by a simple FSA which classifies all strings of 1's and 0's in three classes defined by the following regular expressions:

$$S_1 = 11^* 00^*$$

$$S_2 = S_1 00^* S_1$$

$$S_3 = S_2 00^* S_1$$

The second parameter is computed using two counters, CLS_1 and CLS_2 , which determine the number of 1's in the first and

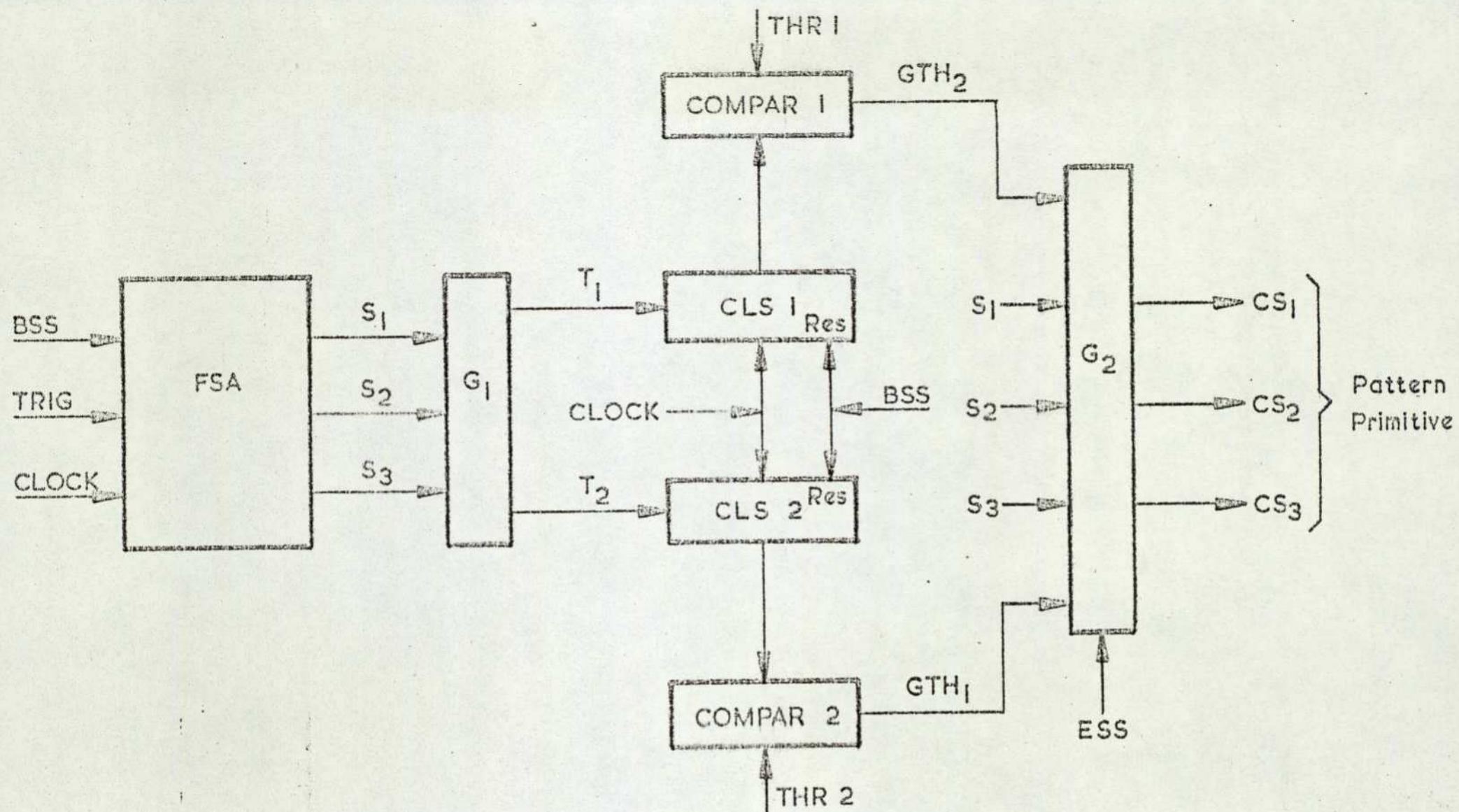


Fig. 9.2 Extraction of pattern primitives

second segments, respectively. The numbers stored in these counters at the end of the scan section are transmitted to the comparators $COMP_1$ and $COMP_2$, which are used to determine their position in the intervals defined by THR_1 and THR_2 .

At the end of the scan section, the resulting parameters are combined in a network of gates G_2 , to define the pattern primitive code $(CS_1CS_2CS_3)$, according to the specifications given in table 2 (see chapter 4).

9.2.2 Extraction of composition descriptors

This function requires the computation of two parameters, defined in par 4.1.2:

- 1) the degree of connectivity
- 2) the type of connectivity

The logical structure of this block is represented in fig. 9.3.

The first parameter is computed using two counters, CNC and CNUC, for the numbers of connected and nonconnected trigger samples, respectively, which are incremented by the following signals:

$$INC = TRIG(i) \cdot PTRIG(i)$$

$$INUC = TRIG(i) \oplus PTRIG(i)$$

At the end of the scan section, the degree of connectivity is computed by comparing the numbers $(2NUC, NC)$ and $(NUC, 2NC)$, using the comparators $COMP_1$ and $COMP_2$, respectively.

The second parameter is calculated simply by employing set-reset flip-flops, which indicate whether the current and

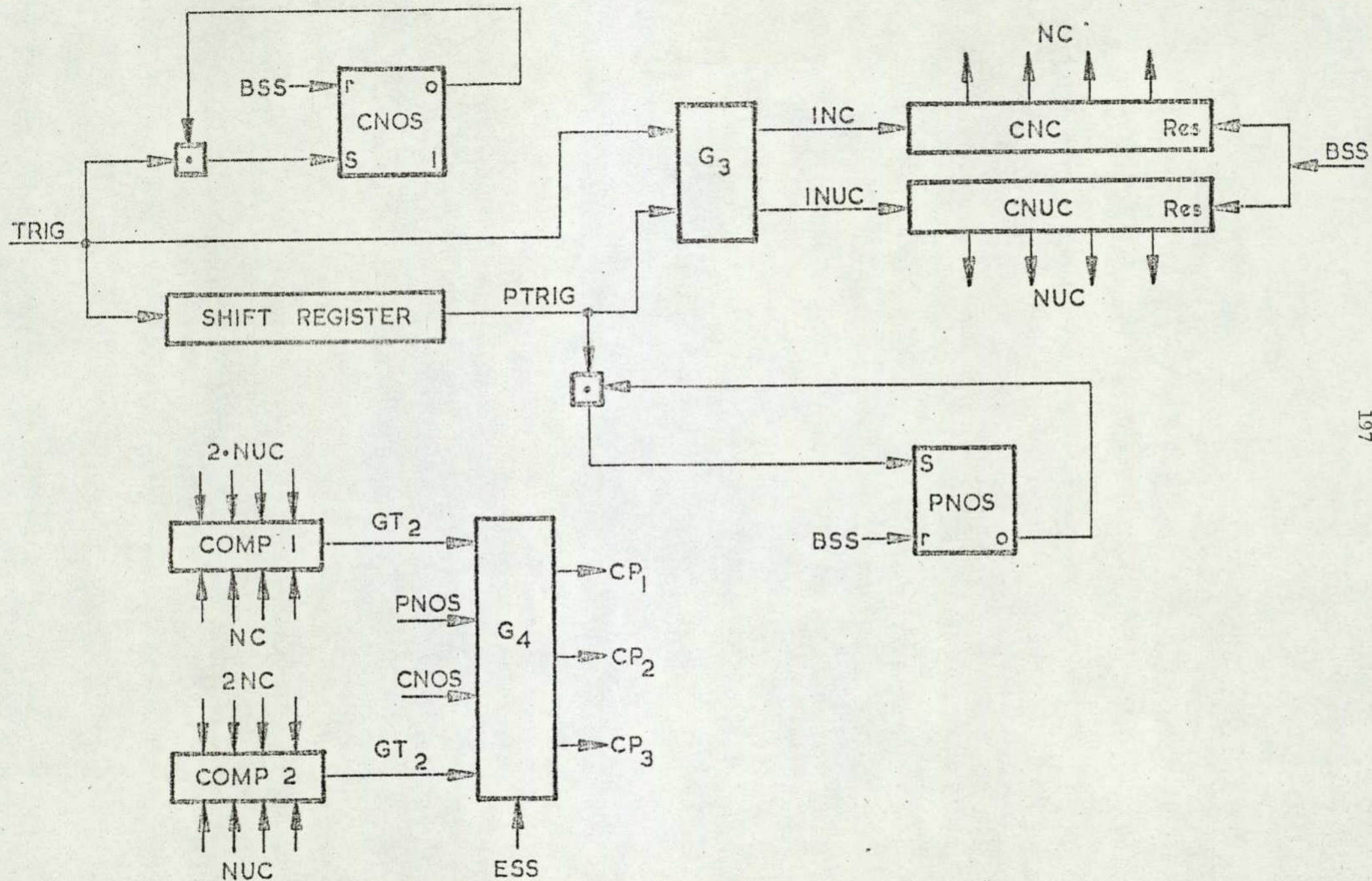


Fig. 2.3. Extraction of composition descriptors

previous scan sections contain 1-trigger samples (CNOS and PNOS, respectively).

At the end of the scan section, the resulting parameters are combined in a network of gates G_4 to define the composition descriptor code ($CP_1 CP_2 CP_3$), according to the specifications given in table 3 (see chapter 4).

Conclusion

The structure of the blocks which extract the pattern primitives and composition descriptors contains simple bipolar MSI circuits which can operate at frequencies far higher than the sampling rate (1-5MHz).

9.2.3 Generating and counting the NOS syntactic elements

In principle, this block implements the functions of the array DEFST used in the CLUMPL simulation program (fig. 9.4):

- 1) indicates whether a defect is active or not, using a flip-flop A which:
 - is initially reset
 - is set when a scan section is assigned the corresponding defect number
 - is reset when the defect is rendered inactive because of not being encountered during K_y successive scans.
- 2) indicates whether a defect has been encountered during a scan or not, using a flip-flop E which:
 - is reset at the beginning of each scan
(from the delayed end of scan signal EOS)
 - is set when a scan section is assigned the corresponding defect number.

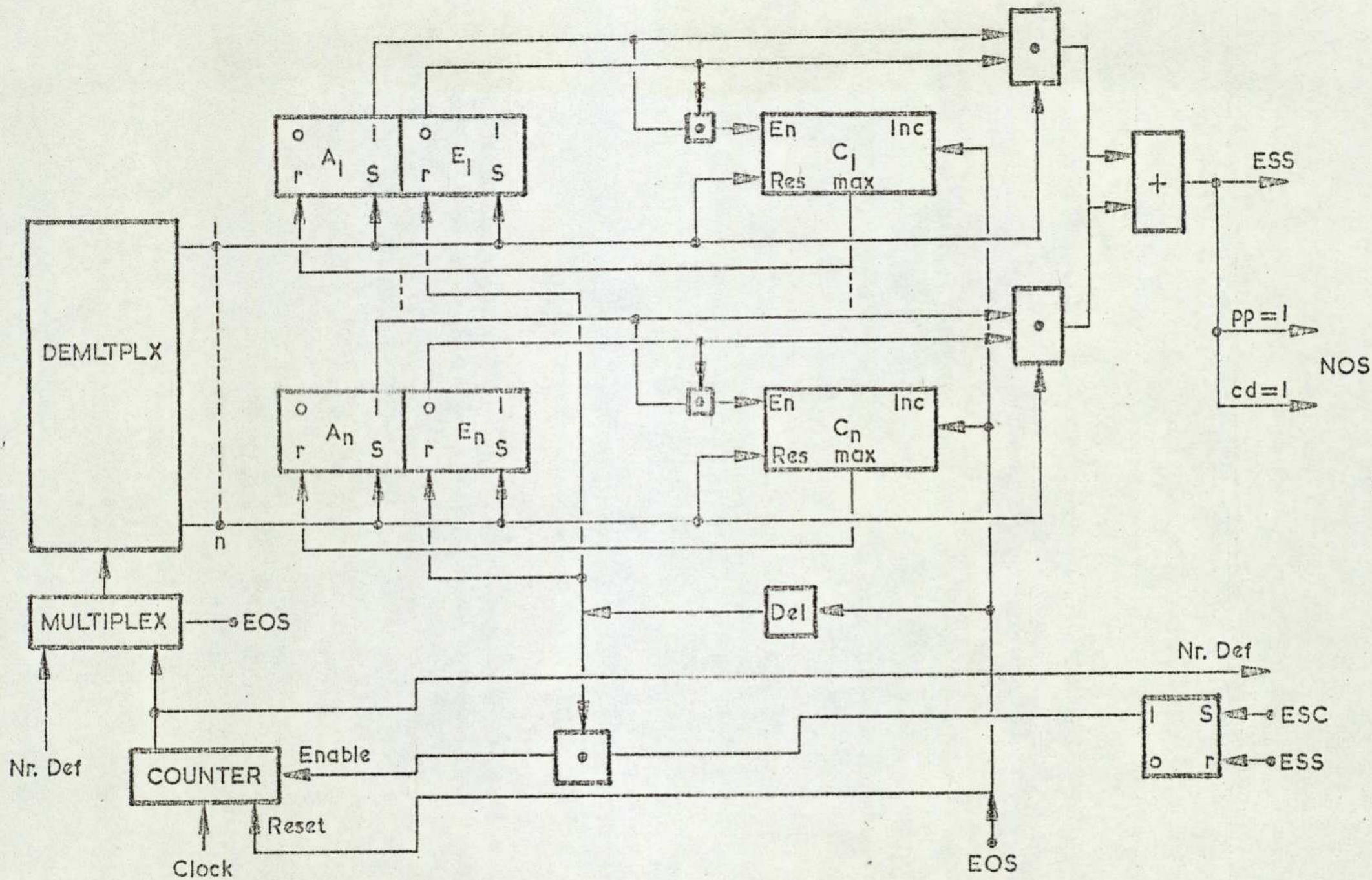


Fig. 9.4 Generating and counting the NOS syntactic elements

3) counts the number of successive scans over which a defect is not encountered, using a counter C, modulo K_Y , which is:

- reset each time a scan section is assigned the corresponding defect number
- incremented at the end of each scan for the active defects not encountered

The counter generates an overflow signal when its contents become K_Y , which resets the corresponding A flip-flop, marking the termination of the defect.

The elements A, E and C are implemented a number of times equal to the maximum number of simultaneously active defects. A demultiplexer DMPLX is used for selecting the required defect number. At the end of each scan, all A and E flip-flops are tested sequentially and a NOS syntactic element is generated for each active defect not encountered. This function is performed by incrementing an additional COUNTER whose outputs are connected to the defect selector through a multiplexor MULTIPLEX. The entire operation can be executed in the time interval available between the end of a scan and the end of the first scan section of the next scan.

When a defect is rendered inactive, the last SIU indicates this by assigning the value 0 to the pattern primitive and the composition descriptor.

9.3 The SIU Buffer Memory (SIUBM)

The SIUBM block, represented in fig. 9.5, stores the SIUs generated by the description block in a buffer realized as a RAM with a number of distinct features:

1. it is connected to two autonomous blocks which operate asynchronously; the description block writes data in the RAM at a high rate, but not frequently; the pattern classifier retrieves data from the RAM at a rate which low, but constant.
2. writing data has priority over retrieving data, but there is no need for interrupts; current read cycles are always terminated normally, but a new read cycle can not be initiated while the SIUBM is busy writing a SIU, indicated by the ESS signal.
3. the contents of the RAM are addressed sequentially, in increasing order, using two pointers implemented as counters:
 - a) a pointer NEWSIU to the location used for storing the last SIU; this pointer is incremented by the ESS signal, which then initiates a write cycle for the new SIU.
 - b) a pointer LASIUP to the location from which the last SIU was retrieved and processed by the classifier; this pointer is incremented when the READ signal transmitted by the classifier is accepted, initiating a read cycle.

The contents of these two pointers are compared continuously, to indicate the state of the buffer:

- if $(LASIUP) < (NEWSIU)$, there are SIUs waiting to be processed, indicated by the signal PROCESS.
- if $(LASIUP) = (NEWSIU)$, the buffer is considered empty, since all SIUs have been processed; when this situation is detected, both counters are reset to 0.

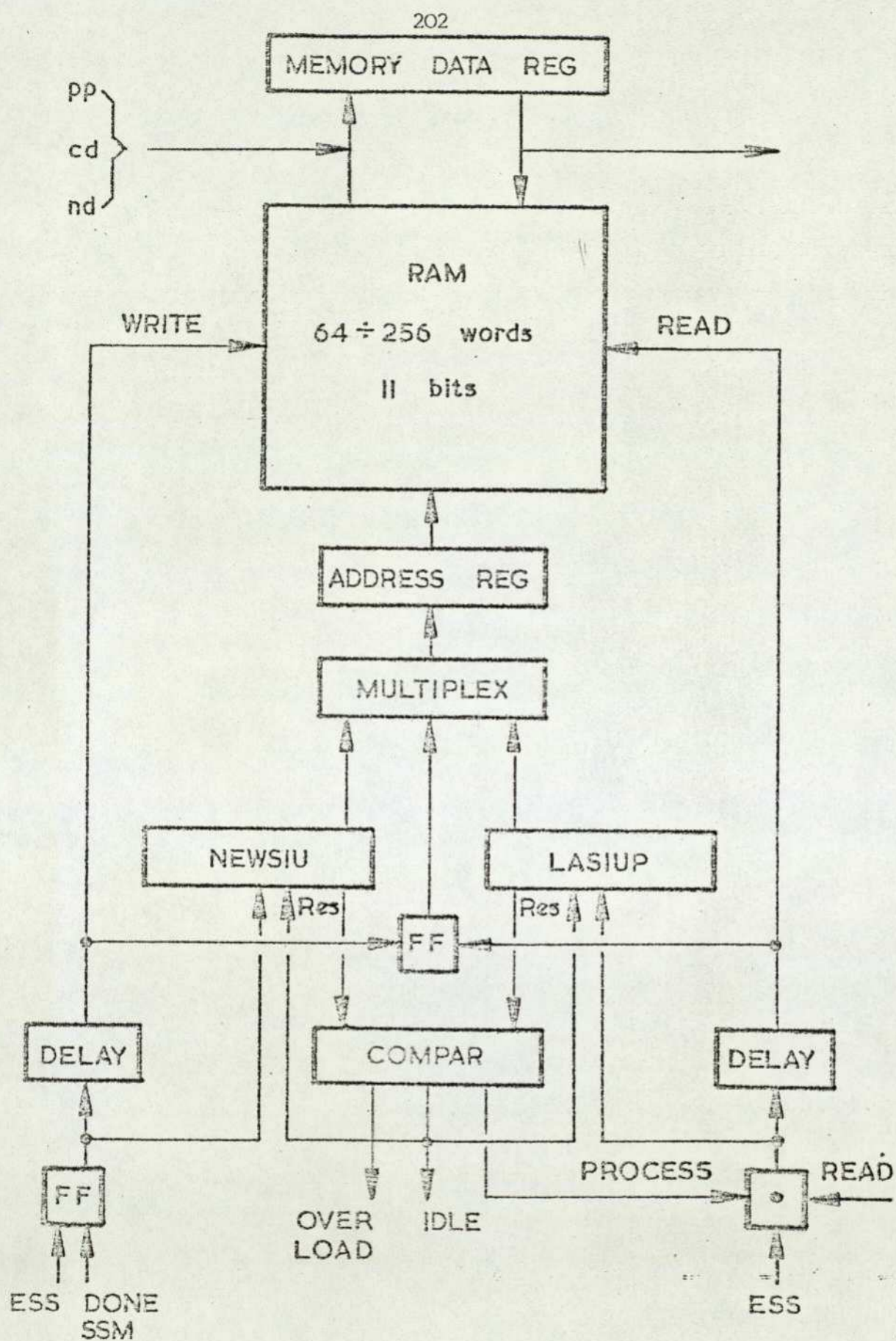


Fig. 9.5 SIU Buffer memory

- if (LASIUP) > (NEWSIU), the buffer has been overloaded and an ERROR signal is generated.

Both pointers access the RAM on a modulo NB base, where NB is the maximum number of SIUs which can be stored. Precautions must be taken to avoid wrong overflow detections which can occur when the pointer LASIUP makes the transition from NB to 0; this problem can be solved by providing extra bits in the counters, used for comparisons, but not for accessing the RAM.

9.4 Implementation of a syntactic pattern classifier using a microcomputer

The classification algorithm described in par 6.2 revealed that the functions which must be performed by the classifier involve only simple computations, a certain amount of RAM for holding intermediate results and ROMs for encoding the syntactic elements and for storing the correction tables. These factors suggest that a microcomputer could represent an economical solution. The present paragraph describes how the architecture of a microprogrammed microcomputer can be adapted for performing the classification algorithm; it will be shown in the next paragraph that the speed of operation is adequate, if bipolar technology is used.

The main difficulty encountered is due to the fact that successive SIUs do not always belong to the same defect, because more than just one defect can be active simultaneously; moreover, these defects are not necessarily terminated in the order in which they were encountered.

A simple method for dealing with this problem is to choose a maximum number of defects, `MNDEF`, which can be treated at any time, and to provide sufficient memory to store the current state of each defect.

Since each SIU specifies the defect number, this can be used to select the memory area reserved for that defect and update the parameters associated to the defect, by processing the SIU.

The flowchart depicted in fig. 9.6 outlines the interaction between the microcomputer and the SIUEM block and the sequence in which the classification algorithm is executed:

- 1 . a new cycle is started when, after finishing the processing of a SIU, the signal `PROCESS` generated by the SIUEM is still high.
2. the next SIU is retrieved from SIUEM by sending a `READ` command.
3. the defect number component of the SIU is used to select the corresponding area of RAM
4. the `pp/cd` component is tested to determine if it is the terminator of a defect or not.
 - a) defect terminated: the defect is classified by executing the `MIN-DIST` subroutine, described in par 9.4.2; finally, the `CLASS` of the defect is transmitted to the output
 - b) defect not terminated: the variables which store the distances from the current defect (string) to each pattern language are updated as follows:

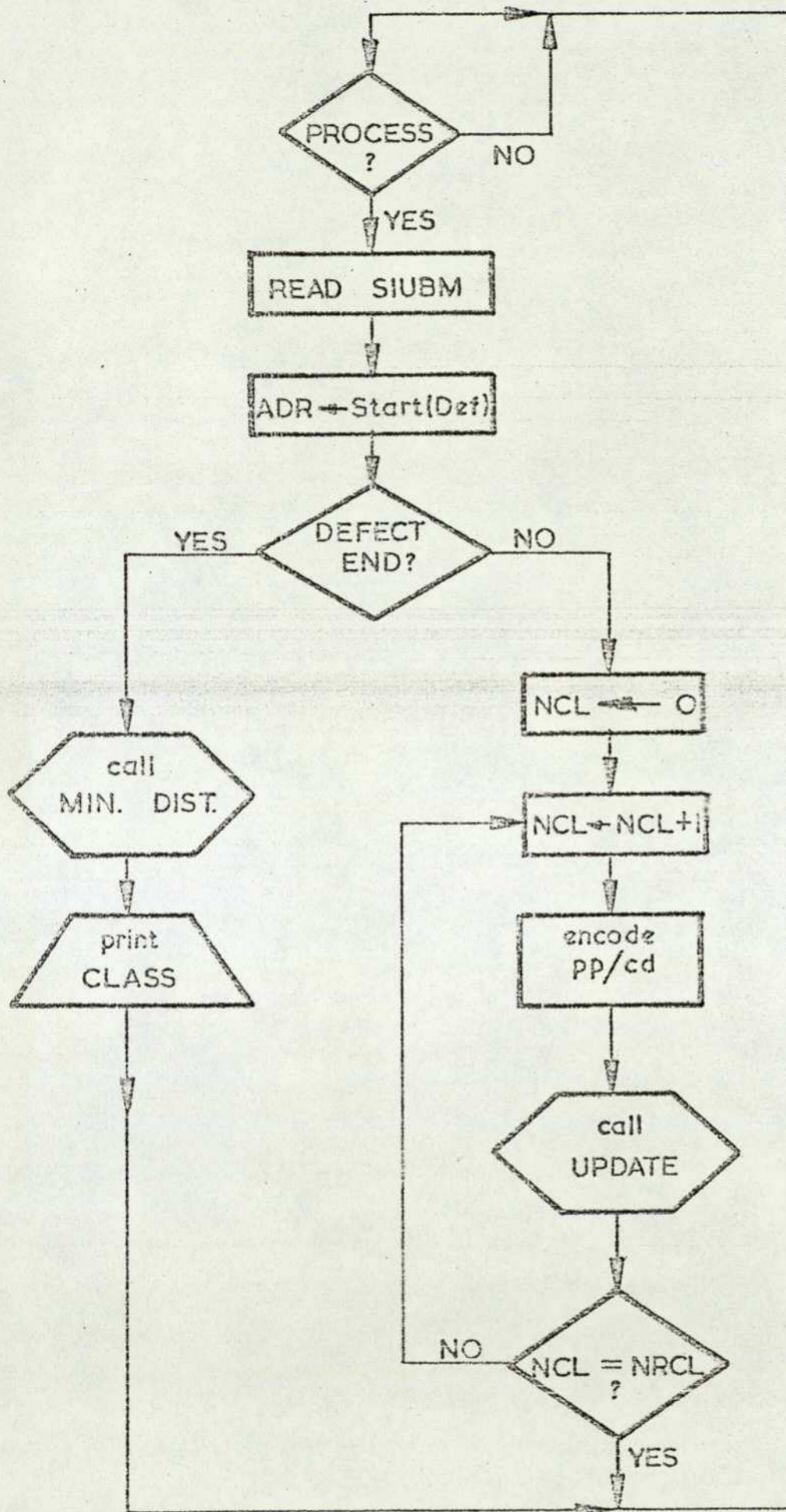


Fig. 9.6 General organization of classification program

- the pp/cd pair is encoded according to its significance for the corresponding language, as described in par 6.6.3 (subroutine IDSPEX).

- the distance from the string to the language is computed by executing the UPDATE subroutine, described in par. 9.4.3.

9.4.1 Selection of the microcomputer configuration

The microcomputer model which has been selected for developing the ideas presented into a specific structure is the INTEL WC-3000, for the following reasons:

1. it is a fast, bipolar microcomputer
2. it is a microprogrammed computer, which allows for development of custom-oriented instructions.
3. it has a N-bit word expandable multibus organisation.

The configuration of the computer will be determined considering the set of parameters which are required by the present application:

1. number of classes: $N_{CL} = 8$
2. number of states/FSA: $|Q_A| = 10$
3. number of pattern primitives: $|pp| = 8$
4. number of composition descriptors: $|cd| = 8$
5. number of symbols/class: $|\sum_{iA}| = 8$
6. maximum number of defects which must be treated simultaneously: $NMDEF = 25$

These considerations lead to a microcomputer configuration as shown in fig. 9.7, which is based on the following specifications:

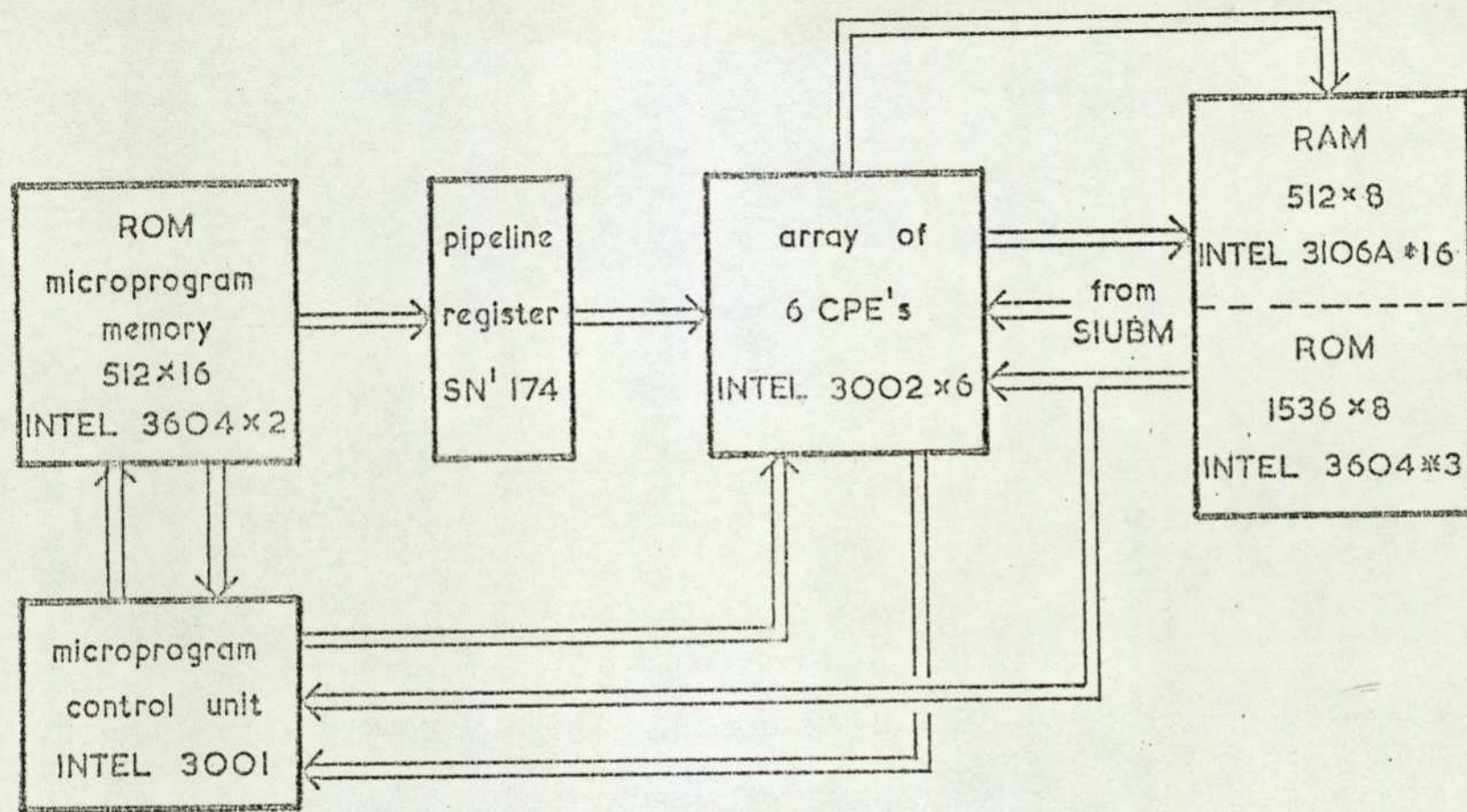


Fig.9.7 Microcomputer — based classifier

1. The RAM stores the current distances from strings to languages; these are small, positive integers in the range 0 - 127 for FSA with $|Q_A| = 10$ states. Since both the old and new values must be stored for $|Q_A| = 10$ states and $NDMAX = 25$ defects, the capacity required is 500 words of 6 - 8 bits.

2. The ROMs store three different tables, namely:

a. the pp/cd pair encoding table, requiring $|pp| \cdot |cd| \cdot |N_{CL}| = 5.2$ words of $\log_2 |\sum_{iA}| = 3$ bits or 256 words of 6 bits

b. the $V(T,S,c)$ table, storing $|Q_A|^2 \cdot |\sum_{iA}| = 512$ words of minimum 6-8 bits.

c. the defect number to memory address conversion table, requiring $NMDEF = 32$ words of 6-8 bits.

Since the microcomputer is intended to perform exclusively as a classifier and the program must be nonerasable, additional ROM capacity is also necessary for storing the program and other constants. The final memory configuration can be estimated at 512 x 8 RAM and 1536 x 8 ROM, a total of 2078 words. This implies a 11-bit address and hence 6 CPE's are needed to enable direct access of the entire memory. A 6 x 2 = 12 bit CPE is also adequate for performing all arithmetic computations in parallel.

9.4.2. Implementation of the MIN-DIST subroutine

The function of this subroutine can be regarded as the direct computation of rel.(6.7), over a number of classes N_{CL} . In order to distinguish between final and non-final states, bit 8 of all RAM locations is used as a flag (1/0, respectively), since 7 bits

are adequate for representing the distances. This solution is advantageous since it reduces the classification process to a sequential search through the RAM for the minimum distance to a final state, and the corresponding state will therefore define the class of the string. The flowchart of this subroutine is represented in fig. 9.8 in terms of microcomputer operations and uses the following notations:

- NRCL - number of classes
- NCL - current class
- NRST(i) - number of states of class i-acceptor
- ADR - current address in the memory area reserved for the defect being classified
- LADR - transition address from a class to another in the memory area
- MIN - current value of minimum distance
- MAX - constant 11111111
- CLASS - class corresponding to state which defines MIN distance

9.4.3 Implementation of the UPDATE subroutine

This subroutine performs the operations required to update rel. (6.6) for a new SIU, represented by the symbol *c* corresponding to the encoded primitive/descriptor pair. The flowchart shown in fig 9.9 describes the execution of the subroutine as a sequence of microcomputer operations; the notations are as in par 9.4.2.

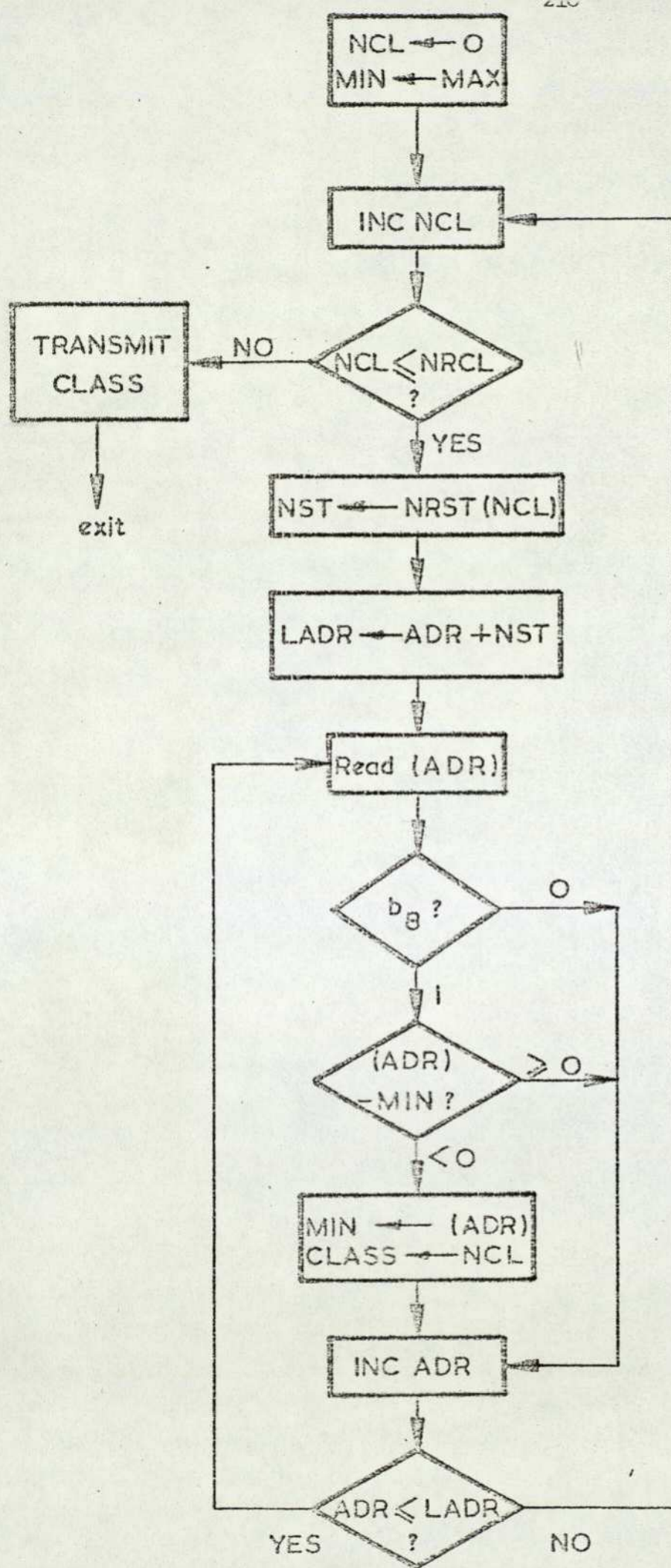


Fig.9.8 Flowchart of subroutine MIN-DIST.

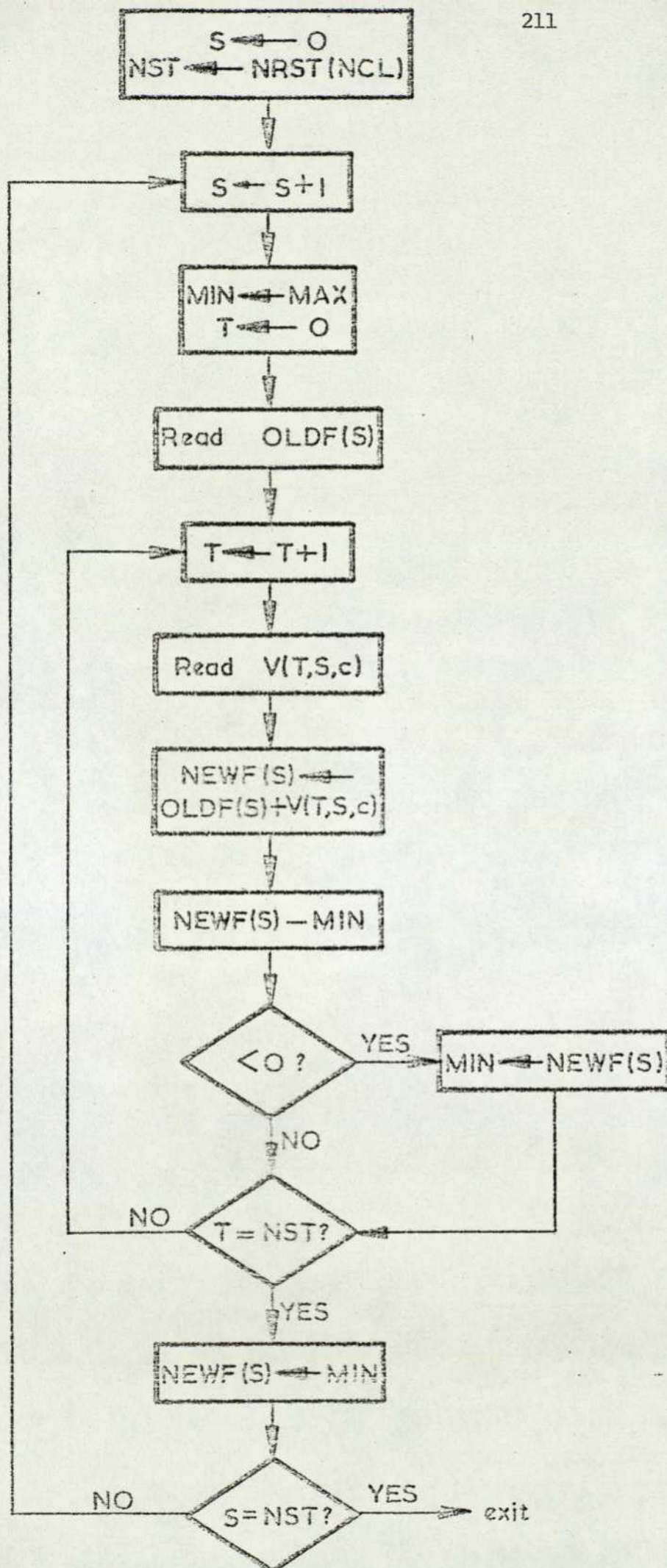


Fig. 9.9 Flowchart of subroutine UPDATE

9.5 Estimation of speed parameters

Since the computation time is an essential feature in the present case, it is necessary to make quantitative estimations on the requirements imposed on the classifier by the system parameters.

Trigger samples are generated at a frequency in the 1- 5 MHz range. The pattern description block described in par. 9.1 can be realized as a relatively simple network of TTL MSI circuits and therefore is capable of operating at frequencies higher than 1 MHz.

A typical dimension of the capture window in the transversal direction is 15 trigger samples, which corresponds to a 15 μ s interval. If the pattern description block operated autonomously and SIUs were generated continuously, the time available for processing completely one SIU would be 15 μ s. In general, the time available is much longer, because scan sections are encountered at a lower rate:

1. it can be assumed that most scans do not contain defects, otherwise there is a major fault in the production process.
2. when defects do occur, normally only one defect is encountered during each scan.
3. situations in which more than 3 defect sections are encountered during one scan are rare.

It is not justified economically to design a system which is capable of operating stationary at a rate dictated by the worst conditions. The important factor is to process the defects in the

average time available and therefore the solution adopted was to introduce an asynchronous buffer memory between the description block and the classifier.

The number of trigger samples generated during a complete scan is about 1,000, which gives a 0.2 msec interval between successive scans, for a 5 MHz sampling frequency. If one estimates that on the average 10% of the scans contain one scan section resulting from a defect, the average time available for processing one scan section, or SIU, is

$$T_a = 2 \text{ msec} \quad (9.1)$$

On the other hand, the time complexity of the classification algorithm can be computed by applying rel (6.16) to the multi-class case:

$$T_C = \sum_1^{N_{CL}} |Q_i|^2 \quad (9.2)$$

which represents the number of times the relation (6.6) must be computed, for one SIU.

The value of T_C can be evaluated for a typical situation, characterised by $N_{CL} = 8$ and average $|Q_A| = 10$, in which case

$$T_C = 800$$

A preliminary evaluation indicated that the number of computations involved in the classification algorithm is too high to be implemented in a MDS microcomputer. However, further investigations showed that a fast bipolar microcomputer, for example INTEL WF 3000, meets both the speed and cost requirements.

The time required to perform the computations involved in evaluating rel. (6.6) can be determined with approximation by analysing the flowcharts which describe this process as a sequence of microinstructions. It is convenient to define three cycles in the microprogram which performs these microinstructions (see flowcharts in fig. 9.6 and 9.9).

1. the main cycle (T_{MC}): retrieves the next SIU and detects whether it indicates the end of a defect or not.
2. the class-cycle (T_{CC}): computes the distance from the current string to a pattern class
3. the state-cycle (T_{SC}): computes the distances from the current string to all the states of an acceptor.

The main parameters of the microcomputer with respect to the speed of operation are the microinstruction cycle (t_{ic}) and the random access memory cycle (t_{mc}), which are specified in [39] and [40] : $t_{ic} = 150$ ns and $t_{mc} = 450$ ns. The actual computation time can therefore be determined by following the operations shown in the flowcharts and counting separately the occurrences of these two types of operations:

$$T_{MC} = 4 t_{ic} + t_{mc} + N_{CL} \cdot T_{CC} \quad (9.3)$$

$$T_{CC} = 3 t_{ic} + T_{UPDATE} \quad (9.4)$$

$$T_{UPDATE} = t_{ic} + |Q_A| \cdot T_{SC} \quad (9.5)$$

$$T_{SC} = 3 t_{ic} + 2 t_{mc} + |Q_A| \cdot 7 \cdot t_{ic} \quad (9.6)$$

By substituting in (9.3) - (9.6) the parameters:

$t_{ic} = 150$ ns, $t_{mc} = 45$ ns, $|Q_A| = 10$, $N_{CL} = 8$, the duration of the main cycle thus obtained is shown to be shorter than the time available

$$T_{MC} \approx 1 \text{ ms} < T_a \approx 2 \text{ ms}$$

The calculations presented are based on the assumption that all T_c basic operations are performed serially. The WF - 3000 microcomputer employs a N-word expandable multibus which could be used for processing in parallel N_{CL} basic operations. The time complexity can be therefore reduced almost N_{CL} times, but this solution is justified only for more complex applications, since it increases the cost of the hardware by the same factor.

Additional to the SIU processing time, there are other computations which have not been considered in the preceding discussion. The most important is the actual classification time, which can be computed by examining the flowchart shown in fig. 9.8; the value obtained ($80 \mu s$) is negligible when considering its contribution to the average SIU processing time.

9.6 Conclusion

The architecture of the syntactic pattern recognition system which has been developed in this chapter meets the speed requirements of the present application. The hardware involved consists mainly of a fast 8 bit word microcomputer, a RAM buffer and a network of the TTL - MSI circuits. Although the design has not been worked out in detail and the evaluation of the computation time is approximate, the conclusion is that the structure suggested in this chapter is technically feasible.

10 . CONCLUSIONS

The thesis presents the results of a theoretical and experimental study on the application of syntactic pattern recognition to the classification of dynamically changing two-dimensional patterns. Existing theoretical methods have been adapted and extended into a unitary syntactic pattern recognition system which incorporates original contributions needed to satisfy the specific requirements of this type of application:

- 1) development of a language for the description of two-dimensional defects which is suited for on-line processing;
- 2) formulation of an efficient algorithm for the inference of a grammar which approximates a given language;
- 3) generalization of a simple classification algorithm based on the concept of syntactic distances, which reduces the errors due to the approximations of the languages.

A set of programs written in FORTRAN has been developed for performing a detailed analysis of the syntactic pattern recognition system in computer simulation.

The validity of the theoretical methods developed in the thesis has been tested on a data base of real defects, acquired using an automated data gathering system designed and constructed as part of the research programme reported in the thesis. The experimental results obtained in computer simulation using this data base show that the syntactic classifier provides an acceptable recognition performance, of about 70 - 80% correct classifications.

A hardware implementation of the proposed system is outlined and shown to be economical and capable of operating within the speed requirements of on-line applications.

The performance of the present system can be improved by incorporating a certain number of refined techniques which are open to further investigation:

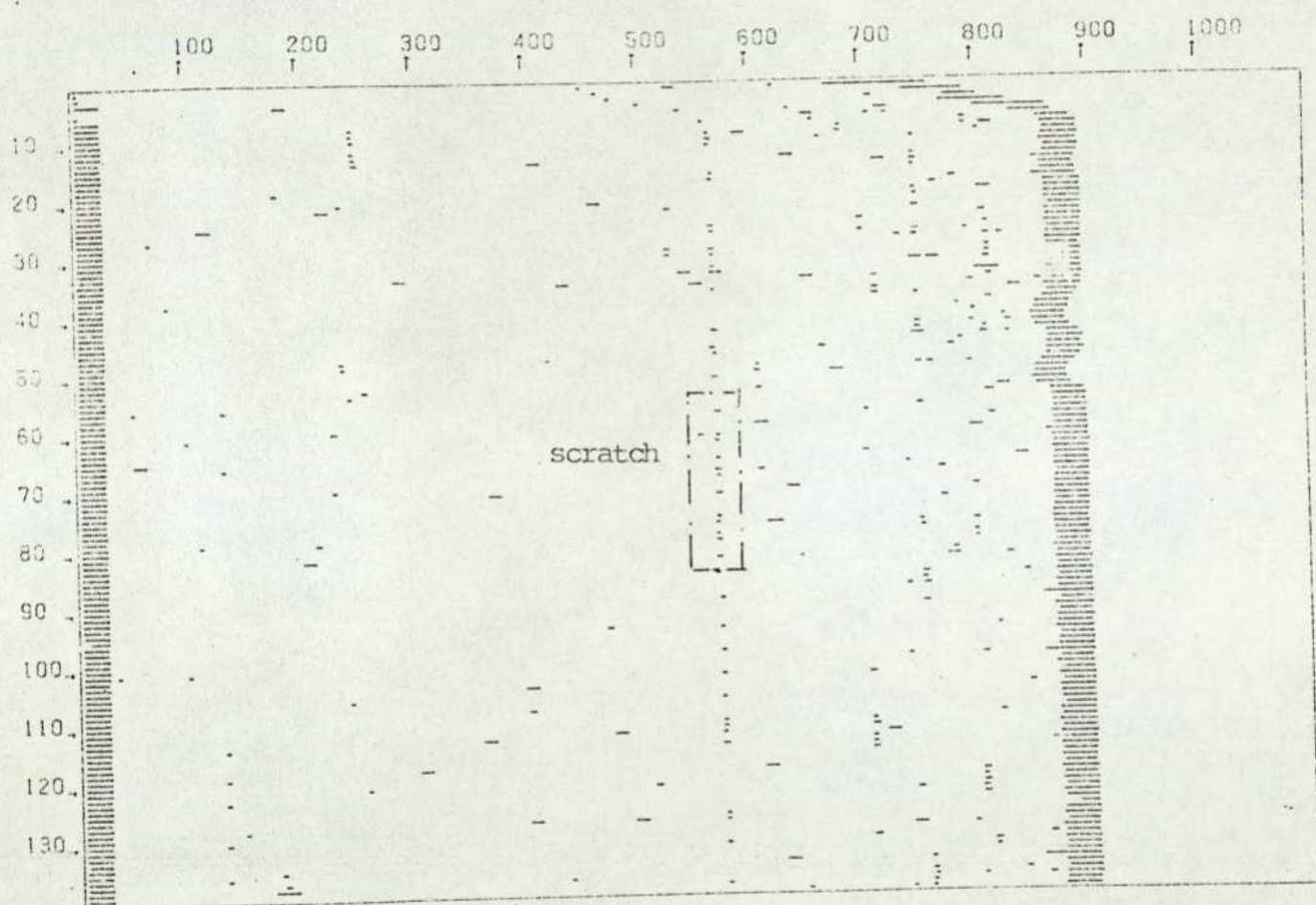
- 1) improved clumping algorithms for a better defect delineation;
- 2) extension of the theoretical methods to incorporate additional information provided by analog data and off-specular video-prints;
- 3) generalization of the concept of MNEO distance to include statistical properties of the languages;
- 4) execution of the grammar inference algorithm in a interactive mode, using a graphic display.

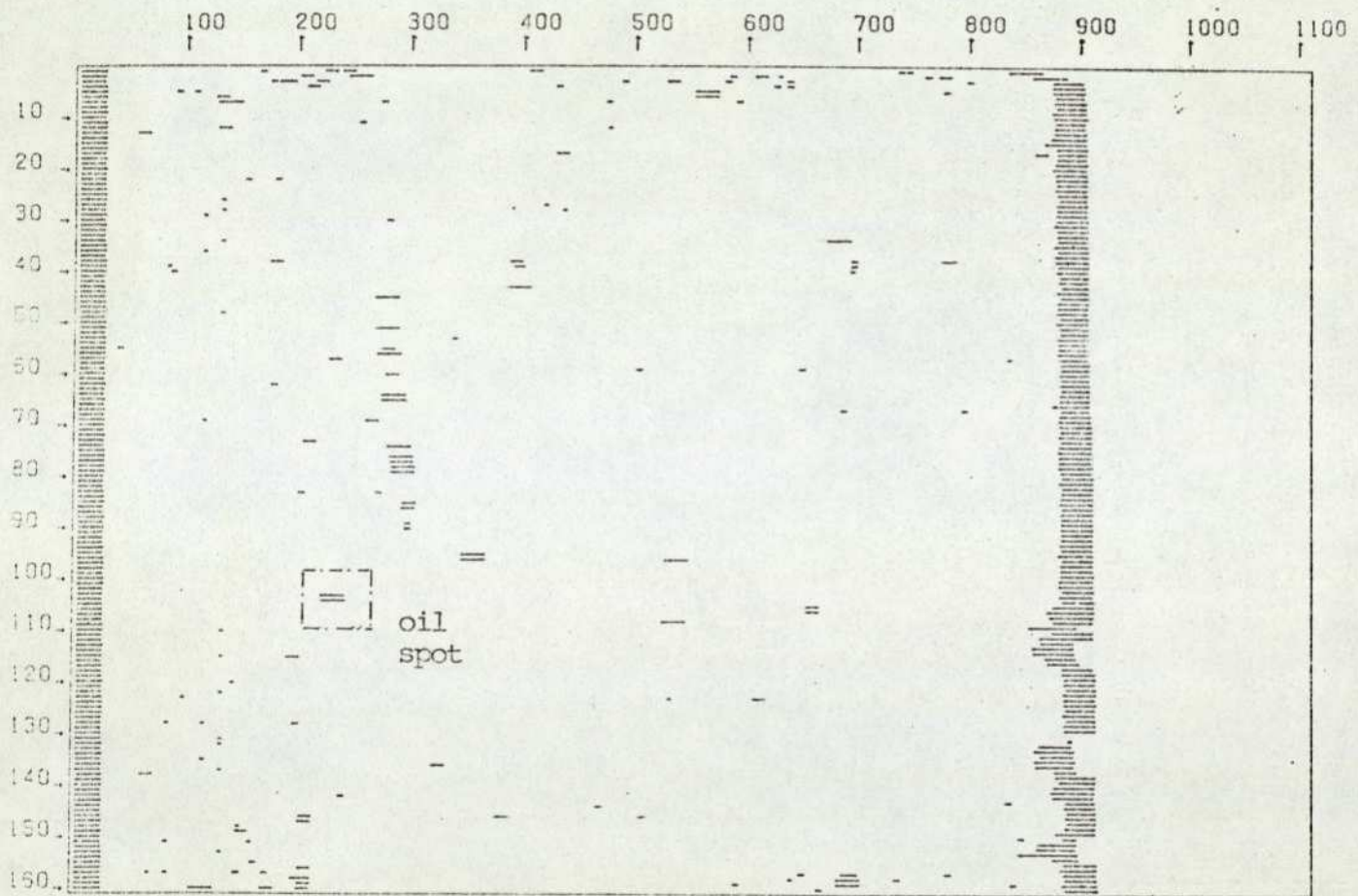
The syntactic pattern recognition system developed in the thesis represents a general framework in which these refinements can be accommodated to improve the recognition performance, without substantial increases in the complexity or reductions in the speed of operation.

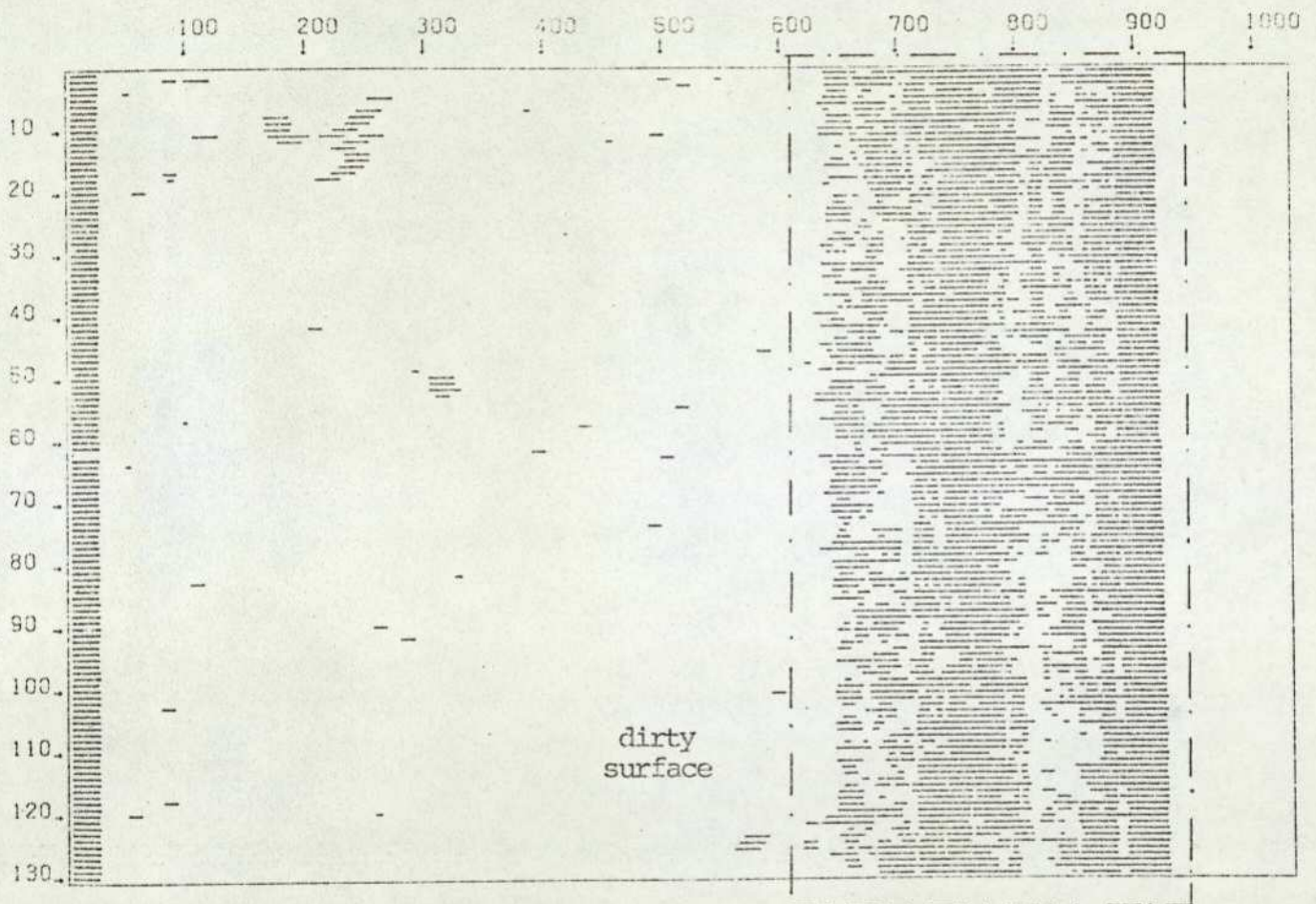
The theoretical principles underlying the operation of the system are intended to be general, but the final decision on the feasibility of its application will depend on the nature of the problem and the structural properties of the defects to be classified.

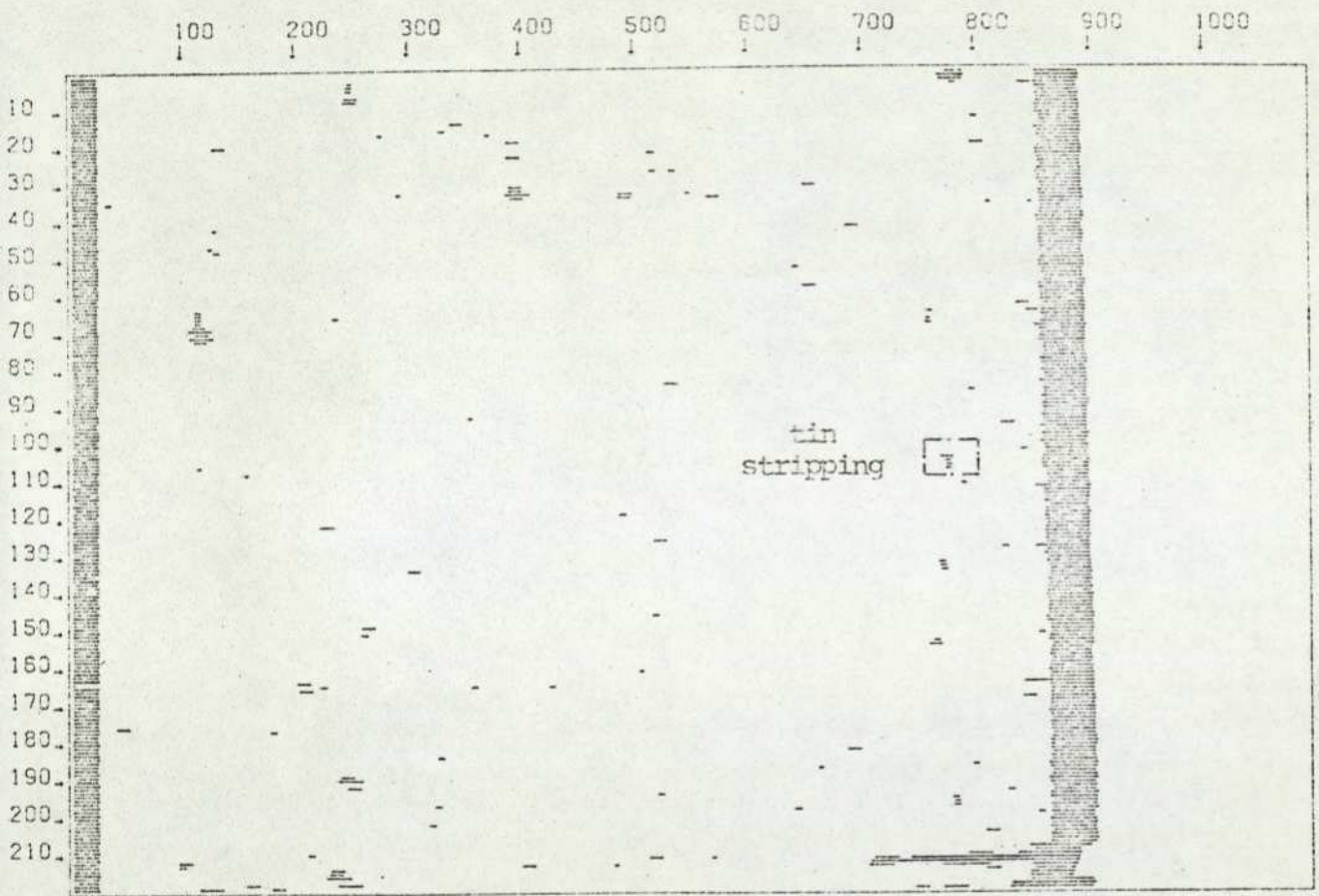
APPENDIX I

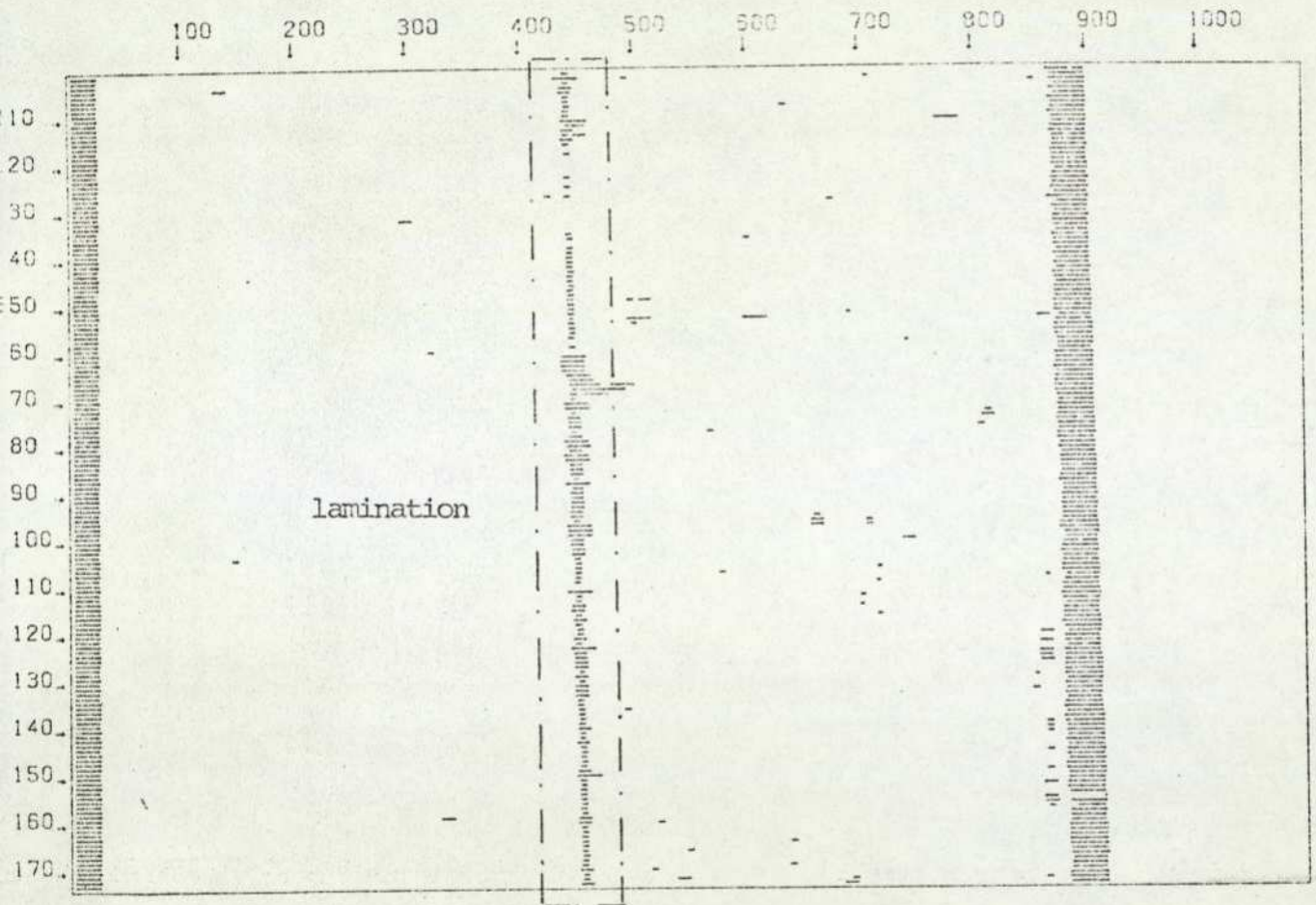
VIDEOPRINTS OF TINPLATE SHEETS

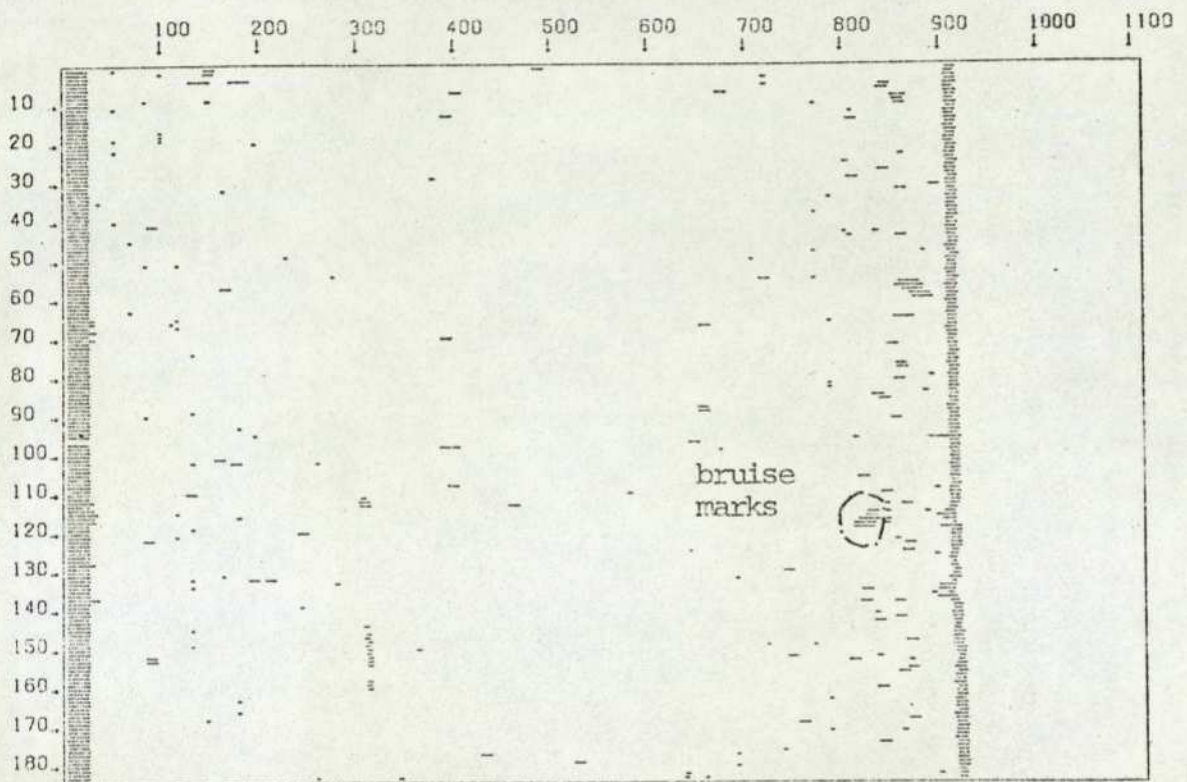












SHEET F179. MATCHED FILTER VIDEOPRINT. 14/03/76. 4 FILTERS WITH THRESHOLDS 9.0/ 10.0/ 12.0/ 15.0/

REFERENCES

1. L. Norton-Wayne, W. J. Hill, L. Finkelstein, V. Popovici. Advanced signal processing in automatic inspection. In: Proceed. IMEKO VII, 1976, London, v. 3, CTH/320/1
2. R. A. Brook, J. F. Claridge, C. Watts. Automatic inspection of flat strips in the steel industry. In: Proceed. IMEKO VII, 1976, London, AML/154/1
3. W. J. Hill. The application of pattern recognition techniques to automatic surface inspection. Instru. Syst. Centre Report, Oct., 1975
4. V. Popovici. Pattern recognition methods for classifying surface defects. Instru. Syst. Centre Report, DSS/VP/71, May, 1974
5. V. Popovici. Application of linguistics to defect classification. Instr. Syst. Centre Report, SS/VP/85, June, 1975
6. A. Aho, J. Ullman. The theory of parsing, translation and compiling (vol. 1 and vol. 2), Prentice Hall, 1972/1973
7. J. Hopcroft, J. Ullman. Formal languages and their relation to automata. Addison-Wesley, 1969
8. K. S. Fu. Syntactic pattern recognition. Academic Press, 1974
9. K. S. Fu, P. H. Swain. On syntactic pattern recognition. In Software Eng. (ed. J. Tou), v. 2, Acad. Press, 1971

10. T. G. Evans. A grammar-controlled pattern analyzer. Proc. IFIP 1968 Congress
11. W. F. Miller, A. C. Shaw. Linguistic methods in picture processing - a survey. 1968 Tall Joint Comput. Conf., 1968
12. K. S. Fu. Stochastic automata, stochastic languages and pattern recognition. J. Cybernetics, v. 1, 31-49 (1971)
13. H. C. Lee, K. S. Fu. A stochastic syntax analysis procedure and its application to pattern recognition. IEEE Trans. on Comp, July, 1972, p. 660-666
14. P. H. Swain, K. S. Fu. Stochastic programmed grammars for syntactic pattern recognition. Pattern Recognition, v. 4, p. 83-100, 1972
15. R. Bellman, R. Kalaba, L. A. Zadeh. Abstraction and pattern classification. J. Math. Anal. Appl., v. 13, p. 1-7, 1966
16. M. G. Thomason. Finite fuzzy automata, regular fuzzy languages and pattern recognition. Pattern Recognition, v. 5, p. 383-390, 1973
17. M. G. Thomason, P. Marinos. Deterministic acceptors of regular fuzzy languages. IEEE Trans. on SMC, p. 228-230, 1974
18. M. G. Thomason. Errors in regular languages. IEEE Trans. on C., v. C-23, p. 597-602, 1974
19. M. G. Thomason, R. Gonzales. Syntactic recognition of imperfectly specified patterns. IEEE Trans. on C., v. C-24, p. 93-95, 1975

20. L. Fung, K. S. Fu. Stochastic syntactic decoding for pattern classification. *IEEE Trans. on C.*, v. C-24, p. 662-667, 1975
21. W. B. Smith. Error detection in formal languages. *J. Comp. Syst. Sci.*, v. 4, p. 385-405, 1970
22. A. V. Aho, J. G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.* v. 1, p. 305-312, Dec., 1972
23. K. S. Fu, T. L. Broth. Grammatical inference: introduction and survey. *IEEE Trans. on SMC*, v. SMC-5, 1975, p. 95-111 (part I) and p. 709-723 (part II)
24. T. G. Evans. Grammatical inference techniques in pattern analysis. In *Software Engineering*, v. 2, (J. T. Tou, ed.), 1971
25. E. M. Gold. Language identification in the limit. *Inform. Control*, v. 10, p. 447-474, 1967
26. J. Horning. A procedure for grammatical inference. *Proc. IFIP Congr.*, 1971
27. S. Crespi-Reghizzi. An effective model for grammar inference. *Proc. IFIP Congr.*, 1971
28. J. A. Feldman. Some decidability results on grammatical inference and complexity. *Inform. Control*, v. 20, p. 244-262, 1972
29. F. Mirianski. Inference of probabilistic grammars. Ph.D. Dissertation, Dept. Elect. Eng. Comp. Sci., Univ. Conn., July, 1974

30. A. R. Patel. Grammatical inference for probabilistic finite-state languages. Ph.D. Dissertation, Dep. Elec. Eng. Comp. Sci., Univ. Conn., 1972
31. C. M. Cook. Grammatical inference by heuristic search. Comp. Sci. Center, Univ. Maryland, Rep. 287/1974
32. A. W. Bierman, J. A. Feldman. On the synthesis of finite-state machines from samples of their behaviour. IEEE Trans. on Comp. v. C-21, p. 592-597, June, 1971
33. R. M. Wharton. Approximate language identification. Inf. Control, v. 26, p. 236-255, 1974
34. V. Popovici. Syntactic pattern recognition system. FORTRAN software. Instr. Syst. Centre Report, Nov., 1976
35. R. A. Wagner. Order-n correction for regular languages. Comm. ACM, v. 17, nr. 5/1974, p. 265-268
36. L. Norton-Wayne, V. Popovici, W. J. Hill. Acquisition system for surface inspection data. DSS/LNW/VP/WJH/May, 1976
37. RACAL NRZ Data Format Unit. Supplement to T7000 Digital Magnetic Tape System. Technical Manual
38. DEC PDP-11 Peripherals Handbook
39. INTEL. Data catalog, 1976
40. M. E. Hoff. Designing central processors with bipolar microcomputer components. National Comp. Conf., 1975, p. 55-62